AFIPS CONFERENCE PROCEEDINGS

VOLUME 38

1971

SPRING JOINT COMPUTER CONFERENCE

May 18 - 20, 1971
Atlantic City, New Jersey

AFIPS PRESS
210 SUMMIT AVENUE
MONTVALE, NEW JERSEY 07645

Library of Congress Catalog Card Number 55-44701
AFIPS PRESS
210 Summit Avenue
Montvale, New Jersey 07645

Printed in the United States of America

Edited by Dr. Nathaniel Macon, Technical Program Chairman

CONTENTS

COMPUTING MACHINES—MENACE OR MESSIAH?—PANEL SESSION

(No papers in this volume)

IMAGE OF THE INDUSTRY—PANEL SESSION

(No papers in this volume)

THE NEW TECHNOLOGY—HARDWARE DESIGN AND EVALUATION

The DINKIAC I—A pseudo-virtual-memoried mini—For stand-alone interactive use	1	R. W. Conn
A multi-channel CRC register	11	A. M. Patel
Features of an advanced front-end CPU	15	R. B. Hibbs
Interpreting the results of a hardware systems monitor	23	J. S. Cockrum E. D. Crockett

LAW ENFORCEMENT AND JUDICIAL ADMINISTRATION—PANEL SESSION (No papers in this volume)

APPLICATIONS REQUIRING MULTIPROCESSORS

4-way parallel processor partition of an atmospheric primitive-equation prediction model	39	E. Morenoff W. Beckett P. G. Kesel F. J. Winninghoff P. M. Wolff
An associative processor for air traffic control	49	K. J. Thurber

COMPUTER AIDED MANAGEMENT OF EARTH RESOURCES—PANEL SESSION (No papers in this volume)

RESPONSIVE GOVERNMENT—PANEL SESSION

(No papers in this volume)

COMPUTERS IN TRANSPORT—FOR MANAGEMENT NEEDS OR SUPPLIERS' DELIGHT?

A computer-aided traffic forecasting technique—The trans Hudson model	61	E. J. Lessieu
Computer graphics for transportation problems	77	D. Cohen J. M. McQuillan
Real time considerations for an airline	83	J. Loo B. T. O'Donald I. R. Whiteman
A computer simulation model of train operations in CTC territory	93	D. Borch

PRESENT AND FUTURE DATA NETWORKS—PANEL SESSION

(No papers in this volume)

TERMINAL ORIENTED DISPLAYS

A general display terminal system	103	J. H. Botterill G. F. Heyne
AIDS—Advanced interactive display system	113	T. R. Stack S. T. Walker
CRT display system for industrial process	123	T. Konishe N. Hamada I. Yasuda
Computer generated closed circuit TV displays with remote terminal control	131	S. Winkler G. W. Price

COMPETITIVE EVALUATION OF INTERACTIVE SYSTEMS—PANEL SESSION (No papers in this volume)

COMPUTERS IN THE ELECTORAL PROCESS

The theory and practice of bipartisan constitutional computer-aided redistricting	137	S. S. Nagel
"Second-generation" computer vote count systems—Assuming a professional responsibility	143	C. H. Springer M. R. Alkus

MICROPROGRAMMING AND EMULATION

Evaluation of hardware-firmware-software trade-offs with mathematical modeling	151	H. Barsamiam A. DeCegama
System/370 integrated emulation under OS and DOS	163	G. R. Allred
A high-level microprogramming language (MPL)	169	R. H. Eckhouse, Jr.
A firmware APL time-sharing system	179	R. Zaks D. Steingart J. Moore

INTERACTIVE APPLICATIONS AND SYSTEMS

Designing a large scale on-line real-time system	191	S. Ishizaki
PERT—A computer-aided game	199	J. Richter-Nielsen
Interactive problem-solving—An experimental study of "lockout" effects	205	B. W. Boehm M. J. Seven R. A. Watson
TYMNET—A terminal-oriented communication network	211	L. R. Tymes
Implementation of an interactive conference system	217	T. W. Hall

COMPUTATIONAL COMPLEXITY—PANEL SESSION

(No papers in this volume)

THE EVOLUTION OF COMPUTER ANIMATION—PANEL SESSION

(No papers in this volume)

SERVING USERS IN HIGHER EDUCATION

Who are the users?—An analysis of computer use in a university computer center	231	E. Hunt G. Diehr D. Garnatz

INFORMATION AND DATA MANAGEMENT

An initial operational problem oriented medical record system—For storage, manipulation and retrieval of medical data	239	J. R. Schultz S. V. Cantrill K. G. Morgan
Laboratory verification of patient identity	265	S. Raymond L. Chalmers W. Steuber
The data system environment simulator (DASYS)	271	L. E. DeCuir R. W. Garrett
Management information systems—What happens after implementation?	277	D. E. Thomas, Jr.
A methodology for the design and optimization of information processing systems	283	J. F. Nunamaker, Jr.

COMPUTER ASSISTED INSTRUCTION

Computer generated repeatable tests	295	F. Prosser D. D. Jensen
R2—A natural language question-answering system	303	K. Biss R. Chien F. Stahl

THE NEW TECHNOLOGY—STORAGE

Performance evaluation of direct access storage devices with a fixed head per track	309	T. Manocha W. L. Martin K. W. Stevens
Drum queueing model	319	G. P. Jain S. R. Arora
Storage hierarchy systems	325	H. Katzan, Jr.
Optimal sizing, loading and re-loading in a multi-level memory hierarchy system	337	S. R. Arora A. Gallo
The TABLON mass storage network	345	R. B. Gentile J. R. Lucas, Jr.

TOPICS IN COMPUTER ARITHMETIC AND IN ARTIFICIAL INTELLIGENCE

A structure for systems that plan abstractly	357	W. W. Jacobs
Unconventional superspeed computer systems	365	T. C. Chen
High speed division for binary computers	373	H. Ling
A unified algorithm for elementary functions	379	S. Walther
A software system for tracing numerical significance during computer program execution	387	H. S. Bright B. A. Colhoun F. B. Mallory

SOFTWARE LIABILITY AND RESPONSIBILITY—PANEL SESSION

(No papers in this volume)

VENTURE CAPITAL—FINANCING YOUNG COMPANIES—PANEL SESSION (No papers in this volume)

FROM THE USER'S VIEWPOINT—PANEL SESSION

(No papers in this volume)

PERIPHERAL PROCESSING—PANEL SESSION

(No papers in this volume)

COMPUTER PICTORICS

Automated interpretation and editing of fuzzy line drawings......... 393 S. K. Chang

Computer graphics study of array response......................... 401 G. W. Byram, G. V. Olds, L. P. LaLumiere

Computer manipulation of digitized pictures....................... 407 N. Macon, M. E. Kiefer

AN INTERNATIONAL VIEW—PANEL SESSION

(No papers in this volume)

SIMULATION OF COMPUTER SYSTEMS

The design of a meta-system...................................... 415 A. S. Noetzel

An interactive simulator generating system for small computers...... 425 J. L. Brame, C. V. Ramamoorthy

APPLICATION OF COMPUTERS TO TRAINING—PANEL SESSION

(No papers in this volume)

THE NEW TECHNOLOGY—DIAGNOSTICS AND RECOVERY

Multiband automatic test equipment—A computer controlled check-out system.. 451 T. Kuroda, T. C. Bush

Coding techniques for failure recovery in a distributive modular memory organization.. 459 S. A. Szygenda, M. J. Flynn

Recovery through programming system/370....................... 467 D. L. Droulette

On automatic testing of one-line, real-time systems................ 477 J. S. Gould

THE NEW TECHNOLOGY—SYSTEMS SOFTWARE

PORTS—A method for dynamic interprogram communication and job control.. 485 R. M. Balzer

Automatic program segmentation based on boolean connectivity..... 491 E. W. Ver Hoef

Partial recompilation.. 497 R. B. Ayres, R. L. Derrenbacher

PL/C—The design of a high-performance compiler for PL/I......... 503 H. L. Morgan, R. A. Wagner

GPL/I—A PL/I extension for computer graphics................. 511 D. N. Smith

ETC—An extendible macro-based compiler......................... 529 B. N. Dickman

THE COMPUTER PROFESSIONAL AND THE CHANGING JOB MARKET—PANEL SESSION

(No papers in this volume)

THE NEW TECHNOLOGY—FILE ORGANIZATION

A file organization method using multiple keys.................... 539 M. L. O'Connell

Arranging frequency dependent data on sequential memories........ 545 C. V. Ramamoorthy, P. R. Blevins

Associative processing of line drawings........................... 557 N. J. Stillman, C. R. Defiore, P. B. Berra

THE NEW TECHNOLOGY—COMPUTER ARCHITECTURE

The hardware-implemented high-level machine language for symbol...	563	G. D. Chesley W. R. Smith
SYMBOL—A major departure from classic software dominated von Neumann computing systems................................	575	R. Rice W. R. Smith
The physical attributes and testing aspects of the symbol system.....	589	B. E. Cowart R. Rice S. F. Lundstrom
SYMBOL—A large experimental system exploring major hardware replacement of software......................................	601	W. R. Smith R. Rice G. D. Chesley T. A. Laliotis S. F. Lundstrom M. A. Calhoun L. D. Gerould T. G. Cook

EDUCATIONAL REQUIREMENTS FOR SYSTEMS ANALYSTS

A semi-automatic relevancy generation technique for data processing education and career development..............................	617	J. D. Benenati
An architectural framework for systems analysis and evaluation......	629	P. Freeman

COMPUTER ACQUISITION—PURCHASE OR LEASE—
PANEL SESSION (No papers in this volume)

COMPUTATION, DECISION MAKING, AND THE
ENVIRONMENT—PANEL SESSION

(No papers in this volume)

The DINKIAC I—A pseudo-virtual-memoried mini—For stand-alone interactive use

by RICHARD W. CONN

University of California
Berkeley, California

INTRODUCTION

The past three years have witnessed the development and sale of a large and unanticipated number of small general purpose digital computers. These machines—the mini-computers—originally intended for real-time use in applications such as production control, now serve many diverse functions, ranging all the way from data buffers to the central processing units of small time-sharing systems. One trade journal even reports a sale to a home hobbyist claiming that initial costs are comparable, and upkeep less, than for other "recreational" equipment such as boats or sports-cars.

Several manufacturers have offered a basic machine with four thousand eight or twelve bit words, and with teletype I/O, for under ten thousand dollars.[1,2] Because of keen marketing competition and recent developments in integrated circuit technology these prices are continuously dropping. Memory costs, however, have not kept pace with the decreased logic costs brought about by the new IC's. Before truly spectacular price drops can be made the cost of memory must be reduced.

Memory in the above context evokes images of undelayed random addressability by word, or, more specifically, of magnetic cores. Yet if we consider computing systems generally, core memory represents but a small percentage of a typical installation's total storage. High core fabrication costs have led—in all but the tiniest systems—to the utilization of memory hierarchies. Devices most commonly comprising these hierarchies are, of course, the familiar magnetic cores, drums, disks, and tapes.

The questions to be examined in this study are: How cheaply can a machine adhering to storage hierarchy principles be built? What will it look like? and What good is it? To be in any position for viewing either of the others we must first address ourselves to the question, "What will it look like?" To do this the design of the Dinkiac, a machine meeting the implied constraints, will be summarily described. Explicitly stated these constraints include cheapness, component availability, and completeness in the sense that the user will not be required to purchase additional hardware. Once the Dinkiac design has been outlined, its usefulness can be assessed, its performance and architecture confirmed by simulation; construction details and alternate features may be presented, and its cost ascertained.

THE DINKIAC

Physically, the Dinkiac will appear as a typical keyboard—cathode-ray-tube display terminal. It will consist of a typewriter-like, 64 key, keyboard; a small CRT with a display capability of up to 84 characters presented in seven rows of twelve characters each; a row of lamps and switches; a single track low quality tape cassette recorder; four magneto strictive delay lines—all packaged together with the necessary register and logic components.

With its 16 bit word size the Dinkiac will appear to a machine language programmer as one of the larger minis. A word will represent data as either a single fixed point binary fraction in two's complement form, or as two eight bit character bytes, the last 6 bits of each conforming to USASCII standards.

Each instruction will comprise one full word in a fixed format with the first four bits (0-3) for the operation code; bit 4 a possible index register designator; bit 5, an indirect bit; bits 6 and 7, a page (delay line) address; and the last eight bits (8-15), the address within a page of one of 256 sixteen bit words.

Main memory will be made up of four magnetostrictive delay lines each storing 4096 bits. These lines will have a bit rate of two megahertz for a maximum access of a little over two milliseconds or an average access of approximately a millisecond. Each of these lines with a capacity of 256 words will be said to store a page of information. Processing may take place in any one of these lines concurrent with an exchange of information between secondary storage and some other line, not including the first, or page zero line. (Many readers will challenge the wisdom of choosing delay lines over shift registers. The latter has a speed advantage as well as the greater potential for cost reduction, matching decreases in the other IC's. There are, however, no large cheap shift registers currently available, and since it is our intention to show that a cheap instrument can be immediately constructed from off-the-shelf components, we are forced to choose the moderately priced and readily available delay line.[3])

The previously noted cassette recorder will provide secondary storage; a single tape retaining information in one of 128 blocks of 256 words each. Bit storage and retrieval rates will be around three kilohertz fixing page transfers at around one and a half seconds. The source and adequacy of these speeds will be discussed in the simulation section.

As originally conceived, the Dinkiac included hardware for automated page swapping, thus inspiring the notion—echoed by the paper's title—of a virtual memory machine; the virtual space being the size of the tape or more accurately the number of tape blocks times the number of words in a block, i.e., 32*K* Dinkiac words. Memory addressing was to have employed a page register-associative search scheme which operated in the following manner: Three (because page zero is not swappable) seven bit page address registers were loaded under program control. An instruction pointing to one of these registers (with the delay line address bits) referred to the tape block indicated by that register's contents. The instruction's address field indicated one of 256 words within the page. The requested page may or may not have been physically present in some delay line. Three seven bit registers were to compare their contents with that of the indicated page register and, if found, switch in the associated line. Because of the great disparity between word access and logic switching time the hardware for this associative search need not have been fast. If a specified page was not in any of the delay lines it was to have been retrieved from the cassette and stored in some line according to an algorithm which first checked sequential delay lines to find one in which the dirty bit had not been set. (The dirty bit was set—by the memory store signal—for any line which had been written into.) If all lines were dirty one line was selected and written out before the requested page was fetched. If program execution was delayed awaiting the fetched page, the program counter was stored and control transferred to a preset interrupt location. The described addressing scheme is shown in Figure 1.

Unfortunately this automation accounted for more than 20 percent of the total logic costs. In addition the primitive page swap algorithm may have proven unsatisfactory and required additional commands or even a complex sequence initiated from a read-only memory. In any event, the logic has been reduced to near minimum and any automated page swapping will now be under software control.

It is assumed that this operating system software will minimally include a keyboard input and display program as well as a cassette directory and search routine. Transfer instructions and busy flags will facilitate its operation and attempts to execute instructions from pages in the process of being swapped will still effect a transfer of control to the interrupt location. The inclusion of a fixed memory interrupt location is the primary reason for not swapping delay line zero.

While the cassette's primary function is to provide intermediate storage it also doubles as a cheap and convenient source of input/output. Initial input, however, is entered by way of the alphanumeric keyboard. Depressing a key will enter an encoded character into an eight bit keyboard buffer, turn off a console lamp, and set a one bit flag register. This flag may be interrogated by a running program and is reset—along with the lamp—by transferring the contents of the keyboard buffer to the accumulator. Striking a key will not enter a new character into the buffer while the flag is set.

Visual output is direct to a CRT from the first 42 word locations of the zero or non-transferable delay line. These words are gated sequentially in pairs (modulo 21) into a 32 bit output buffer on each cycle through memory. The low order six bits in each of the four bytes are, in turn, used as an input to a small read-only memory. This memory in conjunction with an appropriate counter and shift register provides serial output for modulating the CRT's "*Z*" or intensity input. These components together with a character, line, and row counter, and two deflection amplifiers and digital to analog converters, constitute the output device. It should be noted that the memory itself pro-

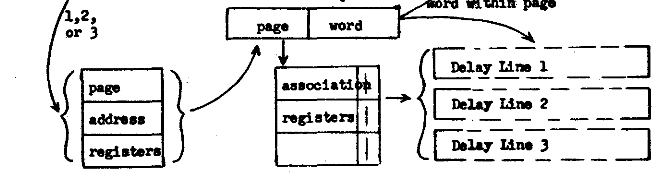


Figure 1

vides for display buffering and that all information is retained in character format. Scan conversion from the 32 bit buffer is performed as needed. Since it is possible to change characters in the output portion of memory before they have actually been displayed, it is anticipated that display programming will be handled as a function of the machine's interactive use. The most obvious example is provided by the displaying of a keyboard input message.

An operation panel located between the keyboard and display tube includes 'power on' and 'interrupt' toggles, 'start' and 'go' buttons, a five position rotary switch, and eighteen display lamps. The start button clears all registers except the program counter—into which the start address $(64)_{10}$ is forced—and loads tape block zero into delay line zero. Once block zero has been read the machine will begin instruction readout from the start location. Depending upon the state of the machine, depressing 'go' will either initiate a read of the next instruction—from the location currently specified by the program counter—or begin the instruction execution. The interrupt toggle will set or reset an interrupt step mode flip-flop. When set, this flag will force a machine halt after each instruction read and after each instruction execute. If 'go' is depressed while halted following a read, the machine will proceed to the execute. If it is depressed after the execute—and without a reset from the toggle—the program counter will be stored and the next instruction will be taken from the interrupt location.

The interrupt arrangement permits program stepping in one of two ways. For possible machine malfunction or difficult logical sequences the display lamps may be used in conjunction with the rotary switch to inspect the contents of the major processor registers. For more routine debugging, the user may choose to enter a subroutine which will convert and store relevant registers for subsequent display on the CRT. This mode will allow him to view, for example, the contents of the accumulator and the program counter—in any format he has chosen—at every other push of the 'go' button.

Given the above design it should be helpful to briefly consider a couple of the Dinkiac's unique operational and programming aspects. First and most obvious is the procedure imposed by keyboard limited input. Since all programs must be typed-in, it is probable that the typical user will be concerned only with conversational routines such as JOSS, FOCAL, or conversational BASIC. These processors should be structured in such a way that an anticipated routine will be scheduled into a delay line and ready for use. For example, an interactive algebraic processor could be segmented such that routines for matching, scheduling, and arithmetic operations are seldom or never swapped, while more complex numerical subroutines are arranged in a hierarchy of priorities with the most common (square root, sine, . . .) at the top and those seldom used (matrix operations, error exceptions and comments, . . .) at the bottom. While the software designer must try to segment these programs for the minimum swapping delay, it should be borne in mind that in conversational systems an occasional delay of several seconds is no cause for concern.[4] Balance between computation and user interaction is the significant factor.

It is hoped that by now the reader—having considered the design overview together with the cursory remarks relating the machine with certain time-sharing concepts—will have acquired sufficient intuition to answer, for himself, the third of our questions, "What good is it?" or more graciously put "What market does the Dinkiac serve?" For our part we will start with the statement that anyone now using a desk calculator can—for the same price and without sacrifice of calculator speeds or functions—enjoy the additional benefits of a completely general purpose digital computer. Additionally, the machine will provide a single user with a computing experience not unlike one he would receive at a time-sharing terminal. That is, for highly interactive work he can expect extremely fast replies with respect to his own response time. For compute bound requests, such as compilations or iterative numeric calculations, he should suffer no greater frustration than that engendered by a small well used time-sharing system. It is accurate to add that for the same jobs these periods of delay would compare favorably with a mini time-sharing system.

Because the tape cassette secondary storage will double as a fast I/O device a library of special purpose application cassettes can also be marketed. Examples are: BASIC for the schools; 'desk calculator' for small businesses; and, 'preparing your federal tax return' for the 'home hobbyist.'

SIMULATION

Our concern with a computer simulation is twofold, aiming first at determining the Dinkiac's gross architectural configuration, that is, the number and length of its delay lines, and second, at obtaining some sense of its overall performance. GPSS/360 (IBM's General Purpose System Simulator for the 360 series) was chosen for this task—both for its ease of use and its ready availability.[5]

For a simulation to serve its intended purpose the assumptions upon which it rests must be both valid and appropriate. The assumptions underlying this simulation are of two kinds, the first has to do with hardware component speeds and may be based on the price quotes of a number of manufacturers, the second requires a knowledge of program behavior and is far more tenuous. An early discussion of equipment characteristics will provide a foundation for the subsequent consideration of these less structured issues.

Magnetostrictive delay lines are offered in models with delays of up to 10 milliseconds at the maximum or 2MHz bit rate. Prices vary only slightly over the range with the longest lines (in quantity lots) costing less than 10 dollars more than the shortest. Since prices are typically constant up to a delay of around 2.5ms, a $4K$ bit line costs no more than one with half that capacity. Restricting the choice to sizes which facilitate binary addressing, these delays and bit rates imply that lines of up to $16K$ bits are feasible.

Because the Dinkiac is a single address machine all non-jump instructions must be taken sequentially, and if operands are positioned properly those with fewer than 128 memory fetches will be executed at delay line speed. (Switching time, even for slow transistor logic, can always be accomplished during the delay line to register transfers and may therefore be completely ignored.) A straight line program, then, will be executed at about the product of the line speed times the number of instructions. For the Dinkiac we have described—with its $4K$ bit line—this would amount to approximately 500 instructions/second while a $2K$ line would double the rate and one with $16K$ bits would cut it to a low of 125 instructions/second.

The tape cassette market is less stable than the market for delay lines and one may find prices ranging all the way from under thirty dollars to 100 times that price. The machines on the low end are intended for audio use while those at the other are designed for the reliable high-speed transfer of digital data. Advertised speeds for the expensive instruments give writing rates at under 10,000 bits/second with reading rates to 20,000. Experiments indicate that digital (square wave) recording on cheap audio equipment can be successful at speeds of two to two and one-half thousand bits per second. Specifications from a number of manufacturers marketing inexpensive recorders indicate that for under 100 dollars one can conservatively assume the following characteristics: (1) Read/write speed of 3.75 ips with a recording density of 800 bpi (bit serial recording) for a transfer rate of 3000 bps; (2) Search speed (fast forward and rewind) of 75 ips; (3) Start/stop time of 60 ms; and (4) Inter-record gap of ½ inch.

The properties given above will be used in the simulation, and to reinforce their conservative character, cassette page transfer times will always include time for the transfer of a full half inch inter-record gap as well as the times for both starting and stopping the tape. This caution also allows for any timing oversight arising from the recording technique, which we have assumed will follow teletype signal transmission methods, i.e., asynchronously, with a start pulse followed by data followed by completion pulses. To time a $16K$ block transfer, then, we will assume that 16,384 data bits plus a 400 bit equivalent inter-record gap are transferred at a rate of 3000 bps to which 120 ms, start and stop time, are added. That is, block transfer time = (((line size+400)/3000)+.120) seconds.

Tape search time will be based upon a full tape capacity of half a million (2^{19}) information bits. (Later we will include some results gathered when providing for 256 blocks of the larger page sizes, i.e., for tapes of 2^{20} and 2^{21} bits.) Tape length, not including inter-record gaps, is approximately 655 inches—2^{19} bits at 800 bpi. Total search time will be determined by adding—to this length—a half inch for each record and dividing by the 75 inches/second rate, or, total search time = ((655+(no. of blocks on tape/2)/75) seconds.

We may now specifically formulate three questions we wish our simulation to answer: (1) What is the best page size? (2) How many lines are necessary for satisfactory performance? and (3) How will the Dinkiac compare with other machines? Given some assumption regarding the number of jumps expected during the execution of a program plus the anticipated distance of the jumps—i.e., what percentage of jumps will remain within 10 words of the current address, 20 words, etc.—it is possible to run simulations based upon the given transfer rates to obtain meaningful results for the first two of these questions. If, however, we wish to relate the Dinkiac's performance to that of other machines we will need some standard.

Fortunately, such a standard exists in terms of average instruction time. Given anticipated percentages for each instruction type and applying these percentages to the machine's actual instruction execution times, we can determine the time required for an 'average' instruction. Gibson has provided us with a set of such percentages by tracing 55 IBM—7090 programs involving 250 million instructions.[6] The traced programs were comprised of 30 FORTRAN source programs, 5 machine-language programs, 10 assemblies, and 10 compilations. Gibson's set of percentages, called the Gibson mix, has been used in many machine comparison studies. Because the Dinkiac has no floating point hardware, approximate averages for subroutine execution times will be given for the floating point instructions. The same will be done for multiplies and divides. The Gibson mix programs were scientific and give a conservative average with respect to a similar mix projected from the data processing field. Figure 2 is a table of Dinkiac instructions, 'worst' case times, and the loosely corresponding Gibson percentage. Execution times are given as delay-line revolutions.

Because subroutines are included, a single Gibson mix instruction must represent more than one of the Dinkiac's. Specifically, 87 percent are one to one, 7.7 percent are ten to one, and 5.3 percent are twenty to one. There are therefore 2.7 Dinkiac instructions to each of Gibson's and the average execution time for these 2.7 instructions is 8.6 revolutions. At 2.7 words for a Gibson instruction, each line of the 256 words/line machine we presented is capable of 'storing' 94.8 Gibson instructions. Similarly a 128 word line will contain 47.4 instructions, and so on. We have greatly simplified the remaining calculations by assuming a Gibson instruction size of 2.5 words and line lengths which are integral multiples of that number—forcing the use of 125 for the 128 word line, 250 for the 256 word line, etc.

Returning now to the still unspecified assumptions regarding program behavior, we find the question of jumps partially resolved by the Gibson mix. The mix assigns a 16.6 percent likelihood to the 'Test and Jump' instruction. We will assume that the jump is taken half this number, or 8.3 percent. To this we must assign some number of jumps to compensate for those subroutine loops incurred by our superimposition of the Gibson instructions over the Dinkiac's. Suppose 100 Gibson (270 Dinkiac) instructions are executed. Of the 270 Dinkiac instructions, 8 will be for multiply and divide, 69 for floating add and subtract, and 106 for floating multiply and divide. Assuming a five instruction loop for the first two instruction types and a ten word loop for the last, we will arrive at 26 jump instructions or slightly less than 10 percent of the instructions executed. We may further assume that these subroutines will be retained in the zero delay line and that return jumps will be back to the lines from which the subroutines are called. The model reflects this analysis.

The question of how far each jump goes with respect to the current program address counter is not easily answered and is closely allied to the question of how often must a new page be fetched. Until some study is made—similar to Gibson's but with just this aim—or, until studies of time-sharing systems provide further insight into page swapping behavior, no well-grounded assumption can be made. We will postulate that of the jumps taken—not including the 10 percent headed for delay line zero—50 percent will remain in the line they are at while the remaining half will go to the lines following with percentages of 50 percent, 37.5 percent, and 12.5 percent, respectively. Here the delay line sequencing is considered circular. This jump distance assumption is, of course, inconsistent with the varying line size and favors short lines. We will compensate for this advantage by making a near worst case assumption regarding page swapping, namely, that a new page be fetched once for every straight line pass through the memory.

We are now in a position to present details of the model. Each GPSS 'transaction' will represent either ten Gibson instructions or a signal to initiate the operation of some given line or tape. Each delay line consists of a holding 'queue' for the transactions, a memory 'facility' and a 'storage' capable of accommodating the appropriate number of instructions for a specified line size. To avoid simulating the simultaneous execution of instructions in more than one line, only sufficient transactions to queue up for a single line are generated at any one time. A transaction entering a facility (one of the delay lines) from a queue 'seizes' that facility precluding its use by any other transaction. An appropriate number (25 for 10 Gibson instructions) of instructions is 'entered' into the line storage and the total storage entries compared with the line capacity. If the storage is full, it is reset to zero; the facility is released; a transaction is removed from the queue; and new transactions are created for the next memory line. If the storage is not full, 18.3 percent of the transactions go to a jump instruction sequence where the clock is advanced 10 'jump' times and the transaction is entered into holding buffers according to the previously discussed jump distribution. In the 81.7 percent non-jump cases, the clock is advanced by the time required for ten line revolutions times a GPSS 'function' which randomly chooses (on the basis of a given bias—in this case the Gibson percentages) the number of revolutions. The facility is then released to allow for another entry from the queue; a transaction is removed from the queue; and ten transactions (instructions) are terminated.

Except in the case of the zero line, the completion of each line triggers a set of transactions for the next in a round-robin fashion with the last line triggering the first. Thirty percent of the completions from the zero line may additionally store a transaction in one of the holding buffers to simulate the subroutine return jumps. A counter at the end of the last line starts an end-of-job sequence which continues the program for only those lines which have items in their holding buffers. Completion of the last line also sends a transaction into the tape queue. Transactions in the tape queue seize a tape facility and then randomly 'pre-empt' one of the swappable delay lines. A pre-empted line is held until 'returned' and is precluded from seizure or use by any other transaction. The tape and pre-empted line times

INSTRUCTION	TIME IN REVOLUTIONS (Worst case for nonsubroutines)	GIBSON PER-CENTAGE
Load and Store Add and Subtract Logical	1.5	38.9
Multiply and Divide (10 word subroutine)	50.	.8
Floating Point Mult. and Div. (single precision) (20 word subroutine)	100.	5.3
Floating Add and Sub. (single precision) (10 word subroutine)	25.	6.9
Shifts and Register	1.	9.7
Test and Jump	.5	16.6
Index Search or Compare	2.	21.8

Figure 2

are advanced by one block transfer time and also, when appropriate, by tape search time. Simulations may be either "non-predictable"—in which case time to search half of the tape plus or minus any random interval up to that same amount is always applied—or, they may be "predictable." In the predictable or "75 percent predictable" runs it is assumed that the tape will have been correctly prepositioned in all but 25 percent of the transfers. At the completion of these tape advance times the pre-empted line is returned and the tape released. A general program flow is given in Figure 3.

Two results quickly emerged from the simulations, most apparent is the ruling out of either very short, or very long lines. The second, while less glaring, verifies the adequacy of a four line machine. It is tempting to continue the simulations with a greater number of storage lines—and when shift register prices fall this may prove feasible. Meanwhile, price considerations for this study dictate that the number be kept as small as possible. Upper and lower performance bounds were found by running the simulation with either no, or with complete, tape buffering.

The number of instructions executed during any one simulation varies slightly due to the randomness of the jumps. All runs, however, simulate the execution of close to 12,300 instructions. Execution time varies from a lower bound of two plus minutes (120,391 ms) to an upper bound of almost 12 minutes (707,961 ms). A table showing the total execution time in milliseconds for thirty-one simulations is given in Figure 4. Figures 5 through 8 are graphs of the four general cases: four lines both predictable and non-predictable and the same for three lines. The dotted lines in Figures 5 and 6 are the results of allowing the number of tape information bits to double once for the 256 word block and twice for the 512 word block. That is, to maintain the tape block count at 256. Each graph includes upper and lower bounds in addition to the simulation's finding for the particular case. The graphs argue convincingly for the 256 word page size, and yield insight into the nature of the balance between instruction execution and page transfer times.

DESIGN SPECIFICS AND OPTIONS

Sufficient detail to familiarize the reader with the Dinkiac's peculiarities was given in an earlier section. Here we will add a few design particulars, as an aid to cost estimation, and present some significant options.

The Dinkiac is designed around a five register bus in a manner typical of the minis. Signals from decoded instructions, together with outputs from a sequencer

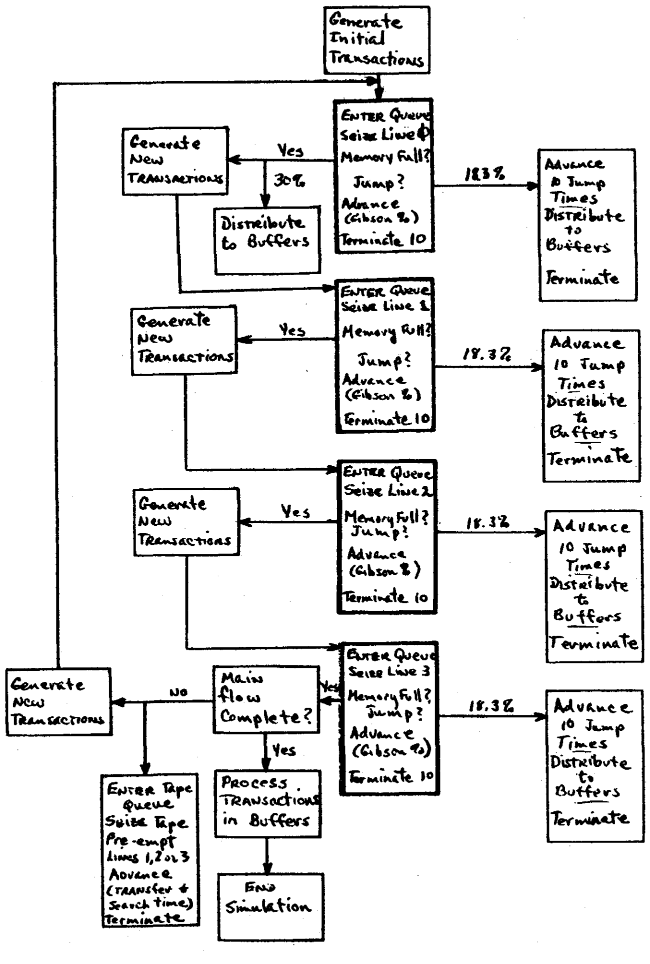

Generate Initial Transactions
Enter Queue Seize Line Φ Memory Full? Jump? Advance (Gibson %) Terminate 10
Yes
30%
Generate New Transactions
Distribute to Buffers
18.3%
Advance 10 Jump Times Distribute to Buffers Terminate
Enter Queue Seize Line 1 Memory Full? Jump? Advance (Gibson %) Terminate 10
Yes
Generate New Transactions
18.3%
Advance 10 Jump Times Distribute to Buffers Terminate
Enter Queue Seize Line 2 Memory Full? Jump? Advance (Gibson %) Terminate 10
Yes
Generate New Transactions
18.3%
Advance 10 Jump Times Distribute to Buffers Terminate
Enter Queue Seize Line 3 Memory Full? Jump? Advance (Gibson %) Terminate 10
Yes
Main flow Complete?
No
Generate New Transactions
18.3%
Advance 10 Jump Times Distribute to Buffers Terminate
Enter Tape Queue Seize Tape Pre-empt Lines 1,2 or 3 Advance (transfer + Search time) Terminate
Yes
Process Transactions in Buffers
End Simulation

Figure 3

CASE	128 Word Page	256 Word Page	512 Word Page
Lower Bound	120,391 ms	234,921 ms	479,801 ms
4 lines, 2^{19} bits 75 percent predictable	217,246	245,853	505,688
4 lines, 2^{19} bits non-predictable	440,482	324,034	506,057
4 lines, 256 pages 75 percent predictable	—	278,479	552,542
4 lines, 256 pages non-predictable	—	454,316	695,778
4 lines, Upper Bound 75 percent predictable	285,037	331,847	549,388
4 lines, Upper Bound non-predictable	602,615	472,056	586,818
3 lines, 2^{19} bits 75 percent predictable	289,290	296,161	498,685
3 lines, 2^{19} bits non-predictable	547,898	402,066	532,240
3 lines, Upper Bound 75 percent predictable	372,937	364,044	583,955
3 lines, Upper Bound non-predictable	707,961	559,366	681,738

Figure 4

and the storage completion lines, determine register gating and the consequent bus information. The machine's instruction set should prove helpful in conveying an intuitive notion of its logical complexity, and is given in Figure 9. Codes in that figure are in hexadecimal unless otherwise shown. '*A*' designates the accumulator; '*CB*' the carry bit; '*M*' the memory; '*MBR*' the memory buffer register; '*P*' the program counter;

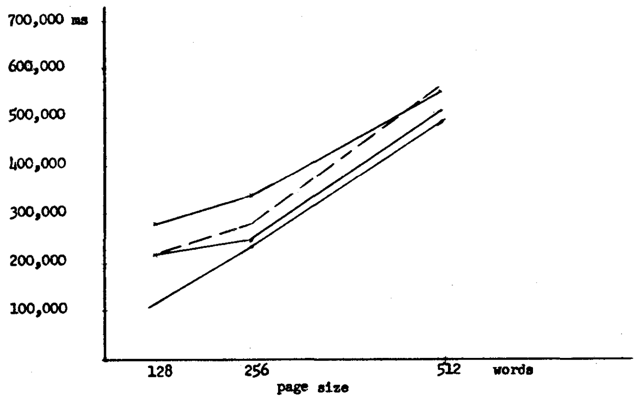


Figure 5—Four line—75 percent predictable

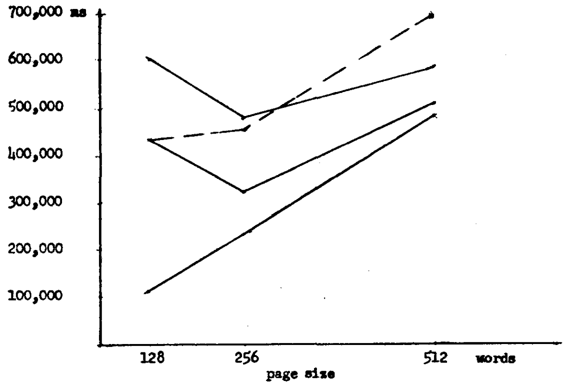


Figure 6—Four line—non-predictable

and, '*Y*' a memory address. Dinkiac word size and instruction format allow for the expansion and modification of this basic instruction set in many ways. For example, an index register may be added, or shifts modified to shift by some specified amount. Multiply and divide logic, too, could be included, and while these instructions might violate the spirit of the machine, they could easily be executed within a single memory cycle.

As implied in the simulation section, the number of delay lines can be increased with only a minor modification to the memory addressing scheme. In this case, the storage lines would continue—as they are now—to be synchronized with a single counter. Such a change could be expected to improve performance by increasing the data transfer-program execution overlap, but it would not alter the sequential instruction time which you may recall is roughly 500 instructions/second for the 256 word/line machine. Recall also that the two MHz bit rate allows for an information exchange between the memory buffer register and the chosen delay line in eight microseconds. This speed would allow the execution of non-memory referencing instructions from contiguous memory locations to proceed at the rate of 125,000 per second for a phenomenal increase of 250 times. A major factor contributing to the Dinkiac's easy circuit realization, however, lies in the difference between memory and switching speeds, and it is this great disparity that allows us to almost disregard the latter. If we wish the increased speed without altering this principle—which also enables us to purchase the cheapest logic components—we must provide both double memory buffer registers and the logic for their utilization. This type of speed-up must be carefully priced and reviewed in the light of the simulation results.

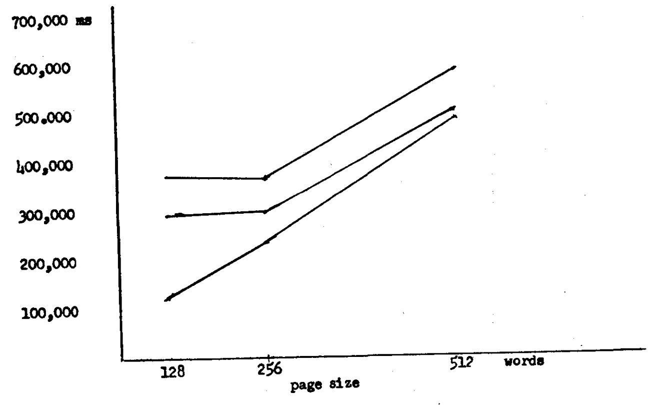


Figure 7—Three line—75 percent predictable

COSTS AND CONCLUSIONS

While the probability is high that any manufacturer seriously considering marketing such a device is already in either the small machine, display terminal or some related business—an instructive way to garner a sense of cost is to consider a prototype builder with no such association but who can avail himself of quantity prices for off-the-shelf items. If we assume a two to one gate to flip-flop ratio—not unrealistic for the proposed serial operation—meaningful logic costs can be ascertained by a simple count of single bit storage registers. Itemizing all registers—not integral with some other priced item (as, for example, the delay line input gates, . . .)—we arrive at a count of less than 200. This count is conservative, allowing bits for miscellaneous control and making no attempt to share or minimize the number or size of the registers.

A notion of dollar value can be ascribed to the count by using quantity prices for standard off-the-shelf TTL gates from leading suppliers. Such an assignment comes to $2.20 per bit where the flip-flop price is $1.60 and the gate cost $.30. This approach is again conservative taking into account no non-standard gates and using no very slow, but adequate, logic. Computed in this way the logic price to a backyard builder with connections is $440.

Similarly pricing the other components puts the delay line memory (four, 4096 bit lines) at $400 (in large quantity); the cassette recorder at $100; the keyboard at $75; the CRT and related components at $175(the display generator including read-only memory is available in lots of over 25 for less than $100); a power supply at $120; and a crystal clock at $100. The total, then, including logic is $1410. It is reasonable to expect that quantity costs to a manufacturer—including labor—would be a good deal less than this amount. We may note in this respect that currently

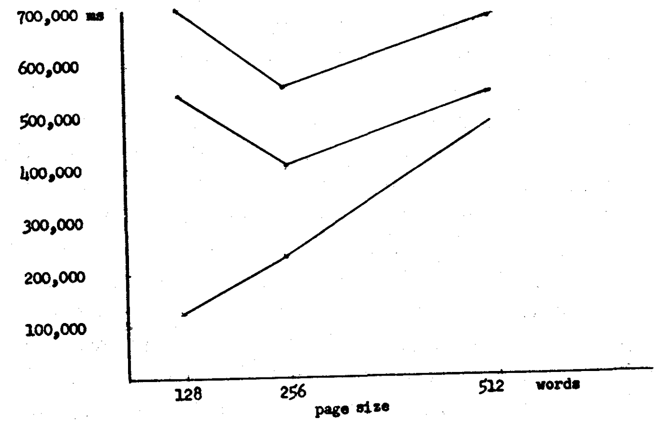


Figure 8—Three line—non-predictable

DINKIAC INSTRUCTIONS

Memory Reference

STO	Y	1xxx	$A \rightarrow M_Y$
ADD	Y	2xxx	$A + M_Y \rightarrow A$
SUB	Y	3xxx	$A - M_Y \rightarrow A$
JMP	Y	4xxx	$Y \rightarrow P$
JAM	Y	5xxx	$Y \rightarrow P$, if $A < 0$
JAZ	Y	6xxx	$Y \rightarrow P$, if $A = 0$
JSP	Y	7xxx	$P+1 \rightarrow M_Y$, $Y+1 \rightarrow P$
LDA	Y	8xxx	$M_Y \rightarrow A$
AND	Y	9xxx	$A \wedge M_Y \rightarrow A$
ISP	Y	Axxx	$M_Y + 1 \rightarrow M_Y$, if $M_Y = 0$ then $P+1 \rightarrow P$
JCB	Y	Cxxx	$Y \rightarrow P$, if $CB = 1$

Non-Memory Reference

NOP		0000	No Operation
HLT		0001	Halt
SNI		0002	$P+1 \rightarrow P$, if Interrupt Flag $\neq 1$
SNK		0003	$P+1 \rightarrow P$, if Keyboard Flag $\neq 1$
CLA		002-	$0 \rightarrow A$
CMA		003-	$2^C(A) \rightarrow A$
CLC		004-	$0 \rightarrow CB$
CMC		005-	$2^C(CB) \rightarrow CB$
LAK		006-	Keyboard Buffer $\rightarrow A_{8-15}$
LAB		007-	$MBR \rightarrow A$
SHR		008-	Shift CB and A right 1
SHL		009-	Shift CB and A left 1
RTR		00A-	Rotate CB and A right 1
RTL		00C-	Rotate CB and A left 1
RLR		001-	Rotate A, 8
SCO	Y	04xx	If Cassette not Busy, $P+1 \rightarrow P$ and Search Cassette 0 for tape page xx.
RCO		$0(10xx)_2$--	If Cassette not Busy, $P+1 \rightarrow P$ and Read tape 0 into memory page xx (where $xx = 1$, 2, or 3).
WCO		$0(11xx)_2$--	If Cassette not Busy, $P+1 \rightarrow P$ and Write tape 0 from memory page xx (where $xx = 1$, 2, or 3).

Figure 9

advertised prices for display terminals are as low as $1500, and include all Dinkiac components excepting three delay lines (the displays have one), a cassette recorder, and computer logic. This price, incidentally, includes beautiful packaging. Suppose we add to the $1500 the excluded items, priced as above, for a grand total of $2340. There is nothing to indicate that a Dinkiac cannot be profitably marketed for under $3000.

This report has attempted to show that a general purpose digital computer—suitable for a large class of users, including those in small businesses and engineering firms, schools, and even private homes—can be built to market for a price near the low end of the desk calculator range. A GPSS simulation has shown the optimum memory length to be the one in which time for the execution of a page of instructions is closely matched with tape block transfer time, and has confirmed the adequacy of four lines, even while assuming highly unfavorable operating parameters. Additionally, by modeling with "Gibson instructions," we were able to acknowledge that the Dinkiac—while short on "bandwidth" in comparison with large machines—is certainly adequate for its intended purpose.

REFERENCES

1 D J THEIS L C HOBBS
Mini-computers for real time applications
DATAMATION No 39 March 1969

2 J W COHEN
Mini-computers
MODERN DATA No 55 August 1969

3 J H EVELETH
A survey of ultrasonic delay lines operating below 100 Mc/s
IEEE Proc Vol 53 No 10 October 1965

4 R B MILLER
Response time in man-computer conversational transactions
AFIPS Conf Proc Vol 33 p 267 1968

5 G GORDON
A general purpose systems simulation program
EJCC Proc p 87 1961

6 J J CLANCY
Notes on the 'bandwidth' of digital simulation
SIMULATION Vol 8 No 1 January 1967

A multi-channel CRC register

by ARVIND M. PATEL

IBM Laboratories
Poughkeepsie, New York

INTRODUCTION

The **C**yclic **R**edundancy **C**heck (CRC) is extremely efficient and well suited for error detection in transmission, retrieval or storage of variable length records of binary data. The cyclic check is capable of detecting nearly all patterns of error with almost negligible amount of redundancy. For example, a 16-digit (2 bytes) CRC character will detect all error-bursts of length 16 or less and better than 99.99 percent of all other error-bursts in binary records of *any* length. For average record length of 1000 bytes this amounts to less than 0.2 percent redundancy. Peterson and Brown's paper[4] is an excellent exposition on the subject of error detection with cyclic codes.

A linear feedback shift register is essentially the only hardware needed for encoding and decoding variable length binary data for error detection by means of CRC character. The CRC character is generated by serially shifting the binary information into the feedback shift register as it is transmitted. The CRC character is then transmitted on the same channel at the end of the information sequence. At the receiver, the received sequence is processed in the same manner. The generated CRC character is compared with the received check character for detection of any errors in the received message. The number of digits in the CRC character determines the checking capability of the code and, in general, equals the number of stages of the encoding and decoding shift register.

Oftentimes, the information is transmitted in parallel, a byte at a time, on a multi-channel system. The message formats of serial and parallel systems are shown in Figure 1. For the multi-channel system, one could provide a separate CRC character for each channel using one shift register per channel. This, however, increases redundancy and cost. Furthermore, if serial and parallel formats are used alternatively in various parts of the total data processing system, it becomes imperative to use compatible hardware which produces same CRC check.

This paper presents a method of constructing a multi-channel circuit that allows parallel-processing of binary data in generating the **C**yclic **R**edundancy **C**heck (CRC). The multi-channel circuit is designed to be compatible with the conventional serial CRC register. This new circuit has the following advantages over the serial CRC register:

1. It allows parallel processing of f bits of data (a byte). This eliminates the serializing and buffering of data that is transmitted or received in the form of a byte.
2. The processing speed is f times faster, with relatively small increase in hardware.

In the following section, a CRC register is described. The serial and parallel (eight-channel) circuits are illustrated using a practical example. In the section "Multi-Channel CRC Register," we develop the mathematics for constructing a multi-channel CRC register. The result on a parallel linear feedback shift register as a $GF(2)$ polynomial divider has been described by Hsiao, et al.,[1,2] but a parallel CRC register has not beer constructed before in the form given in this paper.

CRC REGISTER

The **C**yclic **R**edundancy **C**heck (CRC) can be generated using a $GF(2)$ polynomial divider circuit.[3] In this circuit, to generate the check character, one shifts the binary message sequence, followed by r (the degree of the checking polynomial) zeros, into a polynomial divider circuit. The need for shifting r zeros can be eliminated[4] by changing the input connections of the conventional polynomial divider circuit. This modified circuit is conventionally known as the CRC register.

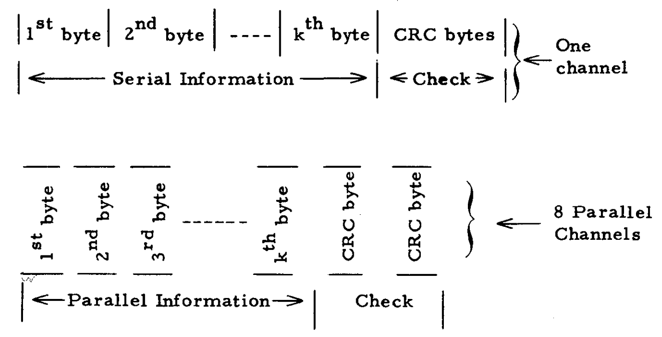


Figure 1—Message formats for serial and parallel systems

Figure 2 gives the circuit connections of a serial CRC register whose checking polynomial is $1+x^2+x^{15}+x^{16}$. This particular CRC register is used in the IBM 2701 system. For encoding, the binary message sequence is entered serially at the high-order end of the feedback shift register as shown in Figure 2 while the contents of the shift register is shifted toward the high order end. In this way, when the last bit of the message sequence is entered, the contents of the shift register represents the check character. In decoding, the received message bits are entered in the shift register in the same manner as in encoding. Likewise, the received check character is then shifted in. When the last bit of the check character is entered, the contents of the shift register represent the syndrome. A non-zero syndrome indicates an error in the received data.

Table I is a complete state transition table of the above process as a 16-bit binary message (namely,

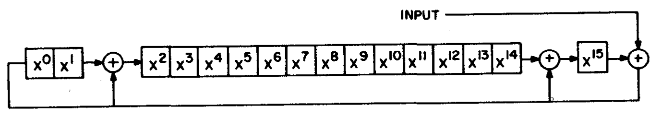


Figure 2—Serial CRC register

1101011110010011) is processed in the CRC register of Figure 2.

Figure 3 presents a multi-channel CRC register that processes eight bits (byte) in parallel. A single shift in this circuit with any eight-bit input sequence is equivalent to eight consecutive shifts in the serial circuit of Figure 2 with the same input sequence.

Table II is the state transition table of the multi-channel CRC register as the same binary message (in the form of two bytes) is processed through it. Note that the parallel circuit of Figure 3 produces the same CRC character eight times faster than the serial circuit in Figure 2.

MULTI-CHANNEL CRC REGISTER

In this section, we develop the mathematics for obtaining a multi-channel CRC register that can process f bits in parallel to generate the CRC character or the syndrome. One shift in the parallel circuit is equivalent to f shifts in the corresponding serial CRC register. The number f is a positive integer, smaller than the degree r of the checking polynomial.

$G(x)$ denotes the checking polynomial, often called

TABLE I—State Transition Table for Serial CRC Register

Time		Contents of Shift Register															
t	Input	x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^{10}	x^{11}	x^{12}	x^{13}	x^{14}	x^{15}
0	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
2	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
4	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1
5	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	1
6	1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0
7	1	1	0	0	1	1	1	1	0	1	0	0	0	0	0	0	1
8	1	0	1	0	0	1	1	1	1	0	1	0	0	0	0	0	0
9	1	1	0	0	0	0	1	1	1	1	0	1	0	0	0	0	1
10	0	1	1	1	0	0	0	1	1	1	1	0	1	0	0	0	1
11	0	1	1	0	1	0	0	0	1	1	1	1	0	1	0	0	1
12	1	0	1	1	0	1	0	0	0	1	1	1	1	0	1	0	0
13	0	0	0	1	1	0	1	0	0	0	1	1	1	1	0	1	0
14	0	0	0	0	1	1	0	1	0	0	0	1	1	1	1	0	1
15	1	0	0	0	0	1	1	0	1	0	0	0	1	1	1	1	0
16	1	1	0	1	0	0	1	1	0	1	0	0	0	1	1	1	0
CRC		1	0	1	0	0	1	1	0	1	0	0	0	1	1	1	0

TABLE II—State Transition Table for Parallel CRC Register

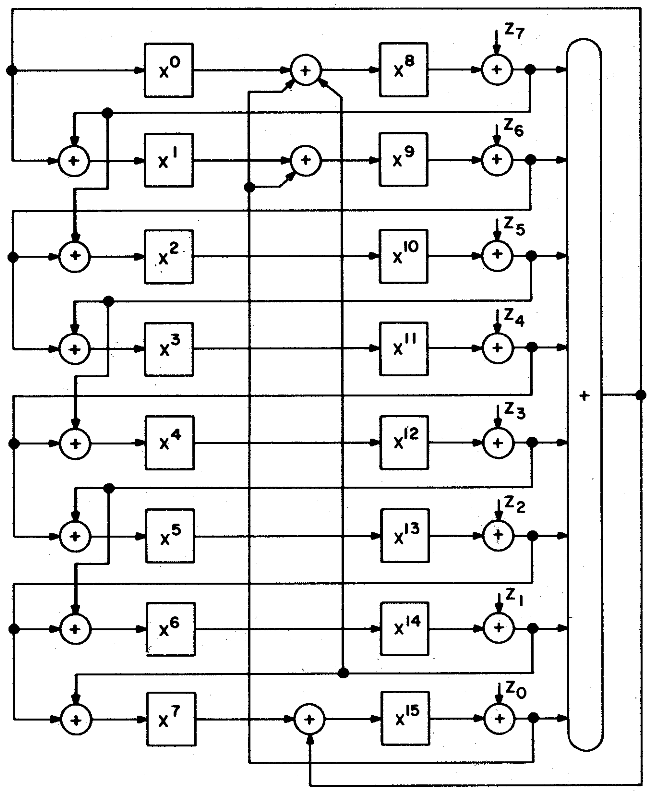
Time	Number of	Input								Contents of Shift Register															
t	the Shift	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^{10}	x^{11}	x^{12}	x^{13}	x^{14}	x^{15}
0	—	—	—	—	—	—	—	—	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	0	1	0	1	1	1	0	1	0	0	1	1	1	1	0	1	0	0	0	0	0	0
16	2	1	0	0	1	0	0	1	1	1	0	1	0	0	1	1	0	1	0	0	0	1	1	1	0
		CRC Character								1	0	1	0	0	1	1	0	1	0	0	0	1	1	1	0

the generator polynomial. We use the following notation:

$$G(x)=G_0+G_1x+G_2x^2+\cdots+G_rx^r \qquad (1)$$

The state vector $X_t=[x_0, x_1, \ldots x_{r-1}]_t$ denotes the contents of the CRC register at time t. T denotes the companion matrix of the polynomial $G(x)$, corresponding to the serial CRC register connections. Let z_t denote the data bit entering the serial CRC register at time t. Then the shifting operation of the serial CRC register is given by the (mod-2) matrix equation

$$X_{t+1}=X_tT\oplus z_tG \qquad (2)$$



INPUT BINARY SEQUENCE: $z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7$

CHECKING POLYNOMIAL : $1+x^2+x^{15}+x^{16}$

Figure 3—Eight-channel CRC register

where G is the vector $[G_0, G_1, G_2\cdots G_{r-1}]$, and T is given by:

$$T=\begin{bmatrix} 0 & 1 & & & \\ 0 & & 1 & & \\ & & & 1 & \\ 0 & & & & 1 \\ G_0 & G_1 & G_2 & \cdots & G_{r-1} \end{bmatrix} \qquad (3)$$

Suppose that $z_t, z_{t+1}, \ldots z_{t+f-1}$ are the f data bits (a byte) entering successively into the serial CRC register during the f consecutive shifting operations. The contents of the CRC register at the end of f shifts is denoted by the vector X_{t+f}. Using Equation 2 iteratively, f times, one can obtain:

$$X_{t+f}=X_tT^f\oplus z_tGT^{f-1}\oplus z_{t+1}GT^{f-2}\oplus\cdots z_{t+f-1}G \qquad (4)$$

Here T^j is the jth power of the matrix T. Let Z_t denote the input data sequence, as follows:

$$Z_t=[Z_{t+f-1}, z_{t+f-2}, \ldots, z^t{}_{+1}, z_t]$$

Let D denote the following partitioned matrix:

$$D=\begin{bmatrix} G \\ \hline GT \\ \hline GT^2 \\ \hline \\ \hline GT^{f-1} \end{bmatrix} \qquad (5)$$

Then, Equation 4 can be rewritten as:

$$X_{t+f}=X_tT^f\oplus Z_tD \qquad (6)$$

The sequential circuit realizing Equation 6 has the property that with the input byte Z_t (f bits in parallel), it changes from state X_t to X_{t+f} in a single shift. This is the equivalent operation to f shifts of the corre-

TABLE III—Rows of The Matrix D

Number of the Shift		Contents of the Serial CRC Register															
		x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^{10}	x^{11}	x^{12}	x^{13}	x^{14}	x^{15}
0	G	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
1	GT	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
2	GT$_2$	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1
3	GT^3	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1
4	GT^4	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1
5	GT^5	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	1
6	GT^6	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1
7	GT^7	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1

sponding serial CRC register with the same input data entered serially.

Note that the matrix T can be partitioned as:

$$T=\left[\frac{0 \mid I_{r-1}}{G}\right] \tag{7}$$

where I_m is the $m \times m$ identity matrix.

In general, it can be shown that T^f is equal to the following partitioned matrix:

$$T^f=\left[\frac{0 \mid I_{r-f}}{D}\right] \tag{8}$$

where D is given by Equation 5. One method of obtaining the matrix D is illustrated in the following Table III using the example given in the section "CRC Register."

Note that the vectors $G, GT, GT^2, \ldots GT^{f-1}$ represent the contents of the serial CRC register as the vector G is shifted $f-1$ times. Table III lists these vectors for the example in "CRC Register." The matrices D and T^f can be obtained using Table III and Equations 5 and 8. Now the implementation of Equation 6 leads to the parallel circuit.

The matrix T^f contains D as a sub-matrix. Hence, proper partitioning of the state vector will result in savings in hardware. We can partition the state vector X_t into two parts:

$$X_t=[X_t^1 \mid X_t^2] \tag{9}$$

where

$$X_t^1=[x_0, x_1, \ldots x_{r-f-1}] \tag{10}$$

$$X_t^2=[x_{r-f}, x_{r-f+1}, \ldots x_{r-1}] \tag{11}$$

Using Equations 8 and 9, we can rewrite equation 6 as follows:

$$X_{t+f}=X_t^1[0 \mid I_{r-f}]\oplus[X_t^2\oplus Z_t][D] \tag{12}$$

Implementation of Equation 12 directly gives the parallel circuit of Figure 2 in the example.

CONCLUSION AND COMMENTS

It is shown that corresponding to any polynomial $g(x)$ of degree r, one can generate a parallel CRC register that processes f bits in parallel $(f \leq r)$. The hardware is minimized by proper partitioning of the matrices in the state transition equation for the parallel circuit. For $f > r$, the theory of this paper can be applied without any change, except that the partitioning will then be applied to the D matrix rather than to the T^f matrix. This is obvious since D, in this case, contains T^f as one of its partitions.

ACKNOWLEDGMENT

The problem was originally suggested by Mr. W. D. Benedict and Dr. M. Y. Hsiao.

REFERENCES

1 M Y HSIAO K Y SIH
Serial-to-parallel transformation of feedback shift register circuits
IEEE Transactions on Electronic Computers Vol EC-13 pp 738-740 December 1964

2 M Y HSIAO
Theories and applications of parallel linear feedback shift register
IBM TR 1708 SDD Poughkeepsie March 1968

3 W W PETERSON
Error correcting codes
MIT Press 1961

4 W W PETERSON D T BROWN
Cyclic codes for error detection
Proceedings of the IRE pp 228-235 January 1961

Features of an advanced front-end CPU

by RICHARD BARR HIBBS

The Bunker-Ramo Corporation
New York, New York

INTRODUCTION

A central processing unit to handle data communications chores as a front-end computer has historically been either a maxi-computer, overpowered for the intended job, or a mini-computer, stripped of many instructions and architectural features that now ease the programming burdens of commercial data processing. Front-end CPU's are evolving into general-purpose machines in their own right due to demands for more generalized processing by the front-end, such as code conversion, message text pre-editing, and local (i.e., not performed by the host computer) message switching. Front-end CPU's must be dual-purpose machines—a special-purpose input-output structure to handle communications efficiently, and a general-purpose data handling structure to perform tasks such as described above. Certain desirable features of a front-end CPU are described informally in this paper, then the architecture of a proposed front-end CPU which incorporates these features is presented.

DESIRABLE FEATURES OF A COMMUNICATIONS PROCESSOR

The more densely a program can be coded, the more reliable it may be considered to be. That is, if the set of machine operation codes include null codes, privileged codes, context-sensitive codes, or codes which bypass normal machine operation, then the set of codes is inherently less reliable than a set which does not include such codes.

The use of re-entrant coding techniques has a beneficial side effect, the elimination of program code which modifies other code or can itself be modified, and reduces the frequency of instructions which can in themselves, cycle indefinitely. Program modification is probably the source of many "phantom clobberers" found in any large software system.

In an environment where multiprogramming is the exception rather than the rule, the added hardware complexity required to implement an indexed base-displacement addressing scheme (a la 360) is questionable, but for the frequent use made of virtual tables. Manipulation of data and control information maintained in tabular form is required to implement re-entrant coding techniques. To efficiently access table structures, a variety of addressing schemes must be available to the programmer.

Queueing of data and control information maintained as elements in a linked list is the basic operation of communications tasks. Low overhead queueing operations on several common types of queues will eliminate an often unwieldy set of system subroutines. The addition of coroutines and subcoroutines to the types of program elements handled by the program control structure would add two very useful facilities to any computer coded as independent modules, and would be particularly valuable in a front-end. Extending the control instruction repertoire beyond the familiar "BRANCH ON CONDITION, "and "INCREMENT (or DECREMENT) AND TEST FOR ZERO" is needed to effectively make use of the new types of program elements.

Rather than settling the question of word versus byte-oriented computer organization, note that by allowing partial-word operation on all data-manipulating instructions, a considerable amount of masking and shifting can be eliminated from operating programs, although only limited flexibility is available without creating difficult instruction coding problems.

As the internal circuitry of a CPU is far more reliable in operation than the attached communications lines, the extra memory cycle time and additional hardware necessary to provide memory parity or error detection and correction is unacceptable. Advances in technology coupled with the use of error-correcting codes may change this point of view, however, in the near future.

The real power of a CPU is not measured by the

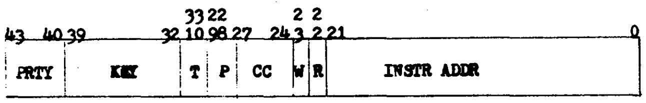


Figure 1—Processor state register (PSR)

instruction cycle time, but rather by the amount of processing performed by each instruction and the number of instructions necessary to perform a given function.

Taking advantage of storage technology by buffering main storage with high-speed local storage and placing operating programs in read-only storage suggests a return to Harvard-class computers with separate addressing spaces for data and instructions. Immediately, the possibility of executing data or operating on instructions is completely eliminated, thus improving the reliability of the software. Storage protection is unimportant for the program store of a Harvard-class machine, but is still important in protecting constant areas in data store from accidental destruction. Both read and write protection of data store are useful, almost mandatory, and should be provided for.

Conventional input-output channels of large-scale computers are constructed to provide an efficient, general scheme for input-output. By specializing the input-output channels to handle communications only, and by integrating channel controls within the front-end, the power of the front-end computer is extended and directed at communications. Assuming only communications lines and interprocessor channels are attached, the lack of general facilities for input-output is not a limitation of the processor.

GENERAL CP ARCHITECTURE

The communications processor (CP) is a multi-accumulator, two's complement, fixed-point binary, stored program digital computer with separate addressing spaces for programs and data. CP control circuitry interfaces with operating programs through the processor state register (PSR), a control register which is the central element of the interrupt mechanism. All input-output channels and their controls are functionally integrated within the CP itself to expedite input-output operations by treating each device interface as an addressable extension of CP data memory.

CP program memory consists of up to 65,536 words of 44 bits each, with the first 2,560 words reserved for interrupt processes. The operand address of all program transfer of control, queue manipulation, and input-output instructions refer to locations in program memory. Program memory is addressed consecutively from 0 to the highest available address—an invalid address generates an addressing error interrupt. The format and interpretation of words contained in program memory is described in the "Instruction Set" Section.

CP data memory, separately addressed from CP program memory, contains up to 4,194,204 thirty-six bit words. CP data memory is under control of a protection lock assigned to each 2048 word module of core and a protection key contained in the current PSR. Storage addresses of data memory run from 0 to the highest available address with invalid addresses generating an addressing error interrupt. Locations 0 to 2048 are reserved as control words for input-output channels.

The PSR consists of PRTY, KEY, T, P, CC, W, R and INSTR ADDR fields, as shown in Figure 1. Only the PRTY field may be modified by an operating program without loading an entire new PSR. The PRTY field specifies the "level" at which the current program is operating—0 indicates non-interruptible, critical processes and 15 indicates non-critical, completely interruptible processes. With 16 priority levels at which the CP can operate, the dispatching urgency of interrupts can be dynamically altered. Every request for interrupt is at a priority determined sometime before the request is generated. All "program" interrupts have a fixed priority of 1, 2, or 3. All input-output interrupts have a priority specified by the START I/O instruction which initiated the operation. When a request for interrupt is presented whose priority is equal to or greater (less numerically) than that specified by the PRTY field, the interrupt request is granted, otherwise the request is stacked by CP control for later servicing.

The KEY and T fields control access to CP data storage. Data storage protection is always in effect. When an instruction requires access to CP data storage, the KEY is matched against the lock associated with the memory module containing the desired address, with access granted according to the match-up.

The CC field controls conditional transfer of control instruction execution. The meanings of each bit are defined according to the preceding instruction executed. The CC field is reset following execution of all but conditional transfer of control instructions. All conditional transfer of control instructions interrogate the CC field according to the mask given by the R1 field of the instruction in order to determine whether or not a branch will be taken. Matching one bits in any bit position causes the branch to be taken.

The INST ADDR field contains the address of the next instruction to be executed. It is updated sequentially after execution of each instruction until a transfer of control instruction takes a branch, when the branch address becomes the next instruction address.

The W and R bits define the operating states of the CP. If W is set, the CP is in the wait state and no instructions are executed. Input-out does not stop, however, when in the wait state. The R bit specifies which set of general registers is used by each instruction.

Two interchangeable sets of general purpose registers exist within the CP. Each register is 36 bits wide and is designated by a number from 0 to 15. Fifteen of the 16 registers may be used as accumulators, index registers, or base registers. Resister zero may be used only as an accumulator.

The P field describes the current type of program element being executed. A main program or subroutine is indicated by 00 or 01, the distinction between them being almost impossible to determine as a return branch from a subroutine takes exactly the same form as an indirect branch within any program element. A coroutine is indicated by 10 and a subcoroutine is indicated by 11. If a program transfer of control instruction is executed which is invalid for the current type of element, a program linkage interrupt is generated.

When an interrupt request is granted, an address presented along with the request is used to specify the address in program memory from which to fetch a new PSR. The old PSR, having been saved in a pushdown stack, may be made the current PSR in order to re-enter the interrupted routine at the end of the interrupt service by executing an UNCHAIN instruction.

As each interrupt is identified with a distinctive new PSR, no interrogation of devices, i.e., polling of interrupts, is required during interrupt service.

INSTRUCTION SET

Each CP instruction occupies one 44-bit word of CP program memory in one of the formats shown in Figure 2. Instructions operate between registers, between storage and registers, or between storage locations. In the register to register and storage to register formats, three addresses are specified—two for operands and one for the result. The result and one operand are specified by the contents of the indicated register. The other operand address refers either to CP memory (storage to register format) or a general register (register to register format). In the storage to storage instruction format, a source and destination address and a length code are specified. All references to CP memory refer to CP data memory unless the instruction is a control instruction (e.g., BRANCH ON CONDITION, LOAD PSR, or PUSH).

Memory addresses of storage to register format

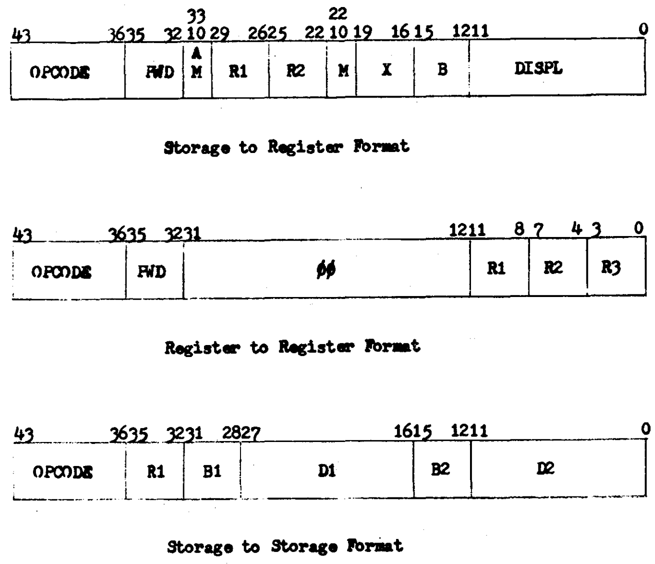


Figure 2—Communications processor (CP) instruction formats

instructions are formed and interpreted according to the addressing mode (AM) field of the CP instruction. If AM is 00, the modifier (M), index (X), base (B), and displacement (DISPL) are taken as an absolute 16 or 22-bit address and the addressing mode is called DIRECT. If either program or data memory contains fewer words than addressable by the full 16 or 22-bit value, any reference to an address lying outside the addressing space will generate an addressing interrupt.

If AM is 01, the addressing mode is called INDEXED and the address is formed from two sums. The DISPL field is added to the contents of the register specified by the B field, unless the B field is zero. If the B field is zero, the value zero is used in forming the first sum. To the first sum, the contents of the register specified by the X field are added, unless the X field is zero. If the X field is zero, the value zero is used in forming the second sum. The second sum is used as the data or program memory address. The register specified by the B field is considered to contain a signed, 35-bit integer, even though the resulting sum will be truncated to a 16 or 22-bit address. The register specified by the X field is considered to contain a signed, 17-bit integer. Addresses are formed in a 36-bit register in the CP control section, then truncated to the appropriate precision.

The high-order 18 bits of the register specified by the X field are taken as a signed, 17-bit modifier of the actual index, contained in the low-order 18 bits of the register, according to the M field of the instruction. If M is 00, the index is not modified. If M is 01, the modifier is added to the index and the sum becomes the new index. If M is 10, the modifier is subtracted from the index and the difference becomes the new index. If M is

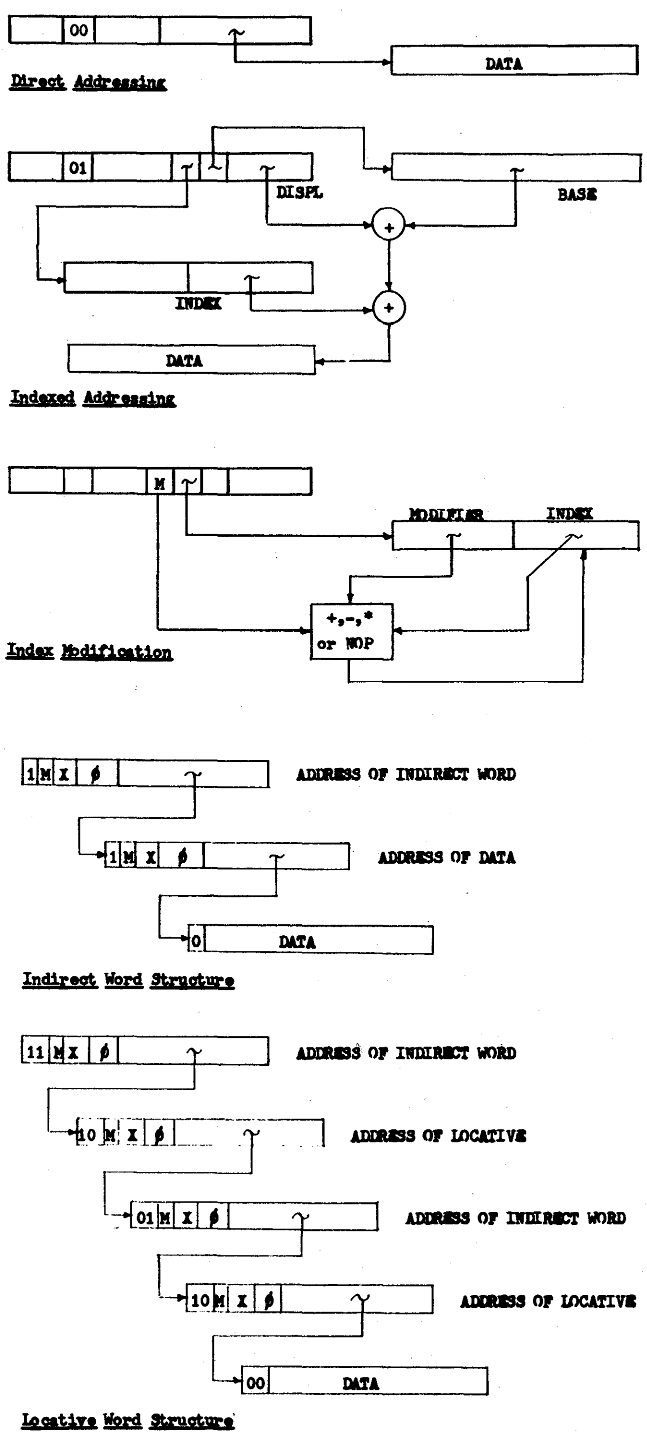


Figure 3—Communications processor (CP) addressing structure

11, the index is multiplied by the modifier and the product becomes the new index. All index modification is performed after current instruction execution, before the next instruction is executed.

If AM is 10, the addressing mode is called INDIRECT and the address is formed as described for the INDEXED mode but is not the address of data or a new program address, but the address of an indirect word. The indirect (I) bit of the indirect word specifies whether the address pointed to by the indirect word is the address of another indirect word or the address of data. If I is 0, the next word is data (or next instruction address). If I is 1, the next word is another indirect word. The M and X fields are interpreted for the indirect word just as they are interpreted for an instruction, with the contents of the register specified by the X field added to the ADDRESS portion of the indirect word. Multi-level indirection and indexing are thus possible.

If AM is 11, the addressing mode is LOCATIVE, a combination of INDEXED and INDIRECT modes. The address is formed as described for the INDEXED mode, but is the address of a locative rather than that of data or a new program address. A locative is distinguishable as data, the address of data (beginning of an indirect chain), the address of another locative, or the address of the address of another locative (beginning of an indirect chain ending with a locative). For a transfer of program control instruction, LOCATIVE mode has the same interpretation as INDIRECT mode. The L field determines the interpretation of the locative. If L is 10, the locative contains the address of another locative. If L is 11, the locative begins an indirect chain terminated by another locative. The M and X fields are interpreted just as for the indirect word.

If the resulting address addresses CP data memory, the protection KEY of the current PSR is compared to the storage lock for that segment of data memory. If they match, access is granted according to the tag bits which match between the PSR and the storage lock. One tag bit allows read access, and the other tag bit allows write access. When the key does not match the lock a protection interrupt is generated. The general CP addressing structure is illustrated by Figure 3.

Storage-to-storage format instructions have fewer, but similar, fields than do storage-to-register format instructions. These are two address instructions, with the source address given by the sum of the D2 field and the contents of the register specified by B2 (unless zero) and the destination address given by the sum of the D1 field and the contents of the 9-bit characters of the fields involved in an operation are contained in the register specified by R1.

Program elements can be one of four types: main program or open subroutine, closed subroutine, co-routine, or subcoroutine. Program control is passed to the main program or passed within any of the elements by means of a BRANCH ON CONDITION or BRANCH ON INDEX instruction. The BRANCH ON CONDITION instruction substitutes a four bit mask for the R1 field which is compared to the four bit CC field of the current PSR. A match in any bit position causes the branch to be taken to the address in program memory given by the B, X, and DISPL fields in the mode specified by the AM field. Note that the R2 and PWD fields do not enter into the instruction execution.

The BRANCH ON INDEX instruction affects the register specified by the X field. The X and M fields are not used in determining the branch address. The R1 field specifies a register containing the value to be compared to the index portion of the register specified by the X field. Only the low-order 18 bits of the R1 register are used in the comparison. The R2 field specifies a register containing a signed increment in bits 0–18 which modifies the index register according to the M field when the branch is not taken. The M field is interpreted as before. The branch is taken whenever the specified test condition is met. The test conditions are: index high, index equal, and index low. Transfer of program control to a subroutine is made with a BRANCH AND LINK instruction. The register specified by the R1 field is taken as a four bit mask for comparison against the CC field of the current PSR, just as for the BRANCH ON CONDITION instruction. If any bits in the mask match the CC, the register specified by the R2 field is first loaded with the address of the next sequential instruction, then the branch is taken just as for the BRANCH ON CONDITION instruction.

A coroutine is a program element defined by its entry locator, which specifies the address of the first instruction to be executed upon entry to the coroutine. An INITIALIZE or LEAVE group instruction defines the contents of the entry locator, and an ENTER group instruction performs the co-transfer into the coroutine. INITIALIZE is a storage to storage format instruction that sets the entry locator for the named coroutine to the given address in CP program memory. LEAVE and ENTER group instructions are also storage to storage instructions which set the entry locator for the current routine to the address of the next sequential instruction then transfer indirect through the entry locator of the target routine (if a coroutine or subcoroutine) or branch to the target address (if a main program or subroutine).

A subcoroutine, like a coroutine, is defined by its entry locator, and also by an exit locator. The entry locator may be defined by an INITIALIZE or a RETURN group instruction. Entry to a subcoroutine is made by executing a CALL group instruction, which sets the entry locator of the current module (if a coroutine or subcoroutine) to the next sequential instruction address and transfers control indirectly into the target subcoroutine through its entry locator, then sets the exit locator to the address of the next sequential

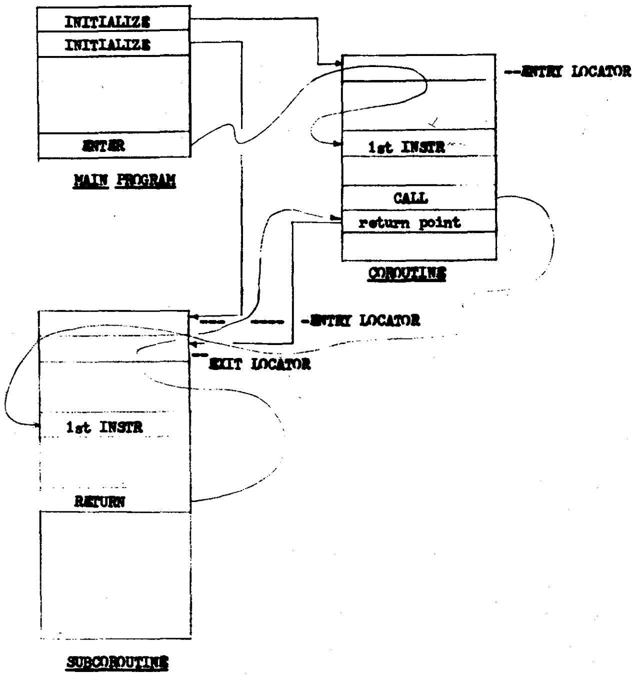


Figure 4—Coroutine and subcoroutine structure

instruction following the CALL in the calling routine. When a RETURN instruction is executed, the entry locator of the current module is set to the next sequential instruction address and a branch is taken to the return point, indirectly through the exit locator. The entry and exit locators occupy contiguous locations of CP program memory. Figure 4 shows coroutine and subcoroutine structure.

To provide low-overhead queueing operations, three types of queues are maintained by hardware: pushdown stacks, normal FIFO or head-tail queues, and double-ended head-tail queues. A pushdown stack is defined as a contiguous area of data memory with a locator word in program memory. The locator word consists of length, count, and pointer fields. The count is decremented by one for each element pushed down into the stack. The pointer initially contains the address of the first available element, and is incremented by the length field for each element added. Likewise, for each element removed, the pointer is decremented by the length field. If the count is zero then an attempt is made to add an item, and overflow condition exists and a stack overflow interrupt is generated.

Normal and double-ended head-tail queues are defined as unbounded linked lists. Any head-tail queue may be either normal or double-ended, defined by the needs of the moment by the instruction executed to manipulate an element. All instructions affect the queue pointer words and the link word of the cell being manipulated.

Notation

$+$	Addition	$\wedge$	Logical AND
$-$	Subtraction	$\vee$	Logical OR
$\cdot$	Multiplication	$\odot$	Coincidence
$/$	Division	$(\ )$	Contents of
$:$	Comparison	M	Effective Address
$\leftarrow$	Replaces	$\leftrightarrows$	Interchange

Instructions

ADD	$(R1) \leftarrow (M) + (R2)$
SUBTRACT	$(R1) \leftarrow (R2) - (M)$
MULTIPLY	$(R1, R1+1) \leftarrow (M) \cdot (R2)$
DIVIDE	$(R1) \leftarrow (R2, R2+1)/(M)$; $(R1+1) \leftarrow$ REMAINDER
AND	$(R1) \leftarrow (M) \wedge (R2)$
OR	$(R1) \leftarrow (M) \vee (R2)$
COINCIDENCE	$(R1) \leftarrow (M) \oplus (R2)$
LOAD MULTIPLE	$(R1, \ldots, R2) \leftarrow (M, \ldots, M_{[R2-R1+1]})$
STORE MULTIPLE	$(M, \ldots, M_{[R2-R1+1]}) \leftarrow (R1, \ldots, R2)$
INTERCHANGE	$(M) \leftrightarrows (R1)$
LOAD ADDRESS	$(R1) \leftarrow M$
COMPARE MASKED	$[(M) \vee (R2)] : (R1)$
LOAD PSR	$(PSR) \leftarrow (M)$

Figure 5—Representative CP instructions

To put a cell on the tail of the queue, a PUT instruction is executed. A GET instruction removes a cell from the head of the queue. FETCH removes from the tail, and STORE adds to the head. Complete freedom in intermixing these instructions would allow, for example, a processing routine to break off processing a message cell for a low-priority message by placing its pointer back on the head of the processing queue in order to respond to a request for processing of a high-priority message cell.

Figure 5 describes the operation of representative instructions of the CP instruction set in shorthand notation.

INPUT-OUTPUT

Input-output is the reason for being of a front-end computer. A high-powered processor such as has been described here must be capable of sustaining high data transfer rates in comparison with processing speed or it will be severely mismatched to its task. In the CP input-output proceeds simultaneously with processing by allowing multiple access to data memory according to priority of input-output (determined by a START I/O instruction). The processor always has lower priority for data memory access than does input-output. By providing one memory port for each 2048 word module of data memory, the overall memory bandwidth is increased; unless, of course, the operating programs attempt to place all buffers in the same (few) modules.

All control information for each device attached to the CP is contained in a line control word (LCW) maintained on a per-device basis in the first 2048 word module of CP data memory. When a START I/O instruction is executed, the control words are activated and a buffer address in data memory is provided, along with the appropriate protection key. Input-output then proceeds until an error occurs, the last data character is transferred into memory, or a HALT I/O instruction is executed for the device. Control information may be modified successfully whenever the control words are not active, thus allowing the operating programs to dynamically reconfigure the input-output to meet shifting demands.

Each communications interface consists of receive and transmit circuitry sufficiently general to allow selection of functional characteristics by signals on the I/O control bus. In order to dynamically reconfigure the attached communications network, each unit must be capable of handling several code structures and transmission speeds.

For a small number of combinations of code and speed, the communications interface for asynchronous transmissions is not overly complex. The advantage of being able to dynamically change operating characteristics of a line is apparent for a time-sharing service, which could use only as many line terminations as the number of simultaneous users to serve several types of terminals, rather than apportioning facilities according to projected loads from the different types of terminals which usually results in several unused lines when the system is heavily loaded.

Synchronous transmissions, if they involve any kind of line discipline for transmission, are code-dependent; so the only kind of dynamic reconfiguration would be to change clock speeds for the line. This trick is now being used by at least one terminal manufacturer to overcome a noisy line—if too many errors occur at, say, 4800 baud, the unit switches to a 2400 baud clock to improve transmission and reception.

Characters are assembled/disassembled at individual line termination units (LTU), buffered, then stored a word at a time in CP data memory. Memory words are left-justified with zero pad. Common controls direct the LTU to select a particular code structure/transmission speed, collect assembled words according to dispatching priority (specified by a START I/O instruction), and direct transfer between CP data memory and the LTU's.

A complete interface is provided for a modem by each LTU, and provision is made for the addition of an Auto-Call Unit. All status-bearing and control leads present at the modem interface are represented by bits in the LCW, allowing an operating program having the proper protection key to actually control the LTU at the interface level. Additionally, for asynchronous lines, the state-of-the-stop-bit is reported/controllable, allowing detection and generation of open line conditions. See Figure 6 for the layout of the LCW.

Syncronous LTU's would be constructed according to the requirements of the user, in order to provide hardware for handling of line discipline. Redundancy checking and message retransmission are handled more easily by hardware than software.

As data can be manipulated easily in the CP as nine-bit bytes, character-by-character line service is not infeasible for applications requiring intimate inspection of message traffic. On the other hand, a complete message could be assembled in data memory before the LTU acknowledges reception. The ability to select between these methods of line service within the same unit indicates some of the power of the CP.

First-generation computers had all input-output control integrated within the CPU for simplicity of construction. Later generations used special-purpose controllers to handle all input-output devices. Now, the newest round of computer announcements shows a return to integrated peripheral controllers as a cost saving and to upgrade performance of essential devices. A dedicated communications controller would utilize integrated device control for both reasons. Similar communications input-output controllers have been built by at least one firm on an experimental basis, but initial efforts provided a costly design. It is not out of the question to expand such a controller to the capacity indicated here, nor to integrate it with the logic of a central processing unit.

CONCLUSION

A rather high-powered communications subsystem has been described in varying degrees of detail. Certain combinations of architectural features are unique in any digital computer, especially so in a front-end which is usually thought of as a small computer system. Many advanced programmers probably make use of data structures closely resembling locatives without realizing that a name exists for such a structure, just as many

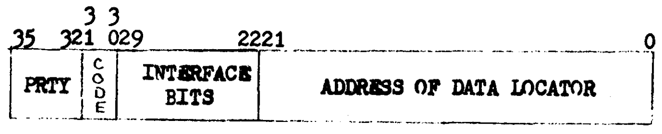


Figure 6—Line control word (LCW) layout

utilize subcoroutines without knowing it. The revival of some first-generation architectural features coupled with the combination of more modern ones yields a machine whose applications could be greatly expended with only small conceptual changes. Still, the primary purpose in attempting the design of a novel central processor was to look for the new ways of handling familiar problems; and the design has fulfilled the author's intentions, raised many questions to be answered, and provided new techniques for discussion. The evolution of computer systems seems to indicate a collection of peripheral processors for a new configuration—the CP being such a proposed component.

REFERENCES

1 L L CONSTANTINE
Integral hardware-software design
Parts 8 and 9 Modern Data November 1968 and February 1969

2 R W COOK M J FLYNN
System design of a dynamic microprocessor
IEEE Transactions on Computers March 1970

3 C J WALTER A B WALTER M J BOHL
Impact of fourth generation software on hardware design
Computer Group News March 1969

4 E C JOSEPH
Evolving digital computer system architecture
Computer Group News March 1969

5 H W LAWSON JR
Programming-language-oriented instruction stream
IEEE Transactions on Computers May 1969

6 G D HORNBUCKLE E A ANCONA
The LX-1 microprocessor and its application to real-time signal processing
IEEE Transactions on Computers August 1970

7 E A HAUCK B A DENT
Burroughs B6500/B7500 stack mechanism
Proceedings Spring Joint Computer Conference 1968

8 J G CLEARY
Process handling on Burroughs B6500
Proceedings Fourth Australian Computer Conference 1969

Interpreting the results of a hardware systems monitor

by J. S. COCKRUM and E. D. CROCKETT

Memorex Corporation
Santa Clara, California

INTRODUCTION

Hardware monitors are widely used to enable the data processing manager to effect cost reductions and improve the efficiency of his installation. Several papers[1-7] have presented hardware monitors and system measurement, but have presented relatively little information regarding the interpretation of the monitoring results.

A brief overview of hardware monitors and the necessity of system measurement is presented. A section deals with determining and measuring events—significant occurrences to a unit of work being processed by the system. A "performance optimization cycle" is developed and actual results of a monitoring run are shown.

The body of this report treats the heretofore neglected area of interpretation of the results. The stress is on providing quantitative measures to assure that an economic return on the computer system is obtained.

The system performance profile is presented and the basic indicators in interpreting the profile are developed. Methods are given for corrective actions of system reconfiguration, program change, data set reorganization, job scheduling and operator procedures. Predictive methods are developed whereby reconfigurations can be evaluated prior to their implementations.

DESCRIPTION OF MONITOR

A hardware monitor consists of sensors, control logic, accumulators, and a recording unit. The sensors are attached on the back panels of the computer system components—CPU, channels, disks, etc. The signals from registers, indicators and activity lines picked up by the sensors can be logically combined or entire registers can be compared at the control panel and then be routed to the accumulators for counting or timing events, e.g., CPU active, any channel busy, disk seek counts and times, etc. Typically, the contents of the accumulators are written periodically to a magnetic tape unit. The magnetic tape is batch processed by a data analysis program to produce a series of analysis, summary and graphic reports. Figure 1 shows a hardware monitoring system.

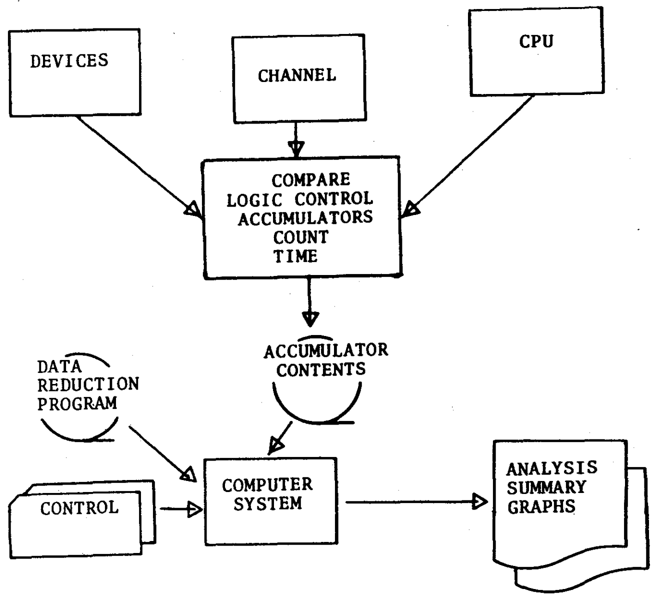


Figure 1—Hardware monitor system

NECESSITY OF MONITORING

The complexity of the present computing systems has made monitoring a necessity for effective management. Effective management means optimizing the computer system performance for increased throughput, turn-around time, or a reduction in expenses; and predicting future computer system needs. A hardware monitor provides a tool to efficiently obtain these management goals. It is easy to install and use, and measures simultaneous occurring events of hardware and software operations without any interference to the system being monitored.

Computer system performance optimization

A computer system can be optimized according to different strategies. The most common strategy is to optimize the system's *throughput*, i.e., the rate at which the workload can be handled by the system. Other strategies include optimizing *turnaround time* (delay between the presentation of the input to the system and the receipt of the output), *availability* (percentage of time during which the system is operating properly), *job time* (length of time the system takes to perform a single application), *cost* (the costs of the computer system used in processing the workload), etc. Often a combined optimation strategy will be followed, e.g., maximize throughput for a given cost. Although a hardware monitor can be used for any optimization strategy, the concern in this paper will be for throughput and cost, since it is felt that these are the principal measures of economic return on a system. To this end, consideration will be given to system configuration including reconfiguration and additions/deletions of components, programs, rotuines to be made resident/non-resident, data set allocation, job scheduling and operating procedures.

The performance optimization cycle which will be developed consists of computer system measurement, evaluation, improvement and returning to new measurements to start a new cycle.

Prediction of future needs

From the records obtained in the performance optimization cycle, not only is the current performance known, but also a historical base is constructed for predicting future needs. Based upon actual system measurements it is possible to predict and evaluate major changes in the capability of the system before their implementation. System changes of reconfiguration, addition/deletion of new devices and model changes can be simulated and analyzed. Accurate prediction of future needs dictates a continuing monitoring program.

MONITORING EVENTS

An *event* in a computer system is an occurrence of significance to a unit of work processed by the system. A hardware monitor can count or time the duration of events or combinations of events.

It is necessary to identify the events upon which the system performance depends and quantitatively determine their interdependency. The basic events monitored are components active, time spent performing various operations, storage utilized, resource contention, system resource overlapping, etc.

Single sensor events

The types of events which can be monitored in a computer system using a single sensor for each are such occurrences as:

CENTRAL PROCESSOR UNIT

1. CPU STOP or MANUAL
2. CPU WAIT
3. CPU RUN
4. MULTIPLEX CHANNEL BUSY
5. SELECTOR CHANNEL BUSY
6. PROGRAM CHECK INTERRUPT
7. I/O INTERRUPT
8. ALLOW INTERRUPT CHANNEL
9. PROBLEM STATE
10. SUPERVISOR STATE
11. INSTRUCTION FETCH
12. EXTERNAL INTERRUPT
13. CONSOLE BUSY
14. STATUS OF THE SYSTEM IN RELATION TO A PARTICULAR PROGRAM (IBM PSW PROTECTION KEYS)

DIRECT ACCESS STORAGE DEVICE

1. CONTROL UNIT BUSY
2. NUMBER OF SEEKS
3. INTERRUPT PENDING
4. READ BUSY
5. WRITE BUSY
6. DATA BUSY BY MODULE
7. TOTAL SEEK TIME BY MODULE

CONTROL UNITS

1. DEVICE BUSY
2. REWINDING TAPES
3. DATA TRANSFER
4. POLL of TERMINALS

UNIT RECORD EQUIPMENT

1. LINES PRINTED
2. CARDS READ
3. DEVICE BUSY
4. CARDS PUNCHED

Multiple sensor events and comparators

Events which require multiple sensors and comparators are:

1. INSTRUCTION ADDRESSES
2. REGISTER CONTENTS
3. INTERFACE DATA
4. PARTITION BOUNDARIES
5. DATA SET BOUNDARIES

Combination events

Any number of combination events can be constructed using the monitor and the data reduction program.

Examples of combination events are:

1. CPU ACTIVE
   $(\text{CPU RUN} \wedge \overline{\text{CPU WAIT} \wedge \text{CPU MANUAL}})$
2. ANY CHANNEL BUSY
   $(\text{CHANNEL 1 BUSY} \vee \text{CHANNEL 2 BUSY} \vee \ldots \vee \text{CHANNEL N BUSY})$
3. ANY CHANNEL BUSY ONLY
   $(\text{ANY CHANNEL BUSY} \wedge \text{CPU WAIT})$
4. TOTAL SYSTEM TIME
   (CPU ACTIVE+SYSTEM WAIT)
5. CPU-CHANNEL OVERLAP
   $(\text{CPU ACTIVE} \wedge \text{ANY CHANNEL BUSY})$
6. CHANNEL OVERLAP
   $(\text{CHANNEL 1 BUSY} \wedge \text{CHANNEL 2 BUSY} \wedge \ldots \wedge \text{CHANNEL N BUSY})$
7. SEEK ONLY
   $(\text{CPU WAIT} \wedge \overline{\text{ANY CHANNEL BUSY}} \wedge \text{SUM OF SEEKS IN PROGRESS ON ALL MODULES})$

```
MEASURED 10/ 6/70     SYSTEM/360 MODEL 40 AND 50 ANALYSIS
------------------------------------------------------------------------------------------------

                              N U M E R I C   P R O L O G U E
------------------------------------------------------------------------------------------------

SYSTEM UTILIZATION MONITOR RECORDINGS WERE MADE ON 10/ 6/70 STARTING AT  8.30. 0.0

ORIGINAL  S U M  RECORDING INTERVAL =      1.0 SECONDS.

S U M  TAPE PHASE SUMMARIZED   30 S U M TAPE RECORD(S) INTO EACH WORK FILE RECORD.

S U M  COUNTERS WILL BE SUMMARIZED EVERY      10 WORK FILE RECORDS, BEGINNING AT      0.0 AND ENDING AT  999999.9 SECONDS.

 COUNTER        ID          TIME/EVENT      DESCRIPTION                                         BASE ID

   0             0              T           TOTAL ELAPSED TIME                                    G
   1             1              T           CPU ACTIVE MODEL 50                                   G
   2             2              T           PROBLEM STATE MODEL 50                                G
   3             3              T           ELAPSED TIME METER RUNNING MODEL 40                   G
   4             4              T           SELECTOR CHANNEL 1 BUSY MODEL 50                      G
   5             5              T           SELECTOR CHANNEL 2 BUSY MODEL 50                      G
   6             6              T           ANY CHANNEL BUSY MODEL 50                             G
   7             7              T           ANY CHANNEL BUSY AND CPU WAIT MODEL 50                G
   8             8              T           CHANNEL 1 AND 2 OVERLAP MODEL 50                      G
   9             9              T           2314 BUSY TO MODEL 40                                 G
  10             A              T           2314 BUSY TO MODEL 50                                 G
  11             B              T           MODEL 40 HAS 2314,MODEL 50 WANTS                      G
  12             C              T           MODEL 50 HAS 2314,MODEL 40 WANTS                      G
  13             D              T           MODEL 40 USING 2314,MODEL 50 WAIT ONLY                G
  14             E              T           MODEL 50 IN MANUAL STATE                              G
  15             F              T           PROBLEM STATE AND MODEL 50 CPU ACTIVE                 G
------------------------------------------------------------------------------------------------
  16             G                          ************ SOFTWARE CLOCK ************              G
------------------------------------------------------------------------------------------------
  17             H                          CONTENTION RATIO BETWEN MODEL 40 AND 50               $
  18             I                          CPU WAIT MODEL 50                                     G
  19             J                          CPU WAIT ONLY MODEL 50                                G
  20             K                          CPU ACTIVE ONLY MODEL 50                              G
  21             L                          NEW ANY CHANNEL BUSY MODEL 50                         M
  22             M                          NEW SYSTEM TIME MODEL 50                              $
  23             N                          NEW ANY CHANNEL BUSY AND WAIT MODEL 50                M
  24             O                          NEW WAIT MODEL 50                                     M
  25             P                          INCREASE IN MODEL 50 SYSTEM TIME                      G


INTERMEDIATE SUMMARY WILL BE PRINTED EVERY        5 INTERVAL SUMMARIES.
```

Figure 2—Description of events monitored and combined

```
MEASURED 1C/ 6/70       SYSTEM/36C MODEL 4C ANC 50 ANALYSIS

FILE SUMMARY OVER PREVIOUS      3G0.0 SECCNDS, BEGINNING AT   9. 5. 0.0 AND ENDING AT   9.1C. C.C

COUNTER    ID      DESCRIPTION     BOARD IC C    JOB ID 24                    COUNTER VALUE          PERCENT                 BASE

   0        0      TOTAL ELAPSED TIME                                          296.950980          100.00 PCT CF CCUNTER 16
   1        1      CPU ACTIVE MODEL 50                                          74.228995           25.C0 PCT OF COUNTER 16
   2        2      PROBLEM STATE MODEL 50                                      170.242988           57.33 PCT OF COUNTER 16
   3        3      ELAPSED TIME METER RUNNING MCDEL 40                         296.686980           99.98 PCT OF COUNTER 16
   4        4      SELECTOR CHANNEL 1 BUSY MCDEL 50                            119.388992           40.21 PCT OF COUNTER 16
   5        5      SELECTOR CHANNEL 2 BUSY MCDEL 50                              5.190000            1.75 PCT OF COUNTER 16
   6        6      ANY CHANNEL BUSY MODEL 5C                                   123.248992           41.51 PCT OF COUNTER 16
   7        7      ANY CHANNEL BUSY AND CPU WAIT MODEL 50                      101.813993           34.29 PCT OF COUNTER 16
   8        8      CHANNEL 1 AND 2 OVERLAP MCDEL 50                              1.326000            0.45 PCT OF COUNTER 16
   9        9      2314 BUSY TC MODEL 40                                        21.121999            7.11 PCT OF COUNTER 16
  10        A      2314 BUSY TO MODEL 50                                       119.437992           40.22 PCT OF CCUNTER 16
  11        B      MODEL 40 HAS 2314,MODEL 50 WANTS                              5.923000            1.99 PCT OF COUNTER 16
  12        C      MODEL 50 HAS 2314,MODEL 4C WANTS                             12.422999            4.18 PCT OF COUNTER 16
  13        D      MODEL 40 USING 2314,MODEL 50 WAIT ONLY                        0.0                 0.0  PCT OF COUNTER 16
  14        E      MODEL 5C IN MANUAL STATE                                      0.0                 0.0  PCT OF COUNTER 16
  15        F      PROBLEM STATE AND MODEL 50 CPU ACTIVE                         9.384999            3.16 PCT CF COUNTER 16
  16        G      ************ SOFTWARE CLOCK ************                    296.939850          100.00 PCT OF COUNTER 16
  17        H      CONTENTION RATIO BETWEN MODEL 40 ANC 50                       0.476777
  18        I      CPU WAIT MODEL 50                                           222.710855           75.C0 PCT OF CCUNTER 16
  19        J      CPU WAIT CNLY MCDEL 5C                                      120.896862           40.71 PCT OF COUNTER 16
  20        K      CPU ACTIVE ONLY MODEL 50                                     52.793996           17.78 PCT OF COUNTER 16
  21        L      NEW ANY CHANNEL BUSY MCDEL 50                               124.578991           41.77 PCT OF COUNTER 22
  22        M      NEW SYSTEM TIME MCDEL 50                                    298.269850
  23        N      NEW ANY CHANNEL BUSY AND WAIT MODEL 50                      103.143993           34.58 PCT OF CCUNTER 22
  24        O      NEW WAIT MODEL 50                                           224.040855           75.11 PCT OF COUNTER 22
  25        P      INCREASE IN MODEL 5C SYSTEM TIME                              1.330000            0.45 PCT OF COUNTER 16
-------------------------------------------------------------------------------------------------------------------------------------
```

Figure 3—Interval summary of event activity

```
MEASURED 1C/ 6/7C      SYSTEM/36C MODEL 4C ANC 5C ANALYSIS


*****************************************************************************************************************************
                                                  INTERMEDIATE/FINAL SUMMARY
*****************************************************************************************************************************


FILE SUMMARY OVER PREVIOUS    3CCC.C SECCNDS, BEGINNING AT  8.30. C.0 ANC ENCING AT  9.2C. C.C

COUNTER    ID      DESCRIPTICN      BCARC IC C    JCB ID 24                    CCUNTER VALUE       PERCENT                BASE

   C        C      TCTAL ELAPSED TIME                                          2969.572795       100.00 PCT CF CCUNTER 16
   1        1      CPU ACTIVE MODEL 50                                          831.427943        28.00 PCT CF CCUNTER 16
   2        2      PRCBLEM STATE MODEL 50                                      1577.766891        53.13 PCT CF CCUNTER 16
   3        3      ELAPSED TIME METER RUNNING MCDEL 4C                         2913.025799        98.1C PCT OF CCUNTER 16
   4        4      SELECTOR CHANNEL 1 BUSY MCDEL 50                             926.357936        31.2C PCT OF CCUNTER 16
   5        5      SELECTOR CHANNEL 2 BUSY MCCEL 50                              19.126999         C.64 PCT CF COUNTER 16
   6        6      ANY CHANNEL BUSY MCDEL 5C                                    941.699935        31.71 PCT CF CCUNTER 16
   7        7      ANY CHANNEL BUSY AND CPU WAIT MCCEL 50                       714.257951        24.05 PCT CF CCUNTER 16
   8        8      CHANNEL 1 AND 2 OVERLAP MCDEL 50                               3.761000         0.13 PCT OF CCUNTER 16
   9        9      2314 BUSY TC MCDEL 40                                        269.897981         9.09 PCT CF CCUNTER 16
  10        A      2314 BUSY TC MODEL 5C                                        924.381936        31.13 PCT CF CCUNTER 16
  11        B      MCDEL 40 HAS 2314,MODEL 5C WANTS                              59.967996         2.02 PCT CF CCUNTER 16
  12        C      MCDEL 5C HAS 2314,MCDEL 4C WANTS                             139.837990         4.71 PCT CF CCUNTER 16
  13        D      MCDEL 40 USING 2314,MCDEL 5C WAIT CNLY                         C.233000         0.C1 PCT OF CCUNTER 16
  14        E      MODEL 50 IN MANUAL STATE                                       0.C              0.C  PCT CF CCUNTER 16
  15        F      PROBLEM STATE ANC MODEL 50 CPU ACTIVE                        255.095982         8.59 PCT CF CCUNTER 16
  16        G      ************ SCFTWARE CLCCK ************                    2969.456699       100.00 PCT CF CCUNTER 16
  17        H      CONTENTICN RATIC BETWEN MCDEL 40 ANC 5C                        0.428839
  18        I      CPU WAIT MODEL 50                                           2138.028756        72.00 PCT CF CCUNTER 16
  19        J      CPU WAIT ONLY MCDEL 50                                      1423.770805        47.95 PCT CF CCUNTER 16
  20        K      CPU ACTIVE CNLY MODEL 5C                                     603.985958        20.34 PCT CF CCUNTER 16
  21        L      NEW ANY CHANNEL BUSY MCDEL 50                                945.484935        31.80 PCT CF CCUNTER 22
  22        M      NEW SYSTEM TIME MODEL 5C                                    2973.241698
  23        N      NEW ANY CHANNEL BUSY ANC WAIT MODEL 50                       718.042951        24.15 PCT OF CCUNTER 22
  24        O      NEW WAIT MODEL 50                                           2141.813756        72.04 PCT OF CCUNTER 22
  25        P      INCREASE IN MODEL 5C SYSTEM TIME                               3.785000         0.13 PCT CF CCUNTER 16


*****************************************************************************************************************************
                                                  INTERMEDIATE/FINAL SUMMARY
*****************************************************************************************************************************
```

Figure 4—Final summary of event activity

```
      STATISTICAL SUMMARY FOR    12 CBSERVATIONS, BEGINNING AT       0.0 AND ENDING AT 999999.9 SECCNDS.

      EACH CBSERVATION REPRESENTS THE ABSCLUTE INCREASE IN COUNTER VALUE  SINCE THE PREVICUS OBSERVATION.

CCUNTER        DESCRIPTION                                    MINIMUM          MAXIMUM          MEAN            S.D.

   0     TCTAL ELAPSEC TIME                                296.935980       297.009980       296.953896        0.019107
   1     CPU ACTIVE MODEL 50                                 7.996999       175.433988        71.048828       40.751117
   2     PRCBLEM STATE MODEL 50                              5.003000       259.682986       132.905824       68.383099
   3     ELAPSED TIME METER RUNNING MODEL 40                42.751997       296.949980       253.088916       86.876050
   4     SELECTOR CHANNEL 1 BUSY MCDEL 50                   13.901999       121.785992        80.687244       32.079867
   5     SELECTOR CHANNEL 2 BUSY MCDEL 50                    0.0             6.476000         1.593917        2.542871
   6     ANY CHANNEL BUSY MCDEL 50                          13.901999       123.248992        81.965661       33.014596
   7     ANY CHANNEL BUSY AND CPU WAIT MODEL 50             12.465999       102.350993        62.647496       27.212985
   8     CHANNEL 1 AND 2 OVERLAP MCDEL 50                    0.0             1.326000         0.313417        0.492087
   9     2314 BUSY TO MODEL 40                               0.0            89.715994        34.665581       32.271344
  10     2314 BUSY TO MODEL 50                              13.906999       121.835992        80.078161       32.818537
  11     MCDEL 40 HAS 2314,MODEL 50 WANTS                    0.0            19.926999         5.099083        6.472185
  12     MCDEL 50 HAS 2314,MODEL 40 WANTS                    0.0            41.016997        11.808083       14.261126
  13     MCDEL 40 USING 2314,MODEL 50 WAIT CNLY              0.0            89.536994        11.207499       26.629362
  14     MCDEL 50 IN MANUAL STATE                            0.0             0.0              0.0             0.0
  15     PROBLEM STATE AND MODEL 50 CPU ACTIVE               0.415000       131.820991        21.336665       35.065729
  16     ************ SOFTWARE CLCCK ************          296.939850       296.998050       296.944700        0.016086
  17     CONTENTICN RATIO BETWEN MCDEL 40 AND 50             0.0             0.656805         0.335766        0.211528
  18     CPU WAIT MODEL 50                                 121.505862       288.942850       225.895871       40.755710
  19     CPU WAIT CNLY MODEL 50                             69.401866       276.476851       163.248376       56.869922
  20     CPU ACTIVE ONLY MCDEL 50                            6.561000       132.226991        51.730663       30.229317
  21     NEW ANY CHANNEL BUSY MODEL 50                      13.901999       124.578991        82.281161       33.267174
  22     NEW SYSTEM TIME MCDEL 50                          296.939850       298.269850       297.250200        0.491280
  23     NEW ANY CHANNEL BUSY AND WAIT MCDEL 50             12.465999       103.143993        62.962996       27.503927
  24     NEW WAIT MODEL 50                                 121.505862       288.942850       226.211371       40.723085
  25     INCREASE IN MODEL 50 SYSTEM TIME                    0.0             1.330000         0.315500        0.492668
```

Figure 5—Statistical summary of event activity

8. CPU ACTIVE ONLY
   (CPU ACTIVE $\wedge$ $\overline{\text{ANY CHANNEL BUSY}}$)
9. CPU ACTIVE DURING PROGRAM STATUS WORD (CPU ACTIVE $\wedge$ DECODE OF PSW) e.g., PROBLEM STATE, SUPERVISOR, PROGRAM, etc.
10. I/O ACTIVE ONLY
    (CPU RUN $\wedge$ CPU WAIT $\wedge$ $\overline{\text{CPU MANUAL}}$)
11. SYSTEM WAIT
    ($\overline{\text{CPU RUN}}$ $\wedge$ CPU WAIT $\wedge$ $\overline{\text{CPU MANUAL}}$) or (TOTAL SYSTEM TIME-CPU ACTIVE)
12. SEEK WITHIN DATA SET
    (SEEK $\wedge$ WITHIN DATA SET BOUNDARIES)
13. INSTRUCTION WITHIN PARTITION
    (INSTRUCTION FETCH WITHIN PARTITION BOUNDARIES)

The next section shows the results of system event monitoring.

RESULTS

Typical results obtained from monitoring the system events given in the previous section can be shown by reproducing actual output from a data reduction program. Figure 2 shows the description of the events monitored and combined. Figure 3 shows an interval summary of event activity. Figure 4 shows a final summary of the event activity. Figure 5 shows a statistical summary of the events. Graphic results are easily correlated with various activities and give a clear picture of the sequence of events. Figures 6-10 show the histograms of CPU ACTIVE, SELECTOR CHANNEL 1 BUSY, SELECTOR CHANNEL 2 BUSY, ANY CHANNEL BUSY ONLY, and CPU WAIT.

The next section deals with the interpretation of these results.

INTERPRETATION

The areas which can be investigated to optimize throughput or cost of a computing system are system configuration, programs, routines resident/non-resident, data set allocation, job scheduling and operation methods. In a multiprogram environment, throughput is a measure of the time required to process a fixed amount of work or simply stated the number of jobs per day. A good assumption is that the CPU processing time is constant whether the jobs are run sequentially or multijobbed in an interweaving process. The improved throughput by multijobbing should come by overlapping system resources, e.g., CPU activity on one job overlapped with I/O activity on another job. This increases the percentage of time the CPU is active to yield better system utilization. Unfortunately, the desired positive effects of multiprogramming are not always obtained, e.g., competition may exist between two different jobs for the same direct access device. It

```
MEASURED 10/ 6/70     SYSTEM/360 MODEL 40 AND 50 ANALYSIS

                                            CPU ACTIVE MODEL 50                                    (BASE=0)

                 0         10        20        30        40        50        60        70        80 .      90        100
JCB       SECS.  x         x         x         x         x         x         x         x         x         x         x  TIME
                 *****************************************************************************************************
24         90.0  11111111  .         .         .         .         .         .         .         .         .         .  8.31.30.0
24        180.0  111111111111111111111111      .         .         .         .         .         .         .         .  8.33. 0.0
24        270.0  11111111111111111111111111111111111111111111      .         .         .         .         .         .  8.34.30.0
24        360.0  111111111111111111111111111111.         .         .         .         .         .         .         .  8.36. 0.0
24        450.0  11111111111111111111111111111 .         .         .         .         .         .         .         .  8.37.30.0
24        540.0  111111111111111111111111111111111       .         .         .         .         .         .         .  8.39. 0.0
24        630.0  111111111 .         .         .         .         .         .         .         .         .         .  8.40.30.0
24        720.0  1111111111111111    .         .         .         .         .         .         .         .         .  8.42. 0.0
24        810.0  111111111111111111111111111111111111111111111     .         .         .         .         .         .  8.43.30.0
24        900.0  11111111111111111111111111111111        .         .         .         .         .         .         .  8.45. 0.0
24        990.0  1111111111111111111111111111  .         .         .         .         .         .         .         .  8.46.30.0
24       1080.0  11111111111111111111111111111111        .         .         .         .         .         .         .  8.48. 0.0
24       1170.0  111111111111111     .         .         .         .         .         .         .         .         .  8.49.30.0
24       1260.0  1111111111111111111111        .         .         .         .         .         .         .         .  8.51. 0.0
24       1350.0  11111111111111111111111111111111111111111111111111111111111 .         .         .         .         .  8.52.30.0
24       1440.0  1111111111111111111111111111111111111111111111111111111111111111111111111       .         .         .  8.54. 0.0
24       1530.0  1111111111111111111111111111111111111111111111111111111111111111111   .         .         .         .  8.55.30.0
24       1620.0  111111111111111111111111111   .         .         .         .         .         .         .         .  8.57. 0.0
24       1710.0  1111111111111111111111        .         .         .         .         .         .         .         .  8.58.30.0
24       1800.0  1111111111111111111111        .         .         .         .         .         .         .         .  9. 0. 0.0
24       1890.0  111111111111111111111111      .         .         .         .         .         .         .         .  9. 1.30.0
24       1980.0  111111111111111111111111      .         .         .         .         .         .         .         .  9. 3. 0.0
24       2070.0  111111111111111111111111      .         .         .         .         .         .         .         .  9. 4.30.0
24       2160.0  11111111111111111111111111    .         .         .         .         .         .         .         .  9. 6. 0.0
24       2250.0  1111111111111111111111111     .         .         .         .         .         .         .         .  9. 7.30.0
24       2340.0  111111111111111111111111      .         .         .         .         .         .         .         .  9. 9. 0.0
24       2430.0  111111111111111111111111111111111       .         .         .         .         .         .         .  9.10.30.0
24       2520.0  11111111111111111111111111111111111111111111      .         .         .         .         .         .  9.12. 0.0
24       2610.0  11111111111111111111111111111111111     .         .         .         .         .         .         .  9.13.30.0
24       2700.0  11111111  .         .         .         .         .         .         .         .         .         .  9.15. 0.0
24       2790.0  11111111111111111   .         .         .         .         .         .         .         .         .  9.16.30.0
24       2880.0  1111111111111       .         .         .         .         .         .         .         .         .  9.18. 0.0
24       2970.0  111111111 .         .         .         .         .         .         .         .         .         .  9.19.30.0
24       3060.0  1111      .         .         .         .         .         .         .         .         .         .  9.21. 0.0
24       3150.0  1         .         .         .         .         .         .         .         .         .         .  9.22.30.0
24       3240.0  11111     .         .         .         .         .         .         .         .         .         .  9.24. 0.0
24       3330.0  111111111111111111  .         .         .         .         .         .         .         .         .  9.25.30.0
24       3420.0  111       .         .         .         .         .         .         .         .         .         .  9.27. 0.0
24       3510.0  1         .         .         .         .         .         .         .         .         .         .  9.28.30.0
24       3600.0  1         .         .         .         .         .         .         .         .         .         .  9.30. 0.0
                 *****************************************************************************************************
JCB       SECS.  x         x         x         x         x         x         x         x         x         x         x  TIME
                 0         10        20        30        40        50        60        70        80        90        100
```

Figure 6—Histogram of CPU active

is crucial in any performance optimizing cycle that the system resources be overlapped for the job stream and that competition for resources be eliminated. This can have far greater effects on throughput and cost than on increasing the speed of the system components. Indeed, it is often possible to increase throughput while at the same time returning or delaying purchase of system components, or going to slower components.

System performance profile

In order to look at the overlapping of system resources, a system performance profile which shows the activity of the CPU, channels and the amount of overlapping between them is constructed as shown in Figure 11. The system performance profile shows the overall system utilization. It may be immediately apparent that some of the system components are essentially unused.

Basic indicators

Some of the basic indicators to look for in interpreting a system performance profile are:

SMALL CHANNEL OVERLAP,
CHANNEL IMBALANCE,
HIGH CHANNEL UTILIZATION,
LARGE WAIT ONLY, and
LARGE CPU ACTIVE ONLY

1. SMALL CHANNEL OVERLAP

The probable cause for small channel overlap when the channel utilization is high is poor device placement on the channels resulting in sequential operations. The control units and devices should be monitored and the job stream examined in order to determine the data sets used. The device

```
MEASURED 1C/ 6/7C      SYSTEM/36C MODEL 4C AND 5C ANALYSIS

                                               SELECTCR CHANNEL 1 BUSY MCDEL 50               (BASE=G)

                0         1C        20        3C        40        50        6C        7C        8C        90       1CC
JCB      SECS.  X         X         X         X         X         X         X         X         X         X         X  TIME
                *****************************************************************************************************
24        90.0  444444    .         .         .         .         .         .         .         .         .         .  8.31.30.C
24       180.C  444444444444444444444444444444.         .         .         .         .         .         .         .  8.33. 0.C
24       270.0  44444444444444444444444444444444444444  .         .         .         .         .         .         .  8.34.30.0
24       360.0  4444444444444444444444444     .         .         .         .         .         .         .         .  8.36. C.0
24       45C.C  444444444444444444444444444   .         .         .         .         .         .         .         .  8.37.30.0
24       540.C  444444444444444444444444444444444444    .         .         .         .         .         .         .  8.39. 0.0
24       630.0  444444444 .         .         .         .         .         .         .         .         .         .  8.40.30.0
24       72C.C  4444444444444444444444        .         .         .         .         .         .         .         .  8.42. 0.0
24       81C.0  444444444444444444444444444444444444    .         .         .         .         .         .         .  8.43.30.C
24       900.0  4444444444444444444444444444444444444   .         .         .         .         .         .         .  8.45. C.0
24       990.C  44444444444444444444444444444444444444  .         .         .         .         .         .         .  8.46.3C.C
24      1080.0  44444444444444444444444444444444444     .         .         .         .         .         .         .  8.48. 0.0
24      1170.0  44444444444444444444.         .         .         .         .         .         .         .         .  8.49.30.0
24      1260.0  4444444444444444444444444444444444      .         .         .         .         .         .         .  8.51. 0.0
24      1350.0  4444444444444444444444444444444444444444.         .         .         .         .         .         .  8.52.30.0
24      1440.0  4444444444444444444 .         .         .         .         .         .         .         .         .  8.54. 0.0
24      1530.0  4444444444444444444444444444444444444444.         .         .         .         .         .         .  8.55.30.0
24      1620.0  4444444444444444444444444444444444444444.         .         .         .         .         .         .  8.57. 0.0
24      1710.C  4444444444444444444444444444444         .         .         .         .         .         .         .  8.58.30.0
24      1800.0  44444444444444444444444444444444444     .         .         .         .         .         .         .  9. 0. C.0
24      1890.0  444444444444444444444444444444444444444444444     .         .         .         .         .         .  9. 1.30.0
24      1980.0  444444444444444444444444444444444444444444        .         .         .         .         .         .  9. 3. C.0
24      2070.C  444444444444444444444444444444444444444 .         .         .         .         .         .         .  9. 4.30.0
24      2160.C  44444444444444444444444444444444444444444         .         .         .         .         .         .  9. 6. 0.C
24      2250.0  44444444444444444444444444444444444444444         .         .         .         .         .         .  9. 7.30.0
24      2340.C  4444444444444444444444444444444444444   .         .         .         .         .         .         .  9. 9. 0.0
24      2430.0  444444444444444444444444444444444444444444444     .         .         .         .         .         .  9.10.30.0
24      2520.0  44444444444444444444444444444444444444  .         .         .         .         .         .         .  9.12. 0.C
24      2610.0  44444444444444444444444444444444444444444444444   .         .         .         .         .         .  9.13.30.C
24      2700.0  444444444444444     .         .         .         .         .         .         .         .         .  9.15. 0.0
24      2790.0  444444444444444444444444444444444444    .         .         .         .         .         .         .  9.16.30.0
24      2880.0  4444444444444444444 .         .         .         .         .         .         .         .         .  9.18. 0.0
24      2970.0  4444444   .         .         .         .         .         .         .         .         .         .  9.19.30.0
24      3060.0  4444444   .         .         .         .         .         .         .         .         .         .  9.21. 0.0
24      3150.0  4         .         .         .         .         .         .         .         .         .         .  9.22.30.0
24      3240.0  4444      .         .         .         .         .         .         .         .         .         .  9.24. C.0
24      3330.0  4444444444444444444444444444444444444444.         .         .         .         .         .         .  9.25.30.0
24      3420.0  44444     .         .         .         .         .         .         .         .         .         .  9.27. 0.0
24      3510.C  4         .         .         .         .         .         .         .         .         .         .  9.28.30.0
24      3600.0  4         .         .         .         .         .         .         .         .         .         .  9.30. 0.0
                *****************************************************************************************************
JOB      SECS.  X         X         X         X         X         X         X         X         X         X         X  TIME
                0         1C        20        30        40        50        60        7C        80        90       1CO
```

Figure 7—Histogram of selector channel 1 busy

and data set information will provide sufficient data so that a rearrangement of devices and data sets to produce better balance can be achieved. A new system performance profile should be constructed to verify the change.

A small channel overlap when the channel utilization is low, means that all the work can be put on one channel with little effect on the job stream processing time.

2. CHANNEL IMBALANCE

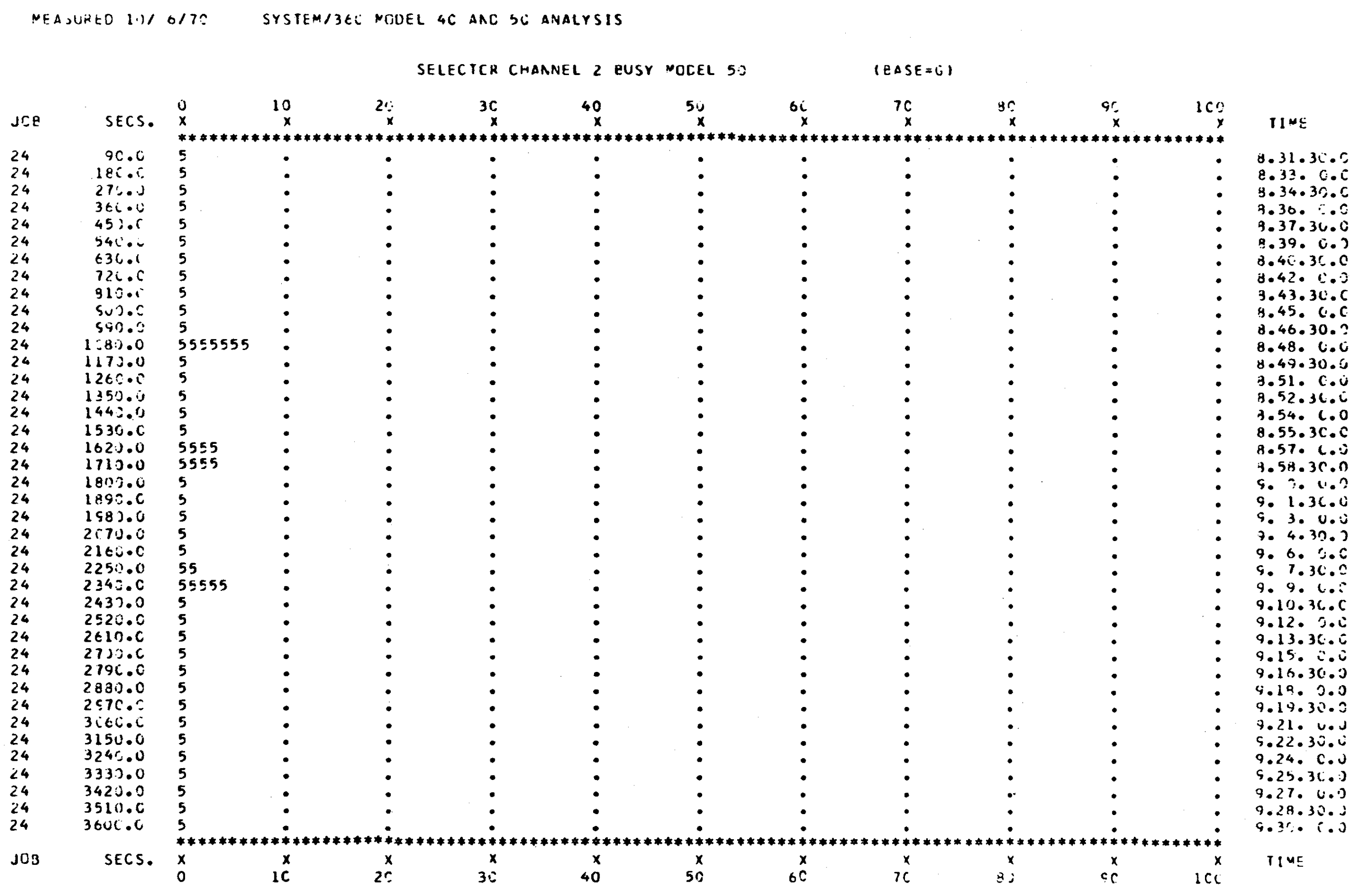
If the channel utilization is high, but the channel load is not balanced, the device activity needs to be measured to determine device rearranging. A new system performance should be constructed to verify the results.

Low channel utilization would indicate that all the work can be placed on one channel with little effect on the system throughput.

3. HIGH CHANNEL UTILIZATION

The system data sets should be examined. There may be a problem as to which routines are resident/non-resident. A measurement should be made to determine transfer time of system routines relative to total device active time. If the transfer time is high, make all system routines non-resident and measure their activity to determine which routines to make resident/non-resident.

Another possible cause is record blocking in data sets on direct access devices. Measure the I/O device utilizations and examine the data sets on each device to locate ones in which a larger number of records could be placed in each block to increase the efficiency of access.



```
MEASURED 10/ 6/70      SYSTEM/360 MODEL 40 AND 50 ANALYSIS

                                      SELECTOR CHANNEL 2 BUSY MODEL 50          (BASE=0)

               0          10         20         30         40         50         60         70         80         90         100
JOB     SECS.  X          X          X          X          X          X          X          X          X          X          X    TIME
               ***************************************************************************************************************
24      90.0   5          .          .          .          .          .          .          .          .          .          .    8.31.30.0
24     180.0   5          .          .          .          .          .          .          .          .          .          .    8.33. 0.0
24     270.0   5          .          .          .          .          .          .          .          .          .          .    8.34.30.0
24     360.0   5          .          .          .          .          .          .          .          .          .          .    8.36. 0.0
24     450.0   5          .          .          .          .          .          .          .          .          .          .    8.37.30.0
24     540.0   5          .          .          .          .          .          .          .          .          .          .    8.39. 0.0
24     630.0   5          .          .          .          .          .          .          .          .          .          .    8.40.30.0
24     720.0   5          .          .          .          .          .          .          .          .          .          .    8.42. 0.0
24     810.0   5          .          .          .          .          .          .          .          .          .          .    8.43.30.0
24     900.0   5          .          .          .          .          .          .          .          .          .          .    8.45. 0.0
24     990.0   5          .          .          .          .          .          .          .          .          .          .    8.46.30.0
24    1080.0   5555555    .          .          .          .          .          .          .          .          .          .    8.48. 0.0
24    1170.0   5          .          .          .          .          .          .          .          .          .          .    8.49.30.0
24    1260.0   5          .          .          .          .          .          .          .          .          .          .    8.51. 0.0
24    1350.0   5          .          .          .          .          .          .          .          .          .          .    8.52.30.0
24    1440.0   5          .          .          .          .          .          .          .          .          .          .    8.54. 0.0
24    1530.0   5          .          .          .          .          .          .          .          .          .          .    8.55.30.0
24    1620.0   5555       .          .          .          .          .          .          .          .          .          .    8.57. 0.0
24    1710.0   5555       .          .          .          .          .          .          .          .          .          .    8.58.30.0
24    1800.0   5          .          .          .          .          .          .          .          .          .          .    9. 0. 0.0
24    1890.0   5          .          .          .          .          .          .          .          .          .          .    9. 1.30.0
24    1980.0   5          .          .          .          .          .          .          .          .          .          .    9. 3. 0.0
24    2070.0   5          .          .          .          .          .          .          .          .          .          .    9. 4.30.0
24    2160.0   5          .          .          .          .          .          .          .          .          .          .    9. 6. 0.0
24    2250.0   55         .          .          .          .          .          .          .          .          .          .    9. 7.30.0
24    2340.0   55555      .          .          .          .          .          .          .          .          .          .    9. 9. 0.0
24    2430.0   5          .          .          .          .          .          .          .          .          .          .    9.10.30.0
24    2520.0   5          .          .          .          .          .          .          .          .          .          .    9.12. 0.0
24    2610.0   5          .          .          .          .          .          .          .          .          .          .    9.13.30.0
24    2700.0   5          .          .          .          .          .          .          .          .          .          .    9.15. 0.0
24    2790.0   5          .          .          .          .          .          .          .          .          .          .    9.16.30.0
24    2880.0   5          .          .          .          .          .          .          .          .          .          .    9.18. 0.0
24    2970.0   5          .          .          .          .          .          .          .          .          .          .    9.19.30.0
24    3060.0   5          .          .          .          .          .          .          .          .          .          .    9.21. 0.0
24    3150.0   5          .          .          .          .          .          .          .          .          .          .    9.22.30.0
24    3240.0   5          .          .          .          .          .          .          .          .          .          .    9.24. 0.0
24    3330.0   5          .          .          .          .          .          .          .          .          .          .    9.25.30.0
24    3420.0   5          .          .          .          .          .          .          .          .          .          .    9.27. 0.0
24    3510.0   5          .          .          .          .          .          .          .          .          .          .    9.28.30.0
24    3600.0   5          .          .          .          .          .          .          .          .          .          .    9.30. 0.0
               ***************************************************************************************************************
JOB     SECS.  X          X          X          X          X          X          X          X          X          X          X    TIME
               0          10         20         30         40         50         60         70         80         90         100
```

Figure 8—Histogram of selector channel 2 busy

4. LARGE WAIT ONLY

If the profile shows a large amount of WAIT ONLY, measure the SEEK ONLY. A large portion of SEEK ONLY time of the WAIT$\wedge\overline{\text{ANY CHANNEL BUSY}}$ time, means the system is waiting for seeks to complete. This indicates a poor data set placement. The direct access devices should be measured to find the cause of arm contention. The AVERAGE SEEK TIME can be determined by measuring the SEEK TIME and count of the NUMBER OF SEEKS on each device. This information should be correlated with the console log to determine which programs and hence which data sets caused the arm contention. Measurements can also be made of SEEK WITHIN DATA SET to determine the number of seeks and the duration of seeks within each data set. Partitioned data sets can be examined to see if there is excessive arm movement between the sequential sets. SEEK TIME can be reduced by rearranging the data sets on the same pack or moving them to different disk packs.

If the SEEK ONLY is insignificant, operation problems are indicated. Possible causes are difficult operator set up procedures, too few operators, poor job scheduling, etc. Measure the amount of NOT READY TIME which occurs on each device during the day. This can indicate operation problems or equipment malfunction. A large amount of WAIT ONLY TIME often occurs at shift changes.

5. LARGE CPU ACTIVE ONLY

A large CPU ACTIVE ONLY time coupled with a low CPU-CHANNEL OVERLAP indicates the benefits of multiprogramming are not being obtained. Possible reasons are poor job scheduling (a balance is needed between CPU and I/O bound jobs); poor data set allocation (data set should

```
MEASURED 10/ 6/7C       SYSTEM/36C MODEL 4C ANC 5U ANALYSIS

                              ANY CHANNEL BUSY AND CPU WAIT MCDEL 5C        (BASE=G)

                 0         10        2U        3C        40        50        60        70        80        90        1CC
JCB       SECS.  X         X         X         X         X         X         X         X         X         X         X  TIME
                 *****************************************************************************************************
24         9C.0  777777    .         .         .         .         .         .         .         .         .         .  8.31.30.0
24        18C.0  77777777777777777777777       .         .         .         .         .         .         .         .  8.33. C.0
24        270.0  777777777777777777777         .         .         .         .         .         .         .         .  8.34.30.C
24        360.0  77777777777777      .         .         .         .         .         .         .         .         .  8.36. C.C
24        45C.C  7777777777777777    .         .         .         .         .         .         .         .         .  8.37.3C.0
24        54C.0  77777777777777777777777777    .         .         .         .         .         .         .         .  8.39. 0.0
24        630.0  77777777  .         .         .         .         .         .         .         .         .         .  8.40.3C.0
24        72C.0  7777777777777777777 .         .         .         .         .         .         .         .         .  8.42. C.C
24        810.0  77777777777777777777777777    .         .         .         .         .         .         .         .  8.43.3C.C
24        900.C  77777777777777777777777777777 .         .         .         .         .         .         .         .  8.45. 0.0
24        990.0  7777777777777777777777777777777         .         .         .         .         .         .         .  8.46.30.0
24       1C80.C  777777777777777777777777777777.         .         .         .         .         .         .         .  8.48. C.0
24       1170.0  77777777777777777   .         .         .         .         .         .         .         .         .  8.49.3C.0
24       1260.C  77777777777777777777777777777 .         .         .         .         .         .         .         .  8.51. C.0
24       1350.0  77777777777777777777.         .         .         .         .         .         .         .         .  8.52.3C.0
24       1440.0  77777777777         .         .         .         .         .         .         .         .         .  8.54. C.0
24       1530.C  77777777777777777777.         .         .         .         .         .         .         .         .  8.55.3C.0
24       1620.C  77777777777777777777777777777777777     .         .         .         .         .         .         .  8.57. C.C
24       1710.C  77777777777777777777777777777 .         .         .         .         .         .         .         .  8.58.3C.0
24       1800.0  77777777777777777777777777777 .         .         .         .         .         .         .         .  9. 0. 0.0
24       1890.C  77777777777777777777777777777777777777  .         .         .         .         .         .         .  9. 1.30.0
24       1980.0  777777777777777777777777777777777777    .         .         .         .         .         .         .  9. 3. 0.0
24       2C70.C  777777777777777777777777777777777       .         .         .         .         .         .         .  9. 4.30.0
24       2160.0  7777777777777777777777777777777777      .         .         .         .         .         .         .  9. 6. C.0
24       2250.0  77777777777777777777777777777777777     .         .         .         .         .         .         .  9. 7.30.0
24       2340.0  7777777777777777777777777777777777      .         .         .         .         .         .         .  9. 9. C.C
24       2430.0  7777777777777777777777777777777777      .         .         .         .         .         .         .  9.10.30.0
24       252C.C  77777777777777777777777777    .         .         .         .         .         .         .         .  9.12. 0.0
24       2610.0  777777777777777777777777777777777777    .         .         .         .         .         .         .  9.13.30.C
24       2700.0  77777777777777      .         .         .         .         .         .         .         .         .  9.15. 0.0
24       279C.C  77777777777777777777777777777777        .         .         .         .         .         .         .  9.16.30.0
24       2880.0  77777777777777777   .         .         .         .         .         .         .         .         .  9.18. C.0
24       2970.0  777777    .         .         .         .         .         .         .         .         .         .  9.19.30.C
24       3C60.0  777777    .         .         .         .         .         .         .         .         .         .  9.21. 0.0
24       3150.C  7         .         .         .         .         .         .         .         .         .         .  9.22.30.0
24       324C.0  777       .         .         .         .         .         .         .         .         .         .  9.24. 0.0
24       3330.0  777777777777777777777777777777777777    .         .         .         .         .         .         .  9.25.30.0
24       342C.0  7777      .         .         .         .         .         .         .         .         .         .  9.27. 0.0
24       351C.C  7         .         .         .         .         .         .         .         .         .         .  9.28.30.0
24       360C.C  7         .         .         .         .         .         .         .         .         .         .  9.30. 0.0
                 *****************************************************************************************************
JOB       SECS.  X         X         X         X         X         X         X         X         X         X         X  TIME
                 C         1C        2C        3C        40        5C        6C        7C        80        9C        1CC
```

Figure 9—Histogram of any channel busy only

be on different channels so they do not have to be retrieved sequentially), or inefficiently written programs. Several of the production jobs which use a large part of the system resources should be selected and run stand alone in the system to create an individual job profile. These profiles show data set usage to make data set placement changes and processing phases which are CPU or I/O bound to enable efficient job scheduling. This will increase CPU utilization and system throughput.

If the system performance still shows a large amount of CPU ACTIVE ONLY time, code optimization of the programs which contribute most to the system load should be undertaken. Comparators can be used to gather statistics of frequency distribution of instructions or set of instructions, branches, iteration in data dependent loops, and percentage time in subroutines. From the statistics gathered, the programmer can see where efforts should be directed for code optimization.

Predictive methods for system reconfiguration

It is possible to evaluate system configurations and calculate the effect on the job stream time resulting from that reconfiguration. The basic measured data is used in equations which define the new configurations to calculate a new job stream time. This new job stream time can be compared with the old job stream time from the measurement, to see if the change produced the desired effect.

1. SYSTEM TIME EQUATION

The time to process a job stream is composed of the overlapped and unoverlapped time of each of the units in the computing system. There are many alternate ways of expressing an equation for the system time. For example, the system time for the system profile of Figure 11 can be expressed by the sum of any of the nonoverlapping times which equal the total system time, e.g., SYSTEM

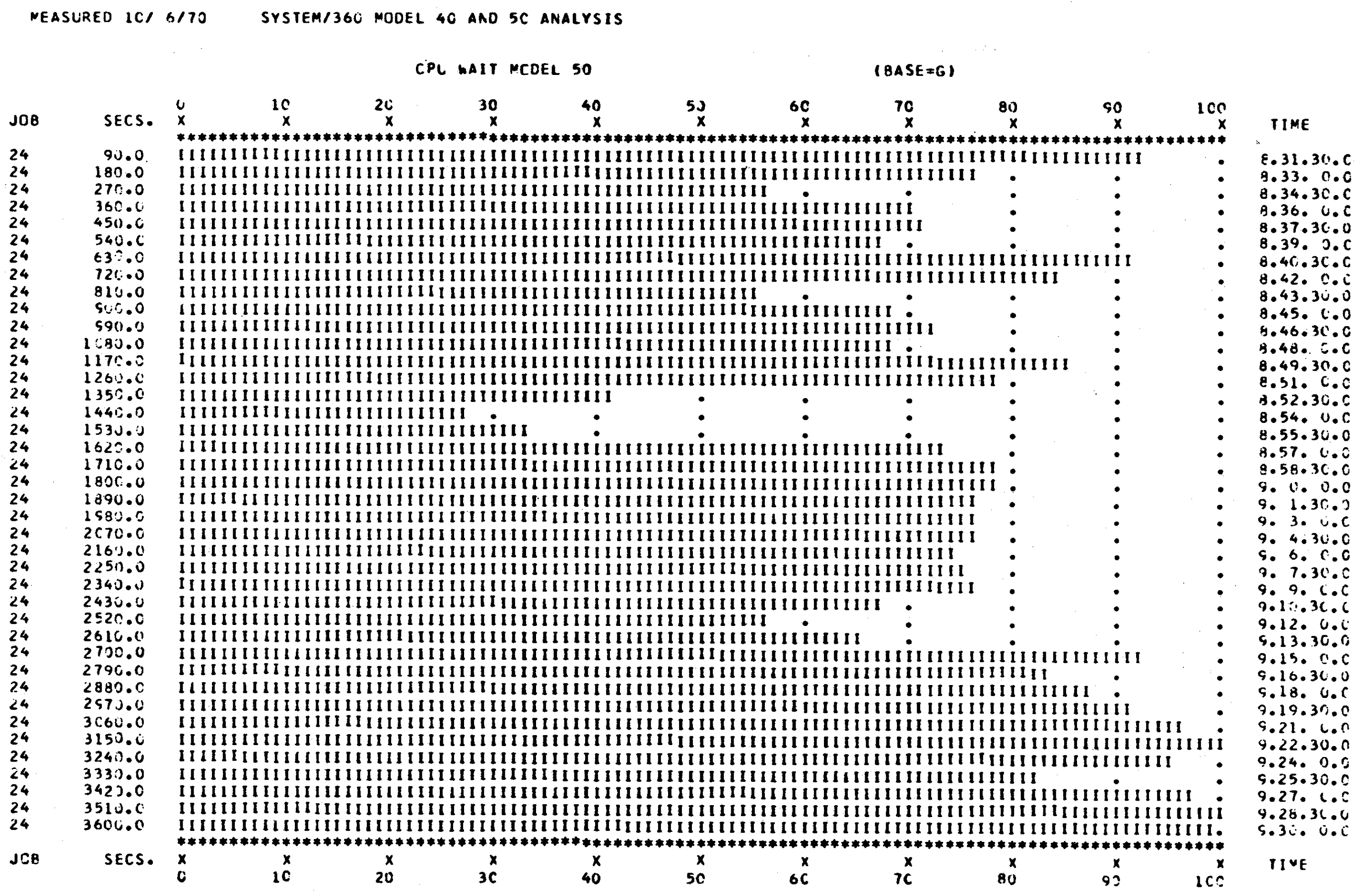


Figure 10—Histogram of CPU wait

TIME=CPU ACTIVE ONLY TIME+ANY CHANNEL BUSY TIME+CPU WAIT ONLY TIME. The system time equation forms the basis for estimating the effect of all configuration changes.

2. THROUGHPUT TIME REDUCTION

A change in the speed of a unit only affects the throughput by reducing the nonoverlapped time of that unit. This assumption is a good approximation of the system behavior and will be used throughout the analysis to calculate the effect of configuration changes. The resulting reduction in the overlapped time does not affect the overall system time. The relationships of the components will vary but the sum of the times of the unaffected units will not change. As an example, the substitution of a faster CPU will have the effect of reducing the total system time by the amount of time that the CPU ACTIVE ONLY TIME is reduced. In

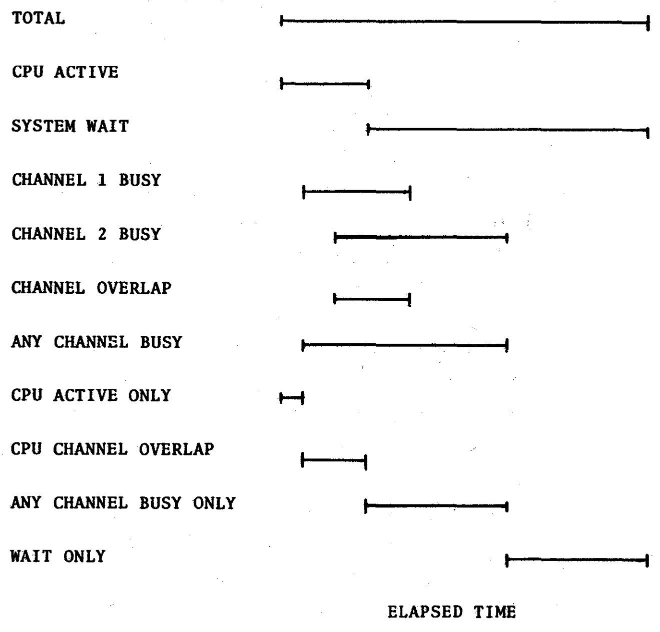


Figure 11—System performance profile

order to calculate the effect of increasing the speed of a component, a measurement, must be made which isolates the nonoverlapped portion of that component. Thus, to calculate the effect of increasing the CPU speed one must make a measurement that will isolate the CPU ACTIVE ONLY TIME, and for increased device speed, the measurement must isolate ANY CHANNEL BUSY ONLY TIME. Improvement factors will be applied to these nonoverlapped times and then added to the other times which make up the system time to calculate a new system time.

Configuration change equations

1. INCREASED CPU SPEED

The CPU ACTIVE ONLY TIME, ANY CHANNEL BUSY TIME and WAIT ONLY TIME are measured. Then an improvement factor for the increase in the CPU speed is used to modify the system time equation. For a CPU which is twice as fast, the improvement factor is 2. The new system time is NEW SYSTEM TIME=CPU ACTIVE ONLY TIME/IMPROVEMENT FACTOR+ANY CHANNEL BUSY TIME+CPU WAIT ONLY TIME.

This new system time is the time to process the job stream that was measured by the monitor.

2. INCREASED DEVICE SPEED

The CPU ACTIVE TIME, ANY CHANNEL BUSY ONLY TIME and CPU WAIT ONLY TIME are measured. Then an improvement factor for the increased device speed is used, e.g., for direct access devices use the ratio of the new average rotational delay divided by the old average rotational delay. Next calculate a new system time by the following equation: NEW SYSTEM TIME=CPU ACTIVE TIME+ANY CHANNEL BUSY ONLY TIME/IMPROVEMENT FACTOR+CPU WAIT ONLY TIME.

3. INCREASED/DECREASED SEEK SPEEDS

The CPU ACTIVE ONLY TIME, ANY CHANNEL BUSY TIME, SEEK ONLY TIME, and WAIT ONLY TIME are measured. Figure 12 shows the measured times. An improvement factor for the change in seek speed is used, e.g., the ratio of the new average seek time divided by the old average seek time. The new system time is given by the equation: NEW SYSTEM TIME=CPU ACTIVE ONLY TIME+ANY CHANNEL BUSY TIME+SEEK ONLY TIME/IMPROVEMENT FACTOR+CPU WAIT ONLY TIME.

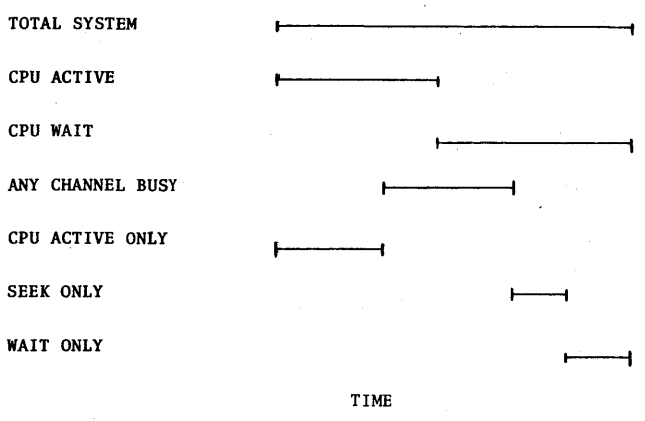


Figure 12—System profile including I/O measurement

4. SLOWER CPU

The CPU ACTIVE TIME, ANY CHANNEL BUSY ONLY TIME and WAIT ONLY TIME are measured. An improvement factor is used for the decrease in CPU speed and the new system time is calculated by the equation: NEW SYSTEM TIME=CPU ACTIVE TIME/IMPROVEMENT FACTOR+ANY CHANNEL BUSY ONLY TIME+WAIT ONLY TIME.

Notice that this equation is not the same as for calculating the effect of substituting a faster CPU. In the case of the slower CPU, the assumption is made that the overlap between the CPU and the channels remain constant, instead of decreasing as in the faster CPU case.

5. SLOWER I/O DEVICES

The CPU ACTIVE ONLY TIME, ANY CHANNEL BUSY TIME and WAIT ONLY TIME are measured. An improvement factor for the decrease in device speed is used for calculating the new system time: NEW SYSTEM TIME=CPU ACTIVE ONLY TIME+ANY CHANNEL BUSY TIME/IMPROVEMENT FACTOR+CPU WAIT ONLY TIME.

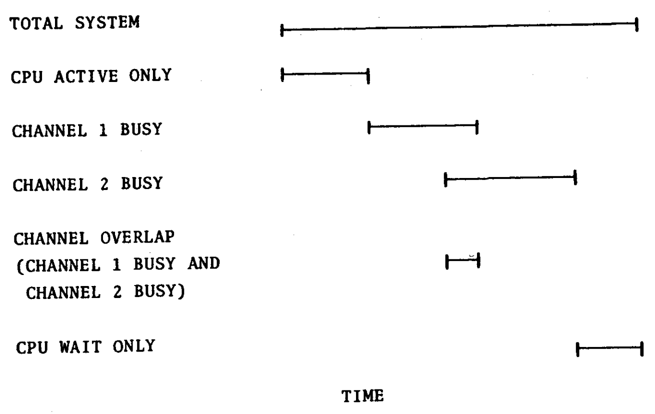


Figure 13—System profile with channels measured separately

This equation is not the same as for substituting faster I/O devices. For slower I/O devices, the overlap between the CPU and channels is assumed to remain constant instead of decreasing as in the faster I/O device case.

6. ALL WORK ON ONE CHANNEL

The CPU ACTIVE ONLY TIME, CHANNEL 1 BUSY TIME, CHANNEL 2 BUSY TIME, . . ., CHANNEL N BUSY TIME, and CPU WAIT ONLY TIME are measured. The system profile is shown in Figure 13. The equation for the new system time using only one channel is: NEW SYSTEM TIME=CPU ACTIVE ONLY TIME+sum of CHANNEL BUSY TIMES+ CPU WAIT ONLY TIME.

7. CHANNEL BALANCING

This calculation is composed of two parts:

Part 1: Measure CPU ACTIVE TIME, DEVICE DATA BUSY TIMES, CHANNEL BUSY TIMES and CPU WAIT ONLY TIME.

Part 2: Examine the DEVICE DATA BUSY TIMES and specify a new device allocation on the channels so that a better balance of the channels is achieved. Calculate a new ANY CHANNEL BUSY ONLY TIME. The new system time is given by: NEW SYSTEM TIME=CPU ACTIVE TIME+ NEW ANY CHANNEL BUSY ONLY TIME+CPU WAIT ONLY TIME.

An example of channel balancing is given in the next section, Example 3.

8. SEVERAL CHANGES IN A SINGLE RUN

The new system time is equal to the sum of the nonoverlapped times + the largest values of the overlapped times. It is necessary to very carefully consider the overlapped areas and determine which area is least affected by the speed change. Using the system profile shown in Figure 14, what is the effect of increasing the speed of the CPU by 2, I/O devices by 1.5, and seek times on the new devices by 2.5?

Each area of the system profile is examined to determine which values to use for the new system time.

AREA 1 The CPU ACTIVE ONLY TIME becomes CPU ACTIVE ONLY TIME/2.

AREA 2 Since the CPU ACTIVE TIME will be decreased more than the ANY CHANNEL BUSY TIME, AREA 2 is changed to CPU ACTIVE TIME $\wedge$ ANY CHANNEL BUSY TIME/1.5.

AREA 3 The new SEEK ONLY TIME is SEEK ONLY TIME/2.5.

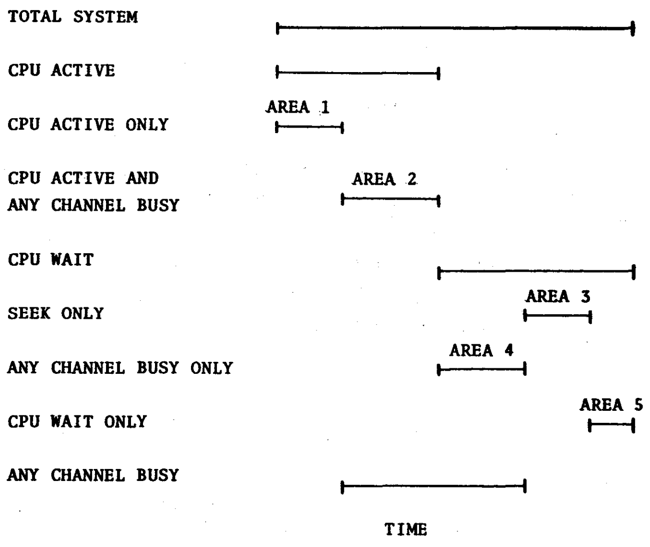


Figure 14—System profile for multiple system changes

AREA 4 The ANY CHANNEL BUSY ONLY TIME becomes ANY CHANNEL BUSY ONLY TIME/1.5.

AREA 5 The CPU WAIT ONLY TIME is unchanged.

Thus,

$$\text{NEW SYSTEM TIME} = \frac{\text{CPU ACTIVE ONLY TIME}}{2} + \frac{\text{CPU ACTIVE TIME} \wedge \text{ANY CHANNEL BUSY TIME}}{1.5} + \frac{\text{SEEK ONLY TIME}}{2.5} + \frac{\text{ANY CHANNEL BUSY ONLY TIME}}{1.5} + \text{CPU WAIT ONLY TIME.}$$

EXAMPLES

All examples will use the same configuration and be run for the same period of time (3600 seconds).

Configuration

1—CPU
2—CHANNELS
4—DIRECT ACCESS DEVICES ON EACH CHANNEL
(D1, D2, D3, D4) on channel 1
(D5, D6, D7, D8) on channel 2

EXAMPLE 1

Putting all of the channel work on one channel

COUNTER	DESCRIPTION	SECONDS	PERCENT
C0	CPU ACTIVE	1200.00	33.33
C1	CHANNEL 1 BUSY	900.00	25.00
C2	CHANNEL 2 BUSY	60.00	1.66
C3	ANY CHANNEL BUSY	945.00	26.25
C4	ANY CHANNEL BUSY$\wedge$WAIT	630.00	17.50
C16	ELAPSED TIME	3600.00	100.00
C17	C0-(C3-C4) CPU ONLY	885.36	24.59
C18	C16-C0-C4 WAIT ONLY	1769.64	49.15

The NEW ANY CHANNEL BUSY is the sum of the channel activity

C19	C1+C2 NEW ANY CHANNEL BUSY	960.00	

The NEW SYSTEM TIME is CPU ONLY+ANY CHANNEL BUSY+WAIT ONLY

C20	C17+C19+C18	3615.00	
C21	C20-C16 SYSTEM SLOW DOWN	15.00	.41

The above equation shows that there would be an increase in running time of 15 seconds by putting all work on one channel. The 15 seconds is exactly the amount of overlap that occurred when the work was on both channels.

EXAMPLE 2

Substituting a CPU that is twice as fast as the old CPU

COUNTER	DESCRIPTION	SECONDS	PERCENT
C0	CPU ACTIVE	1200.00	33.33
C1	CHANNEL 1 BUSY	900.00	25.00
C2	CHANNEL 2 BUSY	60.00	1.66
C3	ANY CHANNEL BUSY	945.00	26.25
C4	ANY CHANNEL BUSY$\wedge$WAIT	630.00	17.50
C16	ELAPSED TIME	3600.00	100.00
C17	C0-(C3-C4) CPU ONLY	885.36	24.59
C18	C16-C0-C4 WAIT ONLY	1769.64	49.15

The new system time is CPU ONLY/IMPROVEMENT FACTOR+ANY CHANNEL BUSY+WAIT ONLY

C19	C17/2+C3+C18	3157.32	
C20	C16-C19 SYSTEM IMPROVEMENT TIME	442.68	12.29

The above equation shows that there will be a decrease in running time of 442.68 seconds.

COUNTER	DESCRIPTION	SECONDS	PERCENT
C21	C16/C19 RST	1.14	
C22	(C0-C17)/2 NEW CPU$\wedge$CHANNEL OVERLAP	157.32	4.98
C23	C19-C0/2 NEW WAIT TIME	2557.32	80.99
C24	C0/2 NEW CPU ACTIVE	600.00	19.00

EXAMPLE 3

Balancing Channels

COUNTER	DESCRIPTION	SECONDS	PERCENT
C0	CPU ACTIVE	1200.00	33.33
C1	CHANNEL 1 BUSY	900.00	25.00
C2	CHANNEL 2 BUSY	60.00	1.66
C3	ANY CHANNEL BUSY	945.00	26.25
C4	ANY CHANNEL$\wedge$WAIT	630.00	17.50
C5	DEVICE 1 DATA BUSY	150.00	4.16
C6	DEVICE 2 DATA BUSY	100.00	2.77
C7	DEVICE 3 DATA BUSY	350.00	9.72
C8	DEVICE 4 DATA BUSY	300.00	8.33
C9	DEVICE 5 DATA BUSY	10.00	.27
C10	DEVICE 6 DATA BUSY	30.00	.83
C11	DEVICE 7 DATA BUSY	15.00	.41
C12	DEVICE 8 DATA BUSY	5.00	.13
C16	ELAPSED TIME	3600.00	100.00

This measurement shows that channels 1 and 2 are not balanced with respect to utilization. An examination of the device utilizations shows that a better device allocation is:

Channel 1 should have devices 2, 3, 5, 7 for a channel utilization of 475 seconds.

Channel 2 should have devices 1, 4, 6, 8 for a channel utilization of 486 seconds.

Next a new system time is computed

COUNTER	DESCRIPTION	SECONDS	PERCENT
C17	C16-C0-C4 WAIT ONLY	1769.64	49.15
C18	C6+C7+C9+C11 NEW CHANNEL 1 BUSY	475.00	13.19
C19	C5+C8+C10+C12 NEW CHANNEL 2 BUSY	486.00	13.74

Since there does not exist an ANY CHANNEL BUSY for the new device arrangement, it will be estimated by using probability theory.

C20	C18$\vee$C19	869.04	24.89

The new ANY CHANNEL BUSY ONLY is assumed to be proportional to the old ANY CHANNEL BUSY ONLY TIME

COUNTER	DESCRIPTION	SECONDS	PERCENT
C21	(C20/C3)*C4	597.64	

The NEW SYSTEM TIME is CPU ACTIVE+ANY CHANNEL BUSY ONLY+WAIT ONLY

C22	C0+C21+C17 NEW SYSTEM TIME	3567.68	
C23	C16-C22 SYSTEM IMPROVEMENT TIME	32.32	.89

The above equation shows that there will be a decrease in system time of 32.32 seconds.

EXAMPLE 4

Calculating a new system time when different types of devices are on the same channel.

The basic configuration will be expanded to include tape as well as disks on channel 2. Then a new system time will be calculated for tapes that are twice as fast. Since tapes and disks exist on the same channel, the measurement should isolate the time that only the tape is operating so that the unit improvement factors can be applied.

The following measurement is made:

COUNTER	DESCRIPTION	SECONDS	PERCENT
C0	CPU ACTIVE	1200.00	33.33
C1	CHANNEL 1 BUSY	900.00	25.00
C2	CHANNEL 2 BUSY	700.00	19.44
C3	ANY CHANNEL BUSY	1424.88	39.58
C4	ANY CHANNEL BUSY$\wedge$WAIT	949.68	26.38
C5	TAPES BUSY$\wedge$DISK ON CHANNEL 2 NOT BUSY	640.00	17.77
C6	TAPES ONLY$\wedge$WAIT	426.24	11.84
C16	ELAPSED TIME	3600.00	100.00
C17	C16-C0-C4 WAIT ONLY TIME	1450.32	40.28

The NEW ANY CHANNEL BUSY ONLY is equal to ANY CHANNEL BUSY ONLY—TAPE ONLY/IMPROVEMENT FACTOR. The NEW SYSTEM TIME is equal to CPU ACTIVE+NEW ANY CHANNEL BUSY ONLY+WAIT ONLY.

C18	C0+C4-C6/2+C17 NEW SYSTEM TIME	3386.88	
C19	C16-C18 SYSTEM IMPROVEMENT TIME	213.12	5.92

Equation 19 shows that substituting a tape twice as fast will reduce the system time by 213.12 seconds.

SUMMARY

The paper has presented a description of hardware monitors, effective methods of optimizing installation throughput and costs, provision for a historical base for predicting future system needs, and significant events to measure. The interpretation of the monitoring results are discussed in detail. Consideration is given to system configuration, programs, routine resident/non-resident, data set allocation, job scheduling and operation methods. Stress is placed on predicting improvements based on actual systems measurements in order to optimize the system with the actual job stream. The performance optimization cycles and the interpretation of the system performance profile were developed.

REFERENCES

1 C T APPLE
The program monitor—A device for program performance measurement
ACM 20th Nat Conf 1965 pp 66-75

2 P CALINGAERT
System performance evaluation: Survey and appraisal
Comm ACM 10 January 1967 pp 12-18

3 G ESTRIN D HOPKINS B COGGAN S D CROCKER
Snuper computer—A computer in instrumentation automation
FJCC 1967 pp 645-656

4 F D SCHULMAN
Hardware measurement device for IBM System/360 time sharing evaluation
Proc ACM Nat Meeting 1967 pp 103-109

5 D J ROEK W C EMERSON
A hardware instrumentation approach to evaluation of a large scale system
ACM Nat Conf 1969 pp 351–367

6 A J BONNER
Using system monitor output to improve performance
IBM Systems Journal 8 1969 pp 290-298

7 *How to find bottlenecks in computer traffic*
Computer Decisions April 1970

4-way parallel processor partition of an atmospheric primitive-equation prediction model

by E. MORENOFF

Ocean Data Systems, Inc.
Rockville, Maryland

and

W. BECKETT, P. G. KESEL, F. J. WINNINGHOFF and P. M. WOLFF

Fleet Numerical Weather Central
Monterey, California

INTRODUCTION

A principal mission of the Fleet Numerical Weather Central is to provide, on an operational basis, numerical meteorological and oceanographic products peculiar to the needs of the Navy. Toward this end the FNWC is also charged with the development and test of numerical techniques applicable to Navy environmental forecasting problems. A recent achievement of this development program has been the design, development, and beginning in September 1970, operational use of the FNWC five-layer, baroclinic, atmospheric prediction model, based on the so-called "primitive-equations," and herein defined as the Primitive Equation Model (PEM).

The PEM was initially written as a single-processor version to be executed in one of the two FNWC computer systems. In this form the PEM was exercised as a research and development tool subject to improvement and revision to enhance the meteorological forecasts being generated.

The development reached a point in early 1970 where the PEM was skillfully simulating the essential three-dimensional, hemispheric distribution of the atmospheric-state parameters (winds, pressure, temperature, moisture, and precipitation). Its ability to predict the generation of new storms, moreover, was particularly encouraging. The FORTRAN coded program, however, required just over three hours to compute a set of 36 hour predictions. To be of operational utility, it was clear that several types of speed-ups were in order.

The principal effort in the development of the operational version of the PEM was directed at partitioning the model to take advantage of all possible computational parallelism to exploit the four powerful central processing units available in the FNWC computer installation. Additional speed-ups involved machine language coding for routines in which the physics were considered firm, and the substitution of table look-up operations for manufacturer supplied algorithms. The resultant four-processor version of the PEM was considered ready for final testing in August 1970, four months after work was initiated.

The one-processor version of the PEM required 184 minutes of elapsed time for the generation of 36-hour prognoses. The four-processor version, on the other hand, requires only one hour of elapsed time to produce the same results.

This paper summarizes the principal factors involved in the successful operation of the 4-processor version of the PEM. Operating System modifications needed to establish 4-way inter-processor communications through Extended Core Storage (ECS) are described in the second section. The PEM structure is described in the third section. The partitions into which the PEM is divided are examined in the fourth section. The fifth section is devoted to the methods employed for synchronizing the execution of the partitions in each of the multiple processors and the model's mode of operation. The results of the PEM development and reduction to operational use are summarized in the last section.

FNWC COMPUTER SYSTEMS COMMUNICATIONS

The Fleet Numerical Weather Central operates two large-scale and two medium-scale computer systems as

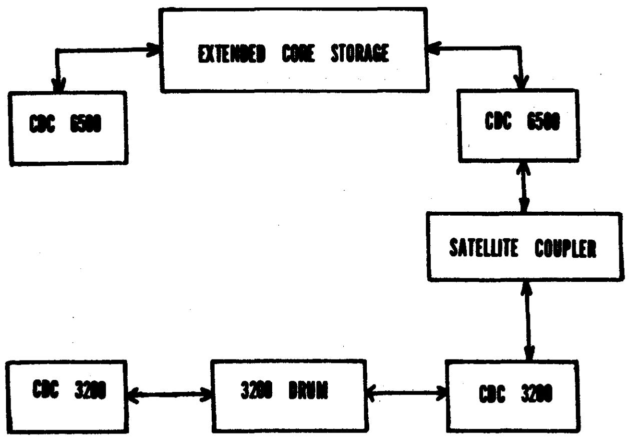


Figure 1—FNWC computer system configuration

shown in Figure 1. The two CDC 2200 computer systems communicate with each other through a random access drum. One of the CDC 3200 computers is linked to one of the CDC 6500 computers by a manufacturer-supplied satellite coupler. The two dual-processor CDC 6500 computer systems are linked with each other through the one million words of Extended Core Storage (ECS).

Normally, the ECS is operated in such a manner that 500,000 words are assigned to each of the two CDC 6500 computer systems with no inter-communication permitted. A mechanism was developed by the FNWC technical staff allowing authorized programs in each of the four central processors of the two CDC computer systems to communicate with each other and, at the same time, be provided with software protection from interference by non-authorized programs.

There are three classifications of ECS access, normal, master and slave, designated for each job in the system by an appropriate ECS access code and a pass key. For normal ECS access these fields are zero. If the ECS access code field designates a job as a master, then the associated pass key will be interpreted as the name of ECS block storage assigned to that job. A slave has no ECS assigned to it but is able to refer the ECS block named by its pass key.

A master job in one of the CDC 6500's may have slave jobs in the other CDC 6500. A communication mechanism called 1SI was established between the operating systems by FNWC technical staff to facilitate implementation of the master-slave ECS access classification. 1SI is a pair of bounce PP routines (one in each machine) which provide a software, full duplex block multi-plexing channel between the machines via ECS. Messages and/or blocks of data may be sent over this channel so that 1SI may be used to call PP programs in the other machine or to pass data such as tables or files between the machines.

Obtaining a master/slave ECS access code is accomplished by two PP programs: ECS and 1EC. A job wishing to establish itself as a master first requests a block of ECS storage in the same manner of a normal access job. Once obtained, the labeling of this block of ECS storage is requested by calling the PP program ECS with the argument specifying the desired pass key and the access code for a master. The program ECS searches the resident control point exchange areas (CPEA) for a master with the same pass key. If one is found the requesting job is aborted even if the program ECS used 1SI to call 1EC in the other machine. 1EC will perform a similar search of the CPEA in its own machine and return its findings to the program ECS via 1SI. If the other machine is down, or if no matching key can be found, the label is established, otherwise the requesting job is aborted. Before returning control to the requesting job, the program ECS increments the ECS parity error flag and monitors via a special monitor function developed at FNWC. A non-zero value of this flag has the effect of preventing ECS storage moves in the half of ECS assigned to the particular machine.

Similarly, a job wishing to establish itself as a slave calls the PP program ECS with the appropriate pass key and access code. ECS searches its own machine's CPEA for a master with a matching key. If none is found, 1EC is called on the other machine via 1SI and the search is repeated in the other CPEA. If still none is found, this fact is indicated to the requesting job. If a match should exist in either machine, the original ECS will have the address (ECRA) and field links (ECFL) of the requesting job saved in its CPEA and will be given the ECRA and ECFL of the matching master.

Modifications made to the ECS storage move program allow ECS storage moves in a machine with no master present. Modifications to the end of job processor reset the ECRA and ECFL of slaves to their values and decrement the ECS parity error flag in the monitor when a master terminates.

ATMOSPHERIC PREDICTION MODEL STRUCTURE

Several developmental variations of a five-layer baroclinic atmospheric prediction model, based on integrations of the so-called primitive equations, were designed and developed by Kesel and Winninghoff[1] in the 1969-1970 period at FNWC Monterey.

The governing equations are written in flux form in a manner similar to Smagorinsky et al.,[2] and Arakawa.[3] The corresponding difference equations are based on the Arakawa technique. This type of scheme precludes nonlinear computational instability by requiring that the flux terms conserve the square of an advected parameter, assuming continuous time derivatives. Total energy is conserved because of requirements placed upon the vertical differencing; specifically, the special form of the hydrostatic equation. Total mass is conserved, when integrated over the entire domain. Linear instability is avoided by meeting the Courant-Friedrichs-Lewy criterion.

The Phillips[4] sigma coordinate system is employed in which pressure, P, is normalized with the underlying terrain pressure, π. At levels where sigma equals 0.9, 0.7, 0.5, 0.3, and 0.1, the horizontal wind components, u and v, the temperature, T, and the height, Z, are carried. The moisture variable, q, is carried at the lowest three of these levels. Vertical velocity, $w \equiv -\dot{\sigma}$, is carried at the layer interfaces, and calculated diagnostically from the continuity equation. See Figure 2.

The Clarke-Berkovsky mountains are used in conjunction with a Kurihara[5] form of the pressure-force terms in the momentum equations to reduce stationary "noise" patterns over high, irregular terrain.

The Richtmyer centered time-differencing method is used with a ten-minute time step, but integrations are recycled every six hours with a Matsuno (Euler backward) step to greatly reduce solution separation. The mesh length of the grid is 381 kilometers at 60 North. The earth is mapped onto a polar stereographic projection for the Northern Hemisphere. In the calculation of map factor and the Coriolis parameter, the sine of the latitude is not permitted to take on values less than that value corresponding to 23 degrees North.

Lateral diffusion is applied at all levels (sparingly) in order to redistribute high frequency components in the mass and motion fields. Surface stress is computed at the lowest layer only.

A considerable part of the heating "package" is fashioned after Mintz and Arakawa,[6] as described by Langlois and Kwok.[7] The albedo is determined as a function of the mean monthly temperature at the earth's surface. A Smagorinsky parameterization of cloudiness is used at one layer (sigma equals 0.7), but based on the relative humidity for the layer between 0.7 and 0.4. Dry convective adjustment precludes hydrostatic instability. Moisture and heat are redistributed in the lowest three layers by use of an Arakawa-Mintz small-scale convection parameterization technique. Small-scale convective precipitation occurs in two of the three types of convection so simulated. Evaporation and large-scale condensation are the main source-sink terms in the moisture conservation equation. Evaporation over land is based on a Bowen ratio, using data from Budyko.

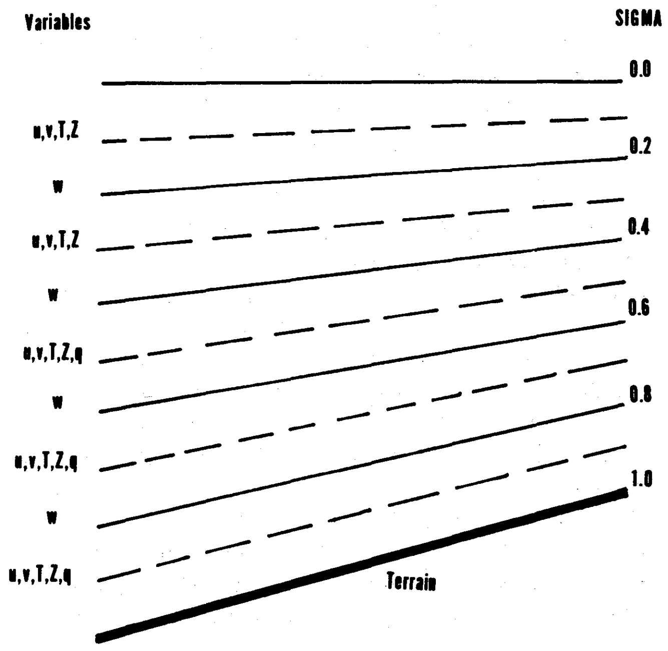


Figure 2—Diagram of levels and variables

In the computation of sensible heat flux over water, the FNWC-produced sea surface temperature distribution is held constant in time. Over land, the required surface temperature is obtained from a heat balance equation. Both long- and short-wave radiative fluxes are computed for two gross layers (sigma = 1.0 to 0.6 and from 0.6 to 0.2). The rates for the upper gross layer are assigned to the upper three computational levels. Those rates for the lower gross layer are assigned to the lower two computational levels.

The type of lateral boundary conditions which led to the over-all best results is a constant-flux *restoration technique* devised by Kesel and Winninghoff, and implemented in January 1970,

The technique was designed to accomplish the following objectives:

a. To eliminate the necessity of altering the initial mass structure of the tropical-subtropical atmosphere as is the case when cyclic continuity is used.
b. To eliminate the problems associated with the imposition of rigid, slippery, insulated-wall boundary conditions; particularly those concerning the false reflection of the computational mode at outflow boundaries.
c. To preserve the perturbation component in the

aforementioned areas in the prognostic period (although no dynamic prediction is attempted south of 4 North the output is much more meteorological than fields which have been fattened as required by cyclic continuity).

The procedure is as follows: All of the distributions of temperature, moisture, wind, and terrain pressure are preserved at initial time. A field of restoration coefficients which vary continuously from unity at and south of 4 North to zero at and north of 17 North is computed. At the end of each ten minute integration step the new values of the state variables are restored back toward their initial values (in the area south of 17 North) according to the amount specified by the field of restoration coefficients. The net effect of this procedure is to produce a fully dynamic forecast north of 17 North, a persistence forecast south of 4 North, and a blend in between. The mathematical-physical effect is that the region acts as an energy sponge for externally (outwardly) propagating inertio-gravity oscillations.

The basic inputs associated with the initialization procedure are the virtual temperature analyses for the Northern Hemisphere at 12 constant pressure levels distributed from 1000 MBS to 50 MBS, height analyses at seven of these pressure levels, moisture analyses at four levels from the surface up to 500 MBS. In addition, the terrain height, sea level pressure and sea surface temperature analyses are used.

Several types of wind initialization have been tried: geostrophic winds (using constant Coriolis parameter); linear balance winds; full balance winds; winds obtained by use of an iterative technique. Aside from geostrophic winds the quickest to compute is the set of non-divergent winds derived from solution of the so called linear balance equation. These are entirely satisfactory for short-range forecasts (up to three days).

The degree of prediction skill currently being observed from the tests is very gratifying. It is clear that little or nothing is known about the initial specification of these parameters over large areas of the Northern Hemisphere, particularly over oceans and at high altitudes.

As noted at the start of the section, the equations are written in flux form and an Arakawa-type conservative differencing scheme is employed. No attempt will be made to exhibit herein a complete set of the corresponding difference equations, since it is well beyond the scope of this paper to do so. Rather, it will suffice to show the main continuous equation forms (using only symbols such as H, Q, and F, to denote all of the diabatic heating effects, moisture source and sink terms, and surface stress, respectively).

There are five prognostic equations, one of which must be integrated prior to parallel integration of the remaining four. These are the continuity equation, the east-west momentum equation, the thermodynamic energy equation, and the moisture conservation equation. Heights (geopotentials) are computed diagnostically from the hydrostatic equation (the scaled vertical equation of motion). Vertical velocities are calculated from a form of the continuity equation. The pressure-force terms are shown in their original forms. [The pressure surfaces are actually synthesized "locally" about each point, by means of the hypsometric conversion of pressure changes to geopotential changes; and geopotential differences are computed on these pressure surfaces.] This Kurihara-type modification tends to reduce inconsistent truncation error when differencing the terrain pressure (which remains fixed in any column) and geopotentials of sigma surfaces (the "smoothness" of which varies with height).

A. East-West Momentum Equation

$$\frac{\partial \pi u}{\partial t} = -m^2\left\{\frac{\partial}{\partial x}\left(\frac{uu\pi}{m}\right)+\frac{\partial}{\partial y}\left(\frac{uv\pi}{m}\right)\right\}+\pi\frac{\partial(wu)}{\partial\sigma}$$

$$+\pi vf - m\left\{\frac{\partial\phi}{\partial x}+RT\frac{\partial\pi}{\partial x}\right\}+K\nabla^2 u\pi+F_x$$

B. North-South Momentum Equation

$$\frac{\partial \pi v}{\partial t} = -m^2\left\{\frac{\partial}{\partial x}\left(\frac{uv\pi}{m}\right)+\frac{\partial}{\partial y}\left(\frac{vv\pi}{m}\right)\right\}+\pi\frac{\partial wv}{\partial\sigma}$$

$$-\pi uf - m\left\{\pi\frac{\partial\phi}{\partial y}+RT\frac{\partial\pi}{\partial y}\right\}+K\nabla^2 v\pi+F_y$$

C. Thermodynamic Energy Equation

$$\frac{\partial \pi T}{\partial t} = -m^2\left\{\frac{\partial}{\partial x}\left(\frac{\pi uT}{m}\right)+\frac{\partial}{\partial y}\left(\frac{\pi vT}{m}\right)\right\}$$

$$+\pi\frac{\partial(wT)}{\partial\sigma}+H\pi+K\nabla^2\pi T$$

$$+\frac{RT}{c_p\sigma}\left\{-w\pi+\sigma\left[\frac{\partial\pi}{\partial t}+m\left(u\frac{\partial\pi}{\partial x}+v\frac{\partial\pi}{\partial y}\right)\right]\right\}$$

D. Moisture Conservation Equation

$$\frac{\partial e\pi}{\partial t} = -m^2\left\{\frac{\partial}{\partial x}\left(\frac{\pi ue}{m}\right)+\frac{\partial}{\partial y}\left(\frac{\pi ve}{m}\right)\right\}+\frac{\pi\partial(we)}{\partial\sigma}+Q\pi$$

where Q = moisture source/sink term

E. Continuity Equation

$$\frac{\partial \pi}{\partial t} = -m^2\left\{\frac{\partial}{\partial x}\left(\frac{u\pi}{m}\right)+\frac{\partial}{\partial y}\left(\frac{v\pi}{m}\right)\right\}+\pi\frac{\partial w}{\partial \sigma}$$

F. Hydrostatic Equation

$$\frac{\partial \phi}{\partial \sigma} = -\frac{RT}{\sigma}$$

PARTITIONING THE MODEL

The PEM may be considered in three distinct sections: the data input and initialization section; the integration section repeated in each forecast time step; and the output section. Each sixth time step, the basic integration section is modified to take into consideration the effects of diabatic heating. This includes incoming solar radiation, outgoing terrestrial radiation, sensible heat exchange at the air-earth interface, and evaporation, Condensation processes, in contrast, are considered every time-step. Each thirty-sixth time step, the results of the preceding forecast hours are output and the integrations reiterated.

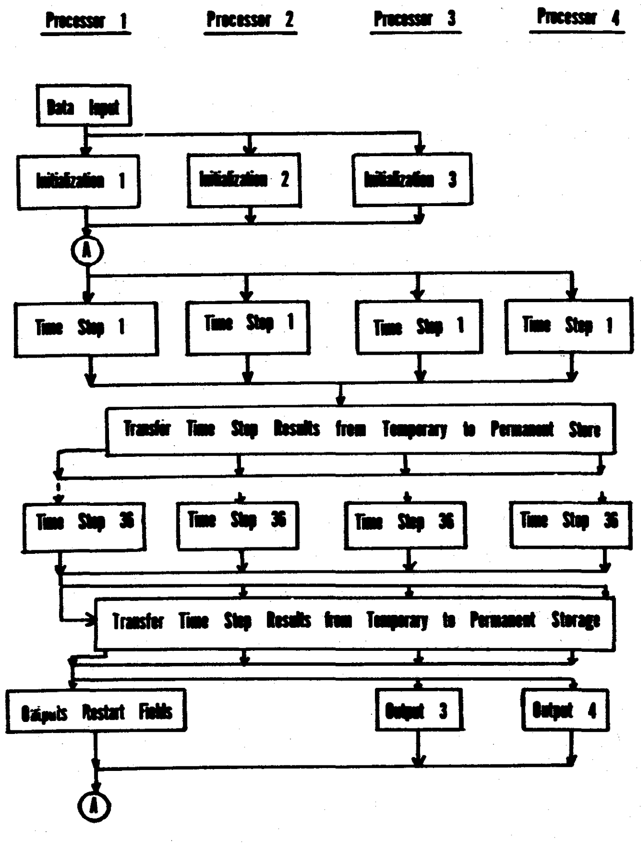


Figure 3—Overall model partition structure

The basic structure of the PEM, as represented by the governing set of difference equations and the method of their solution, is naturally suited for partitioning for parallel operation and concurrent execution in multiple processors. The particular partitioning implemented was selected in order to insure approximately equal elapsed time for the execution of concurrently operating partitions. Four-way partitions were principally employed, although both three-way and two-way partitions were introduced where appropriate.

The basic partition of the model was based on the observation that during each time step in the forecast process the momentum equations in the east-west and north-south directions, the thermodynamic energy equation, and the moisture equation could each be executed concurrently in each of four different processors. By virtue of the centered time-differencing method, the forcing functions to be evaluated in the solution of each of these equations require data generated during the preceding time step and accessed on a read only basis during the current time step. Hence parallel processing could be achieved by providing separate temporary locations for storage of intermediate results during execution of a time step by each processor *and* by providing a mechanism to insure that each processor is at the same time step in the solution of its assigned equation and, where required, at the same level within that time step.

With this four-way partitioning within the basic time step as a starting point, additional possibilities for simultaneity in the model's operation were observed and further partitions developed. For example, prior to the execution of the four-way partitioning within each time step a three-way partition was implemented which allowed the continuity equation to be solved for the interface vertical velocities and the local change of lower boundary pressure at the same time that geopotential-field correction terms are generated. The model's initialization section was similarly partitioned three ways and the output section two ways. Finally, the heating effects computations were implemented as a three-way partition.

The four-way, three-way and two-way partitions were packaged and compiled as four separate programs, one for each of the four FNWC processors. The overall structure of the partitioned model is illustrated in Figure 3. Following completion of the output section at time step (36), the integration sequence is recycled from time step (1) as shown.

Processor 1 is designated as the "master" processor and Processors 2, 3, and 4 as the "slave" processors, both in the sense described in the inter-computer communications section and in the sense that each time step is initiated by command from Processor 1 and termi-

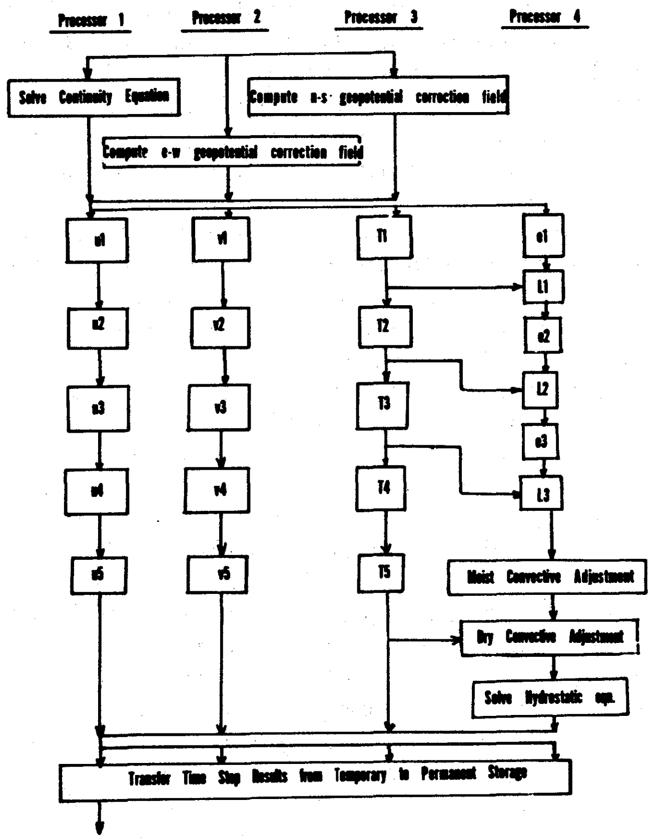

Processor 1
Processor 2
Processor 3
Processor 4
Solve Continuity Equation
Compute n-s geopotential correction field
Compute e-w geopotential correction field
u1
u2
u3
u4
u5
v1
v2
v3
v4
v5
T1
T2
T3
T4
T5
e1
L1
e2
L2
e3
L3
Moist Convective Adjustment
Dry Convective Adjustment
Solve Hydrostatic eq.
Transfer Time Step Results from Temporary to Permanent Storage

Figure 4—Typical time step partition structure

nated by Processor 1 acknowledgment of a "complete" signal emanating from each of Processors 2, 3, and 4. At the completion of each step, results from the computations of that time step are transferred from temporary to permanent locations in storage and the next time step initiated. Once again, the transfer is initiated by command from Processor 1 and terminated by Processor 1 acknowledgment of a transfer complete signal received from Processors 2, 3, and 4.

The structure of a typical time step partition is illustrated in Figure 4. At the start of the time step a three-way split is initiated by Processor 1 during which time Processor 1 integrates the continuity equation to obtain vertical velocities and Processors 2 and 3 compute the ten pressure-force-term geopotential correction fields in the east-west and north-south directions, respectively. At this time Processor 4 is not executing a portion of the model and may either be idling or operating on an independent program in a multi-programmed mode. The completion of the assigned tasks by Processors 2 and 3 are signaled to Processor 1 which then initiates the basic four-way split. The variables u, v, T and e represent the new values of the variables obtained through integration of the east-west and north-south oriented momentum equations, the thermodynamic energy equation and the moisture conservation equation, respectively. The variable L represents the computation of the effects of the large scale condensation process.

Once the computations of the u_i, v_i and T_i ($i=1$, 2, 3, 4, 5) are initiated in Processors 1, 2, and 3, respectively, they proceed independently of one another to the end of the time step. Each "i" value represents another layer in the five-layer atmospheric model.

An added consideration is introduced into the computations of Processor 4, however. Before the effects of the large scale condensation process can be computed for a layer, both the Thermodynamic Energy equation and the Moisture Conservation equation must be solved at that layer. Hence, a level of control is required to synchronize the execution of Processor 4 with Processor 3 within the individual time step computations. Further, the Dry Convective Adjustment computation in Processor 4 requires the completion of all five layers of the Thermodynamic Energy equation before it can be initiated so that a second level of intra-step control is required. At the conclusion of the Dry Convective Adjustment computation, the Hydrostatic equation is integrated in Processor 4 to obtain the new geopotential fields. The time step is concluded with the transfer of intermediate time step results from temporary to permanent storage.

The basic time-step partition structure is modified each sixth time step to include the effects of a diabatic heating. The heating section was implemented as a three-way partition illustrated in Figure 5. Additional intra-step level control is required to synchronize the execution of each of the partitions as shown in the figure. Note that the heating partition in Processor 3 is itself divided to allow as great a degree of simultaneity as possible with the execution of partitions in Processors 1 and 2.

The output section, executed each thirty-sixth time step (at the completion of six forecast hours), is partitioned as shown in Figure 6. The output section partitions were placed in Processors 3 and 4 principally for central memory space considerations, more central memory being available in these processors than in Processors 1 and 2. The basic function of each output partition is co-ordinate transformation of the forecast variables and conversion to forms suitable for the user community.

Each output partition is initiated by command from Processor 1. Processor 4 may immediately begin processing of the east-west and north-south momentum equation variables but must wait on the transformation of the Phi fields until Processor 3 has completed the

Preprocessor program. A three-way partition was not implemented since the Preprocessor must be completed prior to the transformation of the Thermodynamic energy equation and moisture conservation equation variables.

To increase total system reliability a checkpoint restart procedure was designed and coded. At each output step (6, 12, 18, ..., 72 hours) all of those data fields required to restart the PEM are duplicated from their permanent ECS locations onto a magnetic tape by Processor 1, at the same time that Processors 3 and 4 are processing the output forecast fields. The essential difference between these two data sets is that the restart fields contain the variables on sigma surfaces as opposed to the pressure surface distributions required by the consumers.

The "restart" procedure itself requires less than a minute. If the prediction model run is terminated for any type of failure (hardware, software, electric power, bad input data, etc.), the restart capability ensures that the real time loss will be less than ten minutes.

In addition to the four processor version of the Atmospheric Prediction Model a two-processor version was also implemented. The primary motivation for the second implementation was to provide a back-up capability with graceful degradation which could be operated in the event one or two of the central processing units were down for extended periods. The two-processor version will also be used as the vehicle for further research and development efforts to improve the meteorological and numerical aspects of the model, and the quality (skill) of the resultant forecasts.

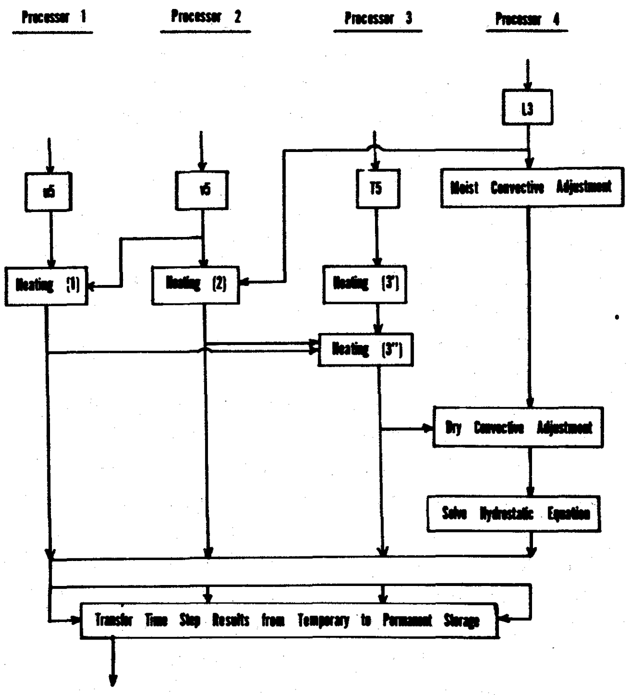

Processor 1
Processor 2
Processor 3
Processor 4
L3
u5
v5
T5
Moist Convective Adjustment
Heating (1)
Heating (2)
Heating (3')
Heating (3")
Dry Convective Adjustment
Solve Hydrostatic Equation
Transfer Time Step Results from Temporary to Permanent Storage

Figure 5—Influence of heating on time step computation

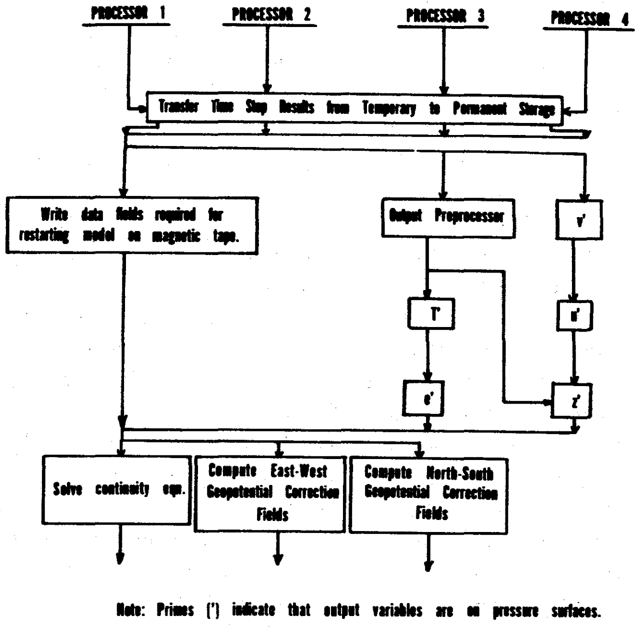

PROCESSOR 1
PROCESSOR 2
PROCESSOR 3
PROCESSOR 4
Transfer Time Step Results from Temporary to Permanent Storage
Write data fields required for restarting model on magnetic tape.
Output Preprocessor
V'
T'
U'
e'
Z'
Solve continuity eqn.
Compute East-West Geopotential Correction Fields
Compute North-South Geopotential Correction Fields
Note: Primes (') indicate that output variables are on pressure surfaces.

Figure 6—Output partition structure

PARTITION SYNCHRONIZATION AND EXECUTION

The parallel execution of the multiple partitions is realizable because it is possible to postulate a mechanism by which the operation of each partition in each of the multiple processors can be exactly synchronized. This mechanism is an adaptation to the requirements of the PEM and the characteristics of the FNWC computer installation of a general program linkage mechanism known as the Buffer File Mode of Operation.[8,9,10]

Implicit in the Buffer File Mode of Operation is the concentration of all inter-program communications through Buffer Files. A Buffer File is a set of fixed length blocks organized in a ring structure and placed in each data path from one program to another. The program generating the data to be passed places the data into the Buffer File once its operations on that data have been completed. The program to receive the data finds the data to be operated on in the Buffer File.

The flow of data through the Buffer File is unidirectional; that is, one program may only write data to the Buffer File and the other may only read data from the

Buffer File. Pointers are maintained which indicate which blocks in the Buffer File have last been written into and read from by the two programs involved in the data transfer. The Buffer File Mode of Operation can be used to synchronize the operation of otherwise asynchronously operating programs in the same or different processors by either of two methods.

In the first instance, program synchronization is effected by regulating the streaming of data through the Buffer File from one program to another. The program writing data to a Buffer File cannot proceed beyond the point in its execution when it is necessary to place data into the Buffer File and there is no room for additional data in the Buffer File. Similarly, a program reading data from the Buffer File cannot proceed beyond the point in its execution when it requires data from the Buffer File and there is no additional data in the Buffer File. The execution of a program, either waiting for additional data in its input Buffer File or for additional space in its output Buffer File, is temporarily delayed, and thereby brought into synchronization with the execution of the other program.

In the second instance, program synchronization is effected by conveying "change of state" or "condition" information from one program to the other. The Buffer File block size is chosen on the basis of the quantity of information to be passed between programs. The internal state change of a program is noted as a block of data in that program's output Buffer File. The fact that there has been a change in state of the program can readily be sensed by the other program which then can read the block of data from the Buffer File. The second program can determine the nature of the change in state of the first program by examination of the data in the block it has read from the Buffer File.

The bi-directional transfer of the program state information is realized by the introduction of Buffer File pairs. The first Buffer File can only be read from by the first program and written to by the second program, while the second can only be read from by the second program and written to by the first program. This method of exchanging state information between programs not only provides a mechanism for synchronizing the execution of two otherwise asynchronously executed programs, but also eliminates the internal program housekeeping which would normally be needed to coordinate the accesses and the sequences of such accesses of the programs to the program state information.

The PEM synchronization mechanism, referred to herein as the Partition Synchronization Mechanism (PSM), is based on the latter alternative. The application of the PSM to the multi-processor FNWC computer environment requires the Buffer Files to reside in some random access storage device jointly accessible by each of the processors. The device which satisfies this requirement is the ECS, operated in the manner previously described.

A pair of Buffer Files is assigned between each two partitions for which bi-directional transfer of state information is required. Hence in the typical time step partition structure illustrated in Figure 4 and amplified in Figure 5, Buffer File pairs are assigned between partitions resident in Processors 1 and 2, 1 and 3, 1 and 4, 2 and 4, and 3 and 4.

The nature of the change of state information to be passed between any pair of partitions in the PEM is whether or not one partition has reached a point in its execution where sufficient data has been developed to allow the other partition to initiate or continue its own execution. This can be represented as a single "GO-NO GO" flag to be sensed by the second partition. Hence, in the PEM the Buffer File recirculating ring structure reduced to a simple single one word block maintained in ECS.

Referring to Figure 3, it can be seen that the issuance of a "GO-NO GO" signal by a partition is equivalent to either a command to "split" the straight line execution of the model into multiple partitions or to "join" the execution of the multiple partitions into a lesser number of partitions. A five character Buffer File naming convention was established to facilitate identification of which process was involved.

The first two characters of the name serve to identify whether the Buffer File is associated with an inter-step or inter-level signal; the former is designated by the characters "IS" and the latter by the characters "IL". The third character specifies whether a split ("S") or a join ("J") is being signaled. The fourth and fifth characters specify the Processors in which the partitions writing and reading the Buffer File are located respectively. Hence Buffer File ISS12 is used by the partition resident to Processor 1 to split its operation by initiating execution in Processor 2 in going from one time step to another.

When the PEM is to be executed the four programs of which it is comprised are loaded, one into each of the four Processors. The programs in Processors 2, 3 and 4 are immediately halted upon initiation and manually delayed until the program in Processor 1, the master Processor, has been assigned the necessary ECS for the model's execution and has initialized all Buffer Files to reflect a NO GO condition. Processors 2, 3 and 4 are then permitted to enter a programmed loop in which each periodically tests a Buffer File to determine when it may initiate processing of its first partition.

While in this programmed loop the slave Processors may either be engaged in the execution of unrelated programs or simply remain in a local counting loop.

Upon completion of the data input phase of its operation, Processor 1 removes the hold on the execution of Processors 2 and 3 which then proceed with the initialization phase while Processor 4 remains at the hold condition. At the completion of its portion of the initialization phase, Processor 1 holds until receipt of a GO signal from Processors 2 and 3, signifying the completion of their assigned partitions. Processors 2 and 3 again enter a hold status after providing the GO signal to Processor 1. Finally, Processor 1 initiates the iterative integration section by signaling the GO condition for Processors 2, 3 and 4. At the completion of the execution of the partitions in Processors 2, 3 and 4 the master Processor is notified via the appropriate Buffer Files and each once more enters the hold condition and remains there until Processor 1, having verified that each partition has been completed, signals the transfer of the time step results from temporary to permanent storage. This process then continues to repeat itself, modified as previously described in each sixth and thirty-sixth time step.

Inter-level holds and go's are generally implemented in the same manner as the inter-step holds and go's described in the preceding paragraph. There is one exception, however. In the partition executed in Processor 4 in the iterative integration section, a separate Buffer File is provided to control the initiation of the execution of the large scale condensation effects computation at each of levels 1, 2 and 3. The separate Buffer file at each level is predicated on the need to allow the partition in Processor 3 to proceed on with its execution after signaling the start of execution of Processor 4 at each level *without* waiting for an acknowledgment of completion of that level by Processor 4.

This emphasizes a particularly important aspect of the operation of the PEM. The execution of the partitions in the different processors cannot get out of synchronization with one another. Each is always working on the same time step at the same time. If the partition in one of the Processors is delayed, for example, while that Processor solves a higher priority problem, then all the Processors at the completion of the processing of their partitions will hold until the delayed Processor "catches-up." The execution of the partitions will not fall out of synchronization.

CONCLUSIONS

The Atmospheric Prediction Model developed at FNWC was partitioned to be operated in a 4-Processor and a 2-Processor configuration, in addition to the 1-Processor configuration for which it was initially designed. The 4-Processor version is currently in operational use at FNWC while the 2-Processor version provides a back-up capability in the event of equipment malfunction and a new research and development tool.

A Partition Synchronization Mechanism was developed for purposes of synchronizing the execution of the partitions being executed in each of the multiple processors. The nature of PSM is such as to insure that each partition is always operating on data in the same time step. The ability to guarantee this synchronization implies it is possible to allow other independent jobs to co-exist and share what computer resources are available with the Partitioned Atmospheric Prediction Model.

The PSM fully utilizes modifications to the operating systems of each of the two CDC 6500 dual processor computers to allow programs in each of the four processors to communicate with each other using ECS. In addition to the intercomputer communications the FNWC operating system modifications insure software protection from interference by non-authorized programs.

As a consequence of employing the 4-Processor version of the Atmospheric Prediction Model, the same meteorological products were generated in 60 minutes rather than the 184 minutes required of the 1-Processor version. This reduction in time allowed the incorporation of a new and more powerful output section and the extension of the basic forecast period from 36 hours to 72 hours. The 72 hour forecast is produced in an elapsed time of 2 hours.

The next step in the evolution of the FNWC PEM involves expanding grid size from 63×63 points to 89×89 points. To accommodate the additional central memory and processing requirements required of such a shift in grid size, partitioning of the horizontal domain rather than the computational burden is under consideration. It is estimated that partitioning the horizontal domain will reduce overall central memory requirements by one-half and allow the 72 hour forecast on the expanded grid to be performed in only four hours as opposed to the five and one-third hours required by the current partitioning method. The results of these new efforts will be reported on in a later paper.

REFERENCES

1 P G KESEL F J WINNINGHOFF
Development of a multi-processor primitive equation atmospheric prediction model

Fleet Numerical Weather Central Monterey California
Unpublished manuscript 1970
2 J SMAGORINSKY S MANAGE
L L HOLLOWAY JR
Numerical results from a 9-level general circulation model of the atmosphere
Monthly Weather Review Vol 93 No 12 pp 727-768 1965
3 A ARAKAWA
Computational design for long term numerical integration of the equations of fluid motion: Two dimensional incompressible flow
Journal of Computer Physics Vol 1 pp 119-143 1966
4 N A PHILLIPS
A coordinate system having some special advantages for numerical forecasting
Journal of Meteorology Vol 14 1957
5 Y KURIHARA
Note on finite difference expression for the hydrostatic relation and pressure gradiant force
Monthly Weather Review Vol 96 No 9 1968
6 A ARAKAWA A KATAYAMA Y MINTZ
Numerical simulation of the general circulation of the atmosphere
Proceedings of WMO/IUGG Symposium of NWP Tokyo 1968
7 W E LANGLOIS H C W KWOK
Description of the Mintz-Arakawa numerical general circulation model
UCLA Dept of Meteorology Technical Report No 3 1969
8 E MORENOFF J B McLEAN
Job linkages and program strings
Rome Air Development Center Technical Report TR-66-71 1966
9 E MORENOFF J B McLEAN
Inter-program communications, program string structures and buffer files
Proceedings of the AFIPS Spring Joint Computer Conference Thompson Books pp 175-183 1967
10 E MORENOFF
The table driven augmented programming environment: A general purpose user-oriented program for extending the capabilities of operating systems
Rome Air Development Center Technical Report TR-69-108 1969

An associative processor for air traffic control

by KENNETH JAMES THURBER

Honeywell Systems and Research Center
St. Paul, Minnesota

INTRODUCTION

In recent years associative memories have been receiving an increasing amount of attention.[1–3] At the same time multiprocessor and parallel processing systems have been under study to solve very large problems.[4–5] An associative processor is one form of a parallel processor that seems able to provide a cost effective solution to many problems such as the air traffic control (ATC) problem.

In general, an associative processor (AP) consists of an associative memory (AM) with arithmetic capability on a per word basis. Usually, the arithmetic logic is a serial adder and the associative processor can thus perform arithmetic operations on the data stored in it on a bit serial basis in parallel over all words.

The two main types of associative processors are a distributed-logic type and bit-slice type (non-distributed logic). The most significant difference in the two types is that the distributed-logic associative processor has logic at every bit position, while the bit-slice associative processor has logic only on a per-word basis. The differences in features of these two approaches are summarized in Table I.

The distributed-logic associative processor has significant speed advantages for the equality search and read/write operations since these operations are performed simultaneously over all bits of every word. On the other hand the bit-slice processor may have a speed advantage for processing operations because it will usually be able to perform bit-slice read and write operations faster than the distributed-logic processor. Thus for a specific problem, the faster of the two approaches will depend on the mix of operations required.

The design of an associative processor that combines the best features of the above approaches and can be applied effectively to problems such as air traffic control is given in this paper. This system has the flexi-

TABLE I—Summary of the method of operation of distributed and bit-slice associative processors

Operations	Distributed Logic	Bit-Slice
Equality Search	Parallel-By-Bit	Serial-By-Bit
Other Search Operations	Serial-By-Bit	Serial-By-Bit
Arithmetic Operations	Serial-By-Bit	Serial-By-Bit
Word Write	Parallel-By-Bit	Serial-By-Bit
Word Read	Parallel-By-Bit	Serial-By-Bit

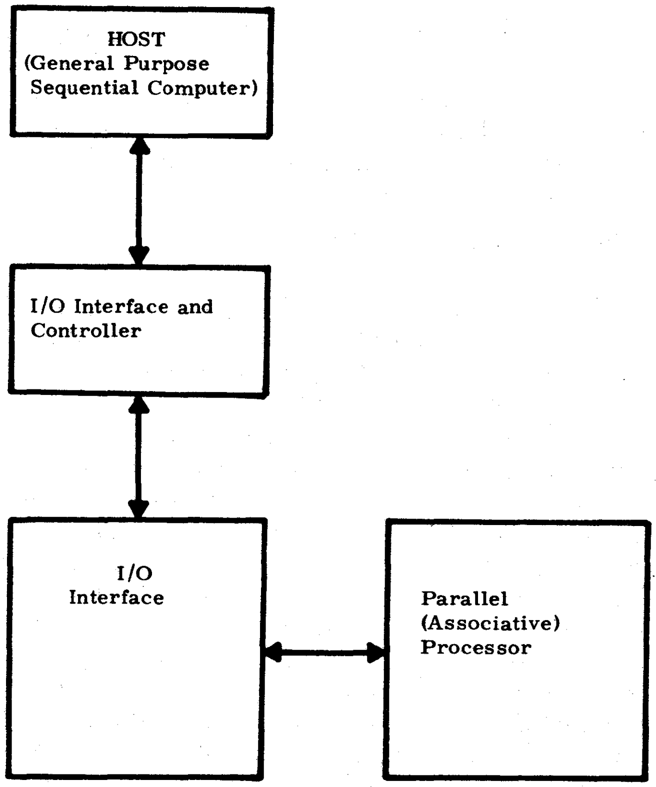


Figure 1(a)—Block diagram of the overall computing system

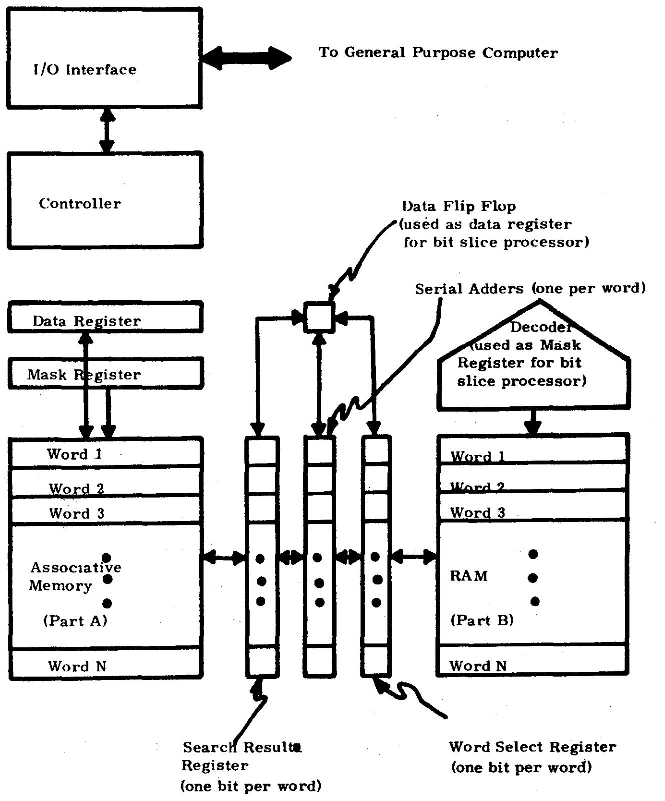

I/O Interface
To General Purpose Computer
Controller
Data Flip Flop (used as data register for bit slice processor)
Serial Adders (one per word)
Data Register
Mask Register
Decoder (used as Mask Register for bit slice processor)
Word 1
Word 2
Word 3
Associative Memory (Part A)
Word N
RAM (Part B)
Search Results Register (one bit per word)
Word Select Register (one bit per word)

Figure 1(b)—Block diagram of the associative processing system

bility to solve the problems that associative processors can solve and do it in a more effective manner than any other processor using the same operation speeds.

SYSTEM DESCRIPTION

There is a large class of problems to which a parallel processor can be applied. However, even this class of problems requires both types of processing; i.e., sequential and parallel. Figure 1(a) shows a general block diagram of a parallel processing system. (For the purposes of this paper, the parallel processor is an associative processor.) The system consists of a host (sequential computer), a control unit for the associative processor (and interface beween the controller and host), the associative processor, and the interface between the associative processor and its controller. *The interface unit to the associative processor is there because generally the associaitve processor and host are incompatible.* For example, in a bit slice type associative processor I/O is accomplished bit serial, whereas, in the host sequential computer I/O is usually accomplished in word parallel. This represents a basic limitation to the overall system! This paper presents a design of an associative processor that does not have this limitation and which has the interface unit built into the system as an integral part of the associative processor.

The overall system is shown in the block diagram in Figure 1(b). The system consists of the following parts:

1. A hybrid associative processor (AP)
2. A microprogrammed controller, and
3. The input output interface.

The input output interface is designed to interface the processor with the host computer. The interface contains registers and gating (such as shown in Figure 2) that perform the following functions: voltage level translations, acceptance of a word from the host processor, routing the word to its appropriate destination (controller or associative processor), and acceptance of a word from the distributed logic portion of the associative processor and transmission of desired

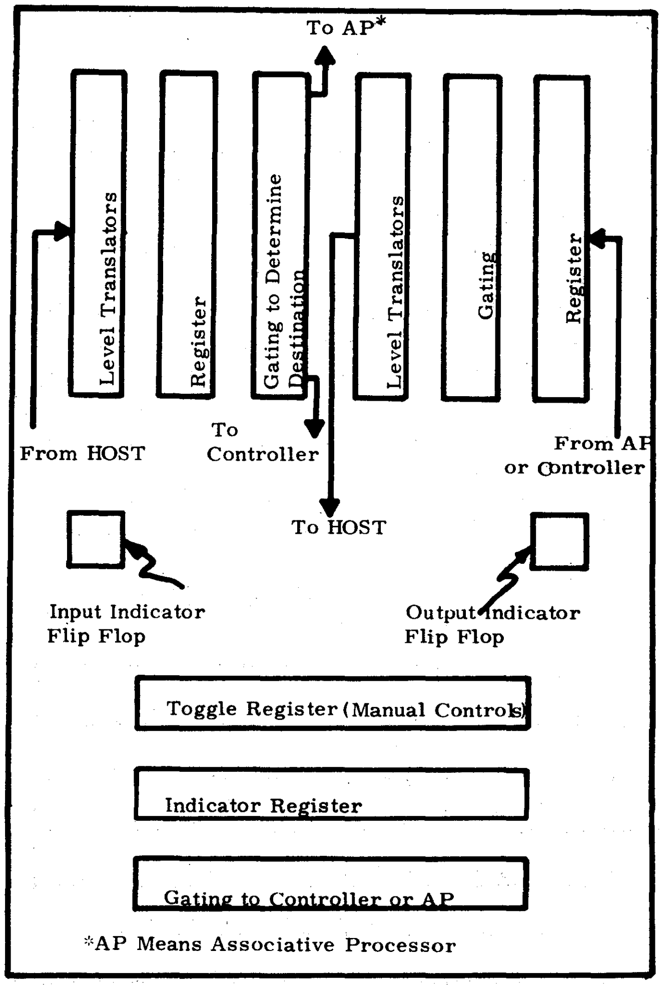

To AP*
Level Translators
Register
Gating to Determine Destination
Level Translators
Gating
Register
From HOST
To Controller
To HOST
From AP or Controller
Input Indicator Flip Flop
Output Indicator Flip Flop
Toggle Register (Manual Controls)
Indicator Register
Gating to Controller or AP
*AP Means Associative Processor

Figure 2—Block Diagram of the I/O Interface

portions of this data to the host processor, a host word at a time.

The microprogrammed controller (Figure 3) accepts instructions from the HOST and then performs the functions called for by the HOST. The controller's memory consists of ROM and RAM. The section of ROM stores the (microinstructions for less-than-search, etc., and the remaining ROM stores constants and other necessary fixed data for the system. The controller also has read/write memory for storing the programs that can be called by the HOST. These programs are written with instructions that are either microinstructions or machine instructions such as equality search, etc. The instructions the HOST sends to the controller activates the programs. This arrangement enables easy design of the software since, the micro-programs, the programs, and HOST/AP interaction software can be written almost independently after they have been defined. The block diagram of the controller is shown in Figure 3.

The associative processor is shown in Figure 4. It consists of two different parts which share the adders and results. registers. One part is a distributed logic associative memory. The memory has the advantage of being able to read and write words in word-parallel form thus eliminating the input/output bottleneck that will occur if only serial-by-bit read/write capabilities are present. The other portion of the processor is a RAM oriented in such a manner that it can do a bit-slice read and write. With the addition of the per-word arithmetic hardware this memory has the fast bit-slice capabilities that we desire for arithmetic operations. This combination gives us the advantages of both

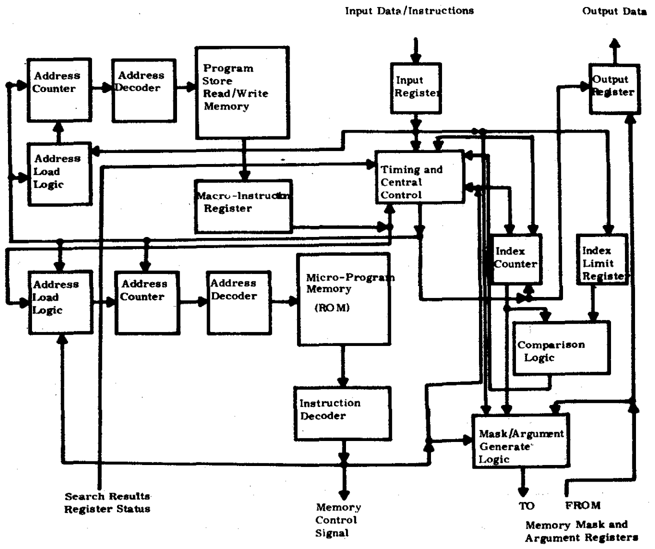


Figure 3—Block Diagram of the Controller for the Associative Processor

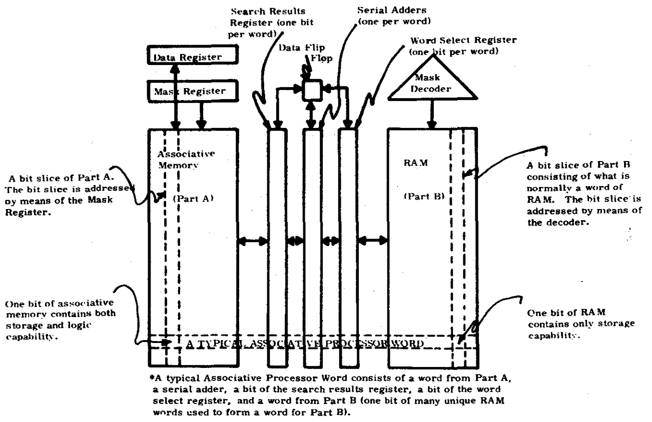

Search Results Register (one bit per word)
Serial Adders (one per word)
Data Register
Data Flip Flop
Word Select Register (one bit per word)
Mask Register
Mask Decoder
Associative Memory (Part A)
RAM (Part B)
A bit slice of Part A. The bit slice is addressed by means of the Mask Register.
A bit slice of Part B consisting of what is normally a word of RAM. The bit slice is addressed by means of the decoder.
One bit of associative memory contains both storage and logic capability.
One bit of RAM contains only storage capability.
A TYPICAL ASSOCIATIVE PROCESSOR WORD
*A typical Associative Processor Word consists of a word from Part A, a serial adder, a bit of the search results register, a bit of the word select register, and a word from Part B (one bit of many unique RAM words used to form a word for Part B).

Figure 4—Block Diagram of the Associative Processor

types of associative processors; i.e., all-parallel equality search and read/write features of the distributed logic approach along with the high speed arithmetic capabilities of a non-distributed logic approach.

The operation of the processor requires that the RAM operate as follows. The RAM can be thought of as being rotated 90° from its normal position. When an address is placed on the input lines to the decoder a "RAM" word is selected, but because of the orientation of the RAM this "RAM" word is a bit slice (a single bit of all data words) to the AP. This bit slice can then be read out into the registers or adders. In addition this bit slice can be gated by the word select register if a subset of words is to be selected. (See the Appendix for a description of an associative memory and its associated registers.) To perform an associative search is very simple. If the bit slice is being compared to a one, it is just read out. If the equality search is on zero, the bit slice is read and every bit complimented. This procedure then yields a 1 in the search results register in every matching bit position. This method allows the bit slice portion of the AP to be implemented using standard off the shelf RAM and conventional IC logic.

PROCESSOR CAPABILITIES

Table II is a comparison of typical associative processor speeds available. The distributed logic system speeds are based upon the Honeywell semiconductor associative memory. A description of the Honeywell associative memory can be found in Reference 1. The bit slice (non-distributed) processor speeds are based upon a bipolar RAM implementation and are what can be achieved with current TTL technology.[6] The bit slice processor uses the decoder as a mask register

TABLE II—Typical operation speeds for the distributed, bit-slice, and hybrid associative processors

	Distributed	Bit Slice	Hybrid	
			Part A (AM)	Part B (RAM)
Bit Slice Read	300ns	100ns	300ns	100ns
Bit Slice Write	200ns	100ns	200ns	100ns
Bit Slice Search	300ns	100ns	300ns	100ns
Parallel Maskable Equality Search	300ns	Not available (100ns/bit)	300ns	Not available (100ns/bit)
Equality Search	300ns	100ns/bit	300ns	100ns/bit
Parallel Word Read	100ns/word	Not available (100ns/bit/word)	100ns/word	Not available (100ns/bit/word)
Parallel Word Write	100ns	Not available (100ns/bit/word)	100ns	Not available 100ns/bit/word)
Add Bit Slice to Bit Slice and Store in a Bit Slice	900ns	400ns	900ns	400ns
Multiple Match Resolve	100ns	100ns	100ns	100ns

and a single flip flop to hold data since it operates in a bit serial fashion. Data will be shifted into the flip flop serially while the decoder address is changed. The speed of the parts is shown in Table II.

The hybrid processor has certain features that can be used to advantage. *The two parts of the system have complimentary properties!*

High speed I/O can be obtained from the hybrid. No data has to be taken from the bit slice part in a bit serial manner. For example if 50 words of 20 bits were to be read from the processor the output time is 5 μs (11 μs) if the words were in the distributed logic portion (bit slice portion). (The extra 6 μs are consumed by reading 20 bit slices from the RAM and storing in the AM portion.) Compare this to a bit slice processor that required 100 μs ($20 \times 50 \times 100ns$) for the same I/O. For most applications this processor has been found to have an I/O rate 10 times that of a bit slice processor and ½ that of a distributed logic processor. In addition, consider the speed of arithmetic multiplication. Multiplication (assume 20 bit operands and 40 bit result) in the distributed logic processor requires about 360 μs ($(20)^2(.9)$ or n^2 bits slice addition operations) compared to 160 μs for a bit slice processor. The worst case in the hybrid processor would be when both 20 bit operands were in the AM and the result was to be stored in the AM. A worst case algorithm would read 40 bits into the RAM, multiply, and store the 40 bits in the AM. This would require 188 μs.

The arithmetic multiply is nearly twice as fast as the distributed logic processor and about the same speed as the bit slice processor.

Arithmetic addition speeds are not significantly enhanced by this processor and are the same as for a bit slice or distributed logic processor depending upon where the operands are stored and the result is to be stored.

Table III is a table summarizing the results of comparing the three processor types. The characteristics of the hybrid processor may be described as faster and more flexible than either of the two standard implementations of associative processors. The values in Table III are for typical operations for problems that have been studied. The hybrid processor combines the best of both standard processing approaches and this can be seen in the table. For most associative processing applications, the hybrid approach should be far superior when compared to either of the other two approaches.

DESCRIPTION OF THE AIR TRAFFIC CONTROL PROBLEM

Three areas of the air traffic control problem will be discussed in this paper. These are tracking, conflict detection, and display processing. For the ATC application the AP size will be 512 words of 104 bits of distributed logic memory and 128 bits of a bit slice type memory. One track will be assigned to each 232 bit word. The controller will require about 2000 words of read/write memory, 500 words of ROM for microprograms, and 2500 words of ROM for system constants.

This processor has been sized to accommodate 512 tracks in the terminal area (64 mile radius). In the terminal area the general purpose computer to which the processor interfaces would probably be the ARTS III (HOST).

Tracking

The tracking function has three main subfunctions that it must perform. These are: correlation of target reports, positional correction of correlated tracks (correction); and positional prediction (prediction) for all tracks.

The correlation function includes the following operations:

1. Obtaining target reports from the HOST.
2. Range and azimuth correlation of target reports against all tracks stored in the associative processor (AP).
3. Tagging the target report for prediction and/or correction.
4. Storing the target report in the track file.

TABLE III—Summary of the computational capabilities of the distributed, bit-slice, and hybrid associative processors

	Distributed Logic	Bit Slice	Hybrid
Equality Searches	10 units/second	1 unit/second	7 units/second
I/O	20 units/second	1 unit/second	10 units/second
Arithmetic	1 unit/second	3 units/second	2-3 units/second
Bit Slice Processing	1 unit/second	3 units/second	2-3 units/second

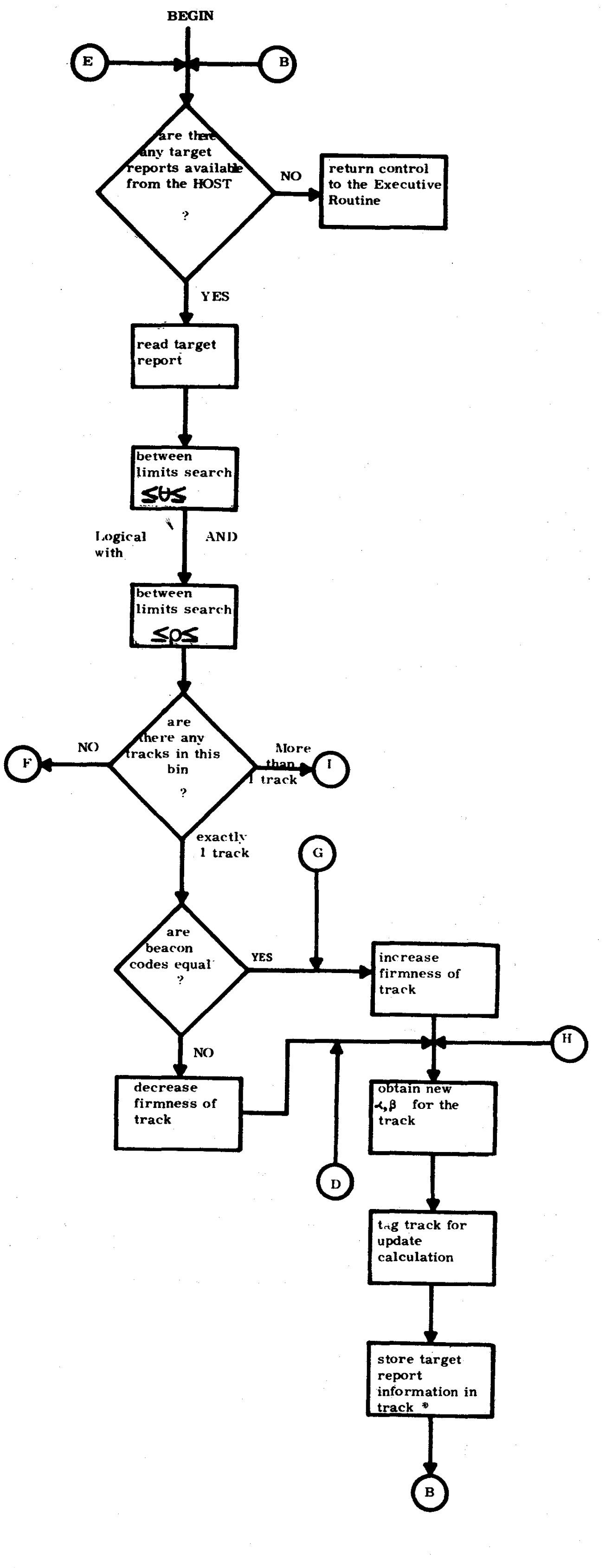

BEGIN
E
B
are there any target reports available from the HOST ?
NO
return control to the Executive Routine
YES
read target report
between limits search ≤θ≤
Logical with
AND
between limits search ≤ρ≤
are there any tracks in this bin ?
NO
F
More than 1 track
I
exactly 1 track
G
are beacon codes equal ?
YES
increase firmness of track
H
NO
decrease firmness of track
obtain new α,β for the track
D
tag track for update calculation
store target report information in track *
B

Figure 5(a)—Path taken by a target report that correlates uniquely with a track in the associative processor (track file)

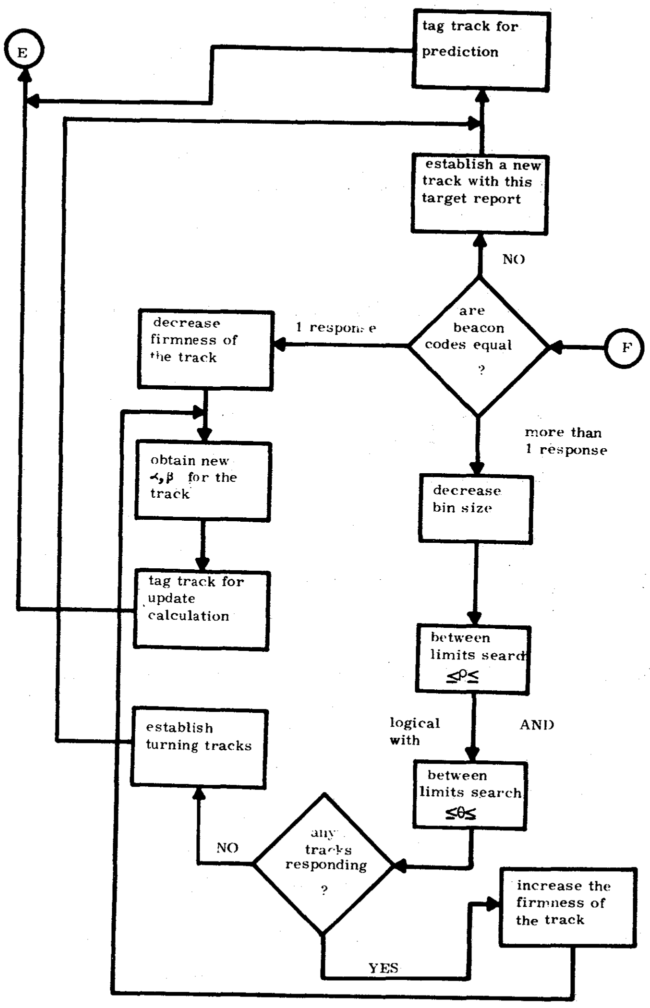

E
tag track for prediction
establish a new track with this target report
NO
are beacon codes equal ?
F
1 response
decrease firmness of the track
more than 1 response
obtain new α,β for the track
decrease bin size
tag track for update calculation
between limits search ≤ρ≤
logical with
AND
establish turning tracks
between limits search ≤θ≤
NO
any tracks responding ?
increase the firmness of the track
YES

Figure 5(b)—Path taken to establish a new track or a turning track

Figure 5 is a flow chart for this portion (correlation) of the air traffic control function. The correlation function will be done as target reports are available from the HOST. The correlation function will be performed once for each target report; i.e., once for each track in the system. Therefore for 256 tracks, the function will be called 256 times every four seconds (one radar scan), etc.

When actually performing the functions on all tracks, the tracks to be corrected will be corrected, and then all tracks will be predicted to their next position.

All tracks correlated during the last ⅛ second will be updated, therefore groups of tracks will be updated eight times per second or thirty-two times per scan (4 seconds for a complete radar scan). To correct the tracks position, four equations must be solved.

These are:

$$(X_c)_n = (X_p)_n + \alpha(X_R - X_p)_n \qquad (1)$$

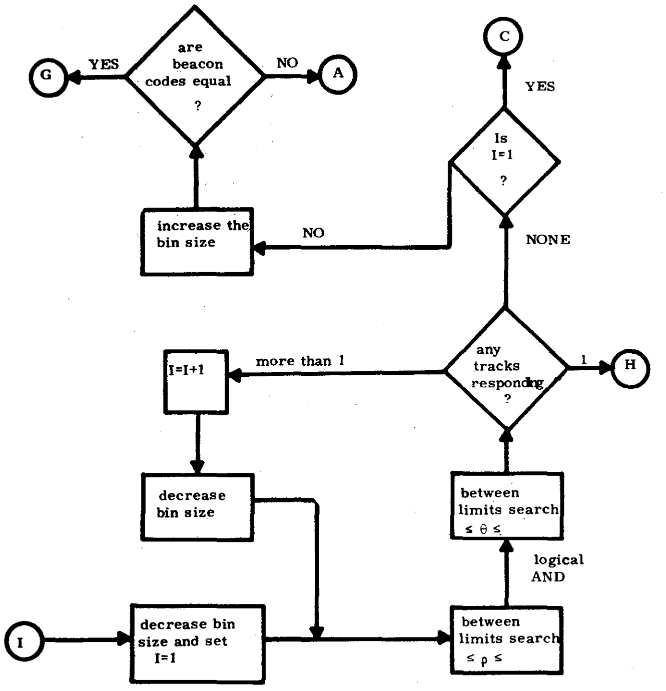


Figure 5(c)—Path taken by a target that correlates with more than one track

$$(Y_c)_n = (Y_p)_n + \alpha(Y_R - Y_p)_n \tag{2}$$

$$(\dot{X}_c)_n = (\dot{X}_c)_{n-1} + \beta/t(X_R - X_p)_n \tag{3}$$

$$(\dot{Y}_c)_n = (\dot{Y}_c)_{n-1} + \beta/t(Y_R - Y_p)_n \tag{4}$$

where X_c means X corrected; X_R means X reported by radar return; X_p means X predicted; and α and β are constants determined by the tracks past history or firmness.

The prediction equations are as follows:

$$(X_p)_n = (X_m)_{n-1} + (\dot{X}_c)_{n-1}T \tag{1}$$

$$(Y_p)_n = (Y_m)_{n-1} + (\dot{Y}_c)_{n-1}T \tag{2}$$

where

$$(X_m)_{n-1} = (X_c)_{n-1} \quad \text{or} \quad (X_p)_{n-1}$$

and

$$(Y_m)_{n-1} = (Y_c)_{n-1} \quad \text{or} \quad (Y_p)_{n-1}$$

The turning track equation is given by the following formula:

$$(X_t)_n = N[V \sin(\theta \pm RN) - \dot{X}_c] + X_p \tag{1}$$

where,

N = time

V = velocity

R = rate of turn

A similar equation can be derived for Y values. After the prediction calculation, the turning tracks will be calculated.

Conflict detection

Figure 6 is a flow chart for a conflict detection scheme. This algorithm uses X, Y oriented rectangles to do gross filtering of the data. The remaining tracks that are potential conflicts are then subjected to a detailed calculation involving the law of cosines to determine if the circular shapes overlap. Any conflicts are then outputted to the HOST for conflict resolution and false alarm checking.

Figure 7(a) shows the ideal conflict detection areas. The circle around the airplane is an area of immediate danger. The larger area is an area of potential future danger. In order to effectively process a conflict algorithm a search is made over rectangular areas surrounding the shapes. This is shown in Figure 7(b). Figure 7(c) shows the basic philosophy behind the conflict equation. It is desired to know if any circles overlap, however, this is a very hard search to accomplish. Refinements and approximations to this criteria designed

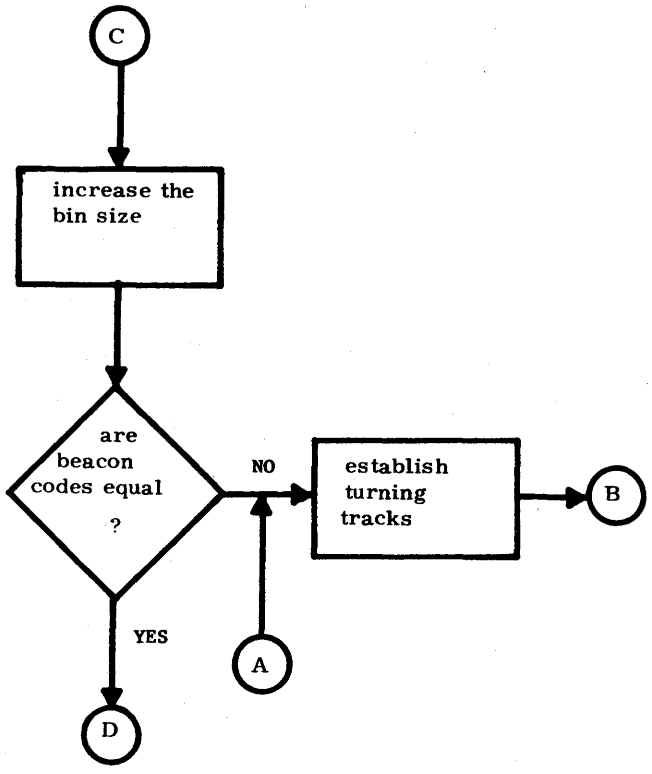
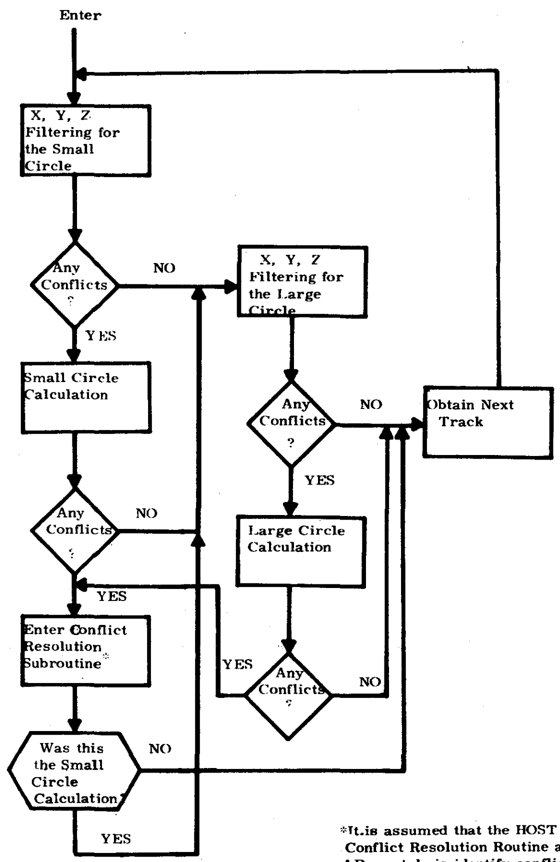


Figure 5(d)—Path that establishes turning tracks for target reports that correlated with more than one track



*It is assumed that the HOST contains a Conflict Resolution Routine and all the AP must do is identify conflicts and pass them to the HOST via this subroutine.

Figure 6—Overall conflict detection algorithm

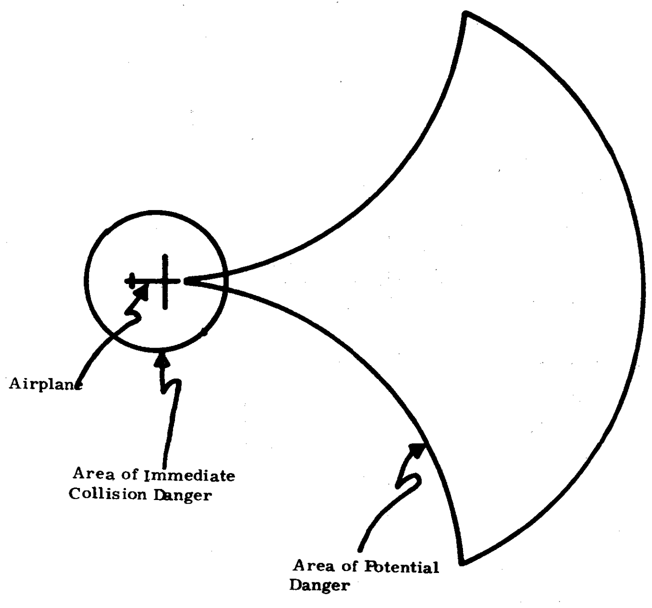


Figure 7(a)—The privileged airspace around an aircraft

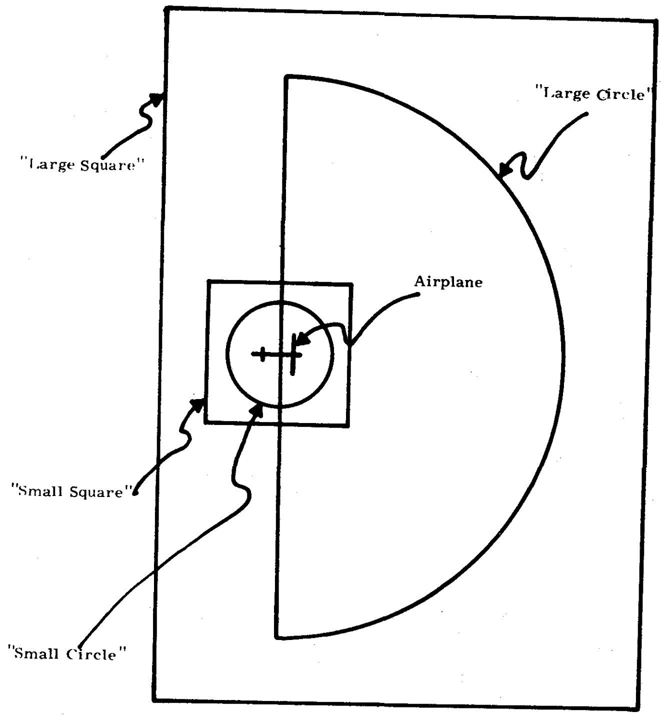


Figure 7(b)—Conflict areas for the conflict detection algorithm

to shape the search areas more like those shown in Figure 7(a) have been considered but are beyond the scope of this paper.

Figure 7(c) shows the equation that is derived from the conflict detection function. This equation is just the Law of Cosines applied to the conflict detection problem to determine if the circular shapes overlap. To avoid a possible conflict the following must be true: $[\rho^2+\rho^2 i-2\rho\rho i\ \cos(\theta-\theta_i)]-[(R+R_i)^2]>0$. This equation must be true for all aircraft being compared to the aircraft being processed.

Display processing

The display processing function will send the display data to the HOST after filtering the data. It is assumed that there is reserved storage in the HOST that contains the detailed filter information for each display (i.e., for each display there is a node of data containing the X, Y, Z limits that the display is controlling) and information to assemble the display data in the HOST refresh memory.

It is assumed that the display data are read out to the HOST, the display data assembled, and the display data entered into the refresh memory. This is done twice a second; i.e., the complete display routine is processed twice a second. The HOST will receive the

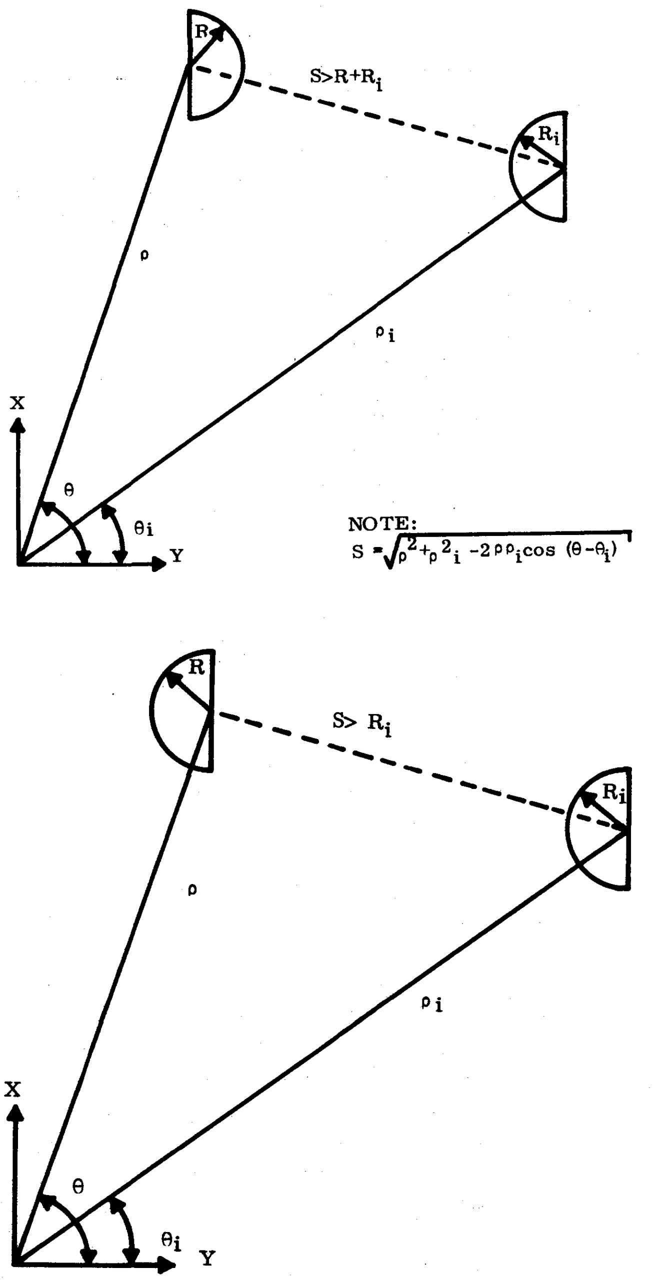

R
S>R+R_i
R_i
ρ
ρi
X
θ
θi
Y
NOTE:
$S = \sqrt{\rho^2 + \rho_i^2 - 2\rho\rho_i \cos(\theta - \theta_i)}$
R
S> R_i
R_i
ρ
ρi
X
θ
θi
Y

Figure 7(c)—Derivation of the conflict detection equation and one of its possible refinements

GPID field of the word, X position, Y position, and altitude and thus can assemble the full and partial data blocks, along with the tabular list information as each track is sent to the appropriate display refresh memory. In order to perform this function the HOST needs a table of information of each track stored in its memory. This function is performed quite fast since it is all searches and reading. All of the detailed display information that does not change very fast is kept in the HOST and can be identified by use of the GPID field that is sent over with the positional information. The algorithm for this function is given in Figure 8.

Definition and allocation of memory fields

The Associative Processor contains the following fields:

- X_c—corrected X position
- Y_c—corrected Y position
- Z—altitude if the plane has a beacon transponder otherwise zero
- BC—Beacon transponder code otherwise zero
- GPID—A code that identifies this track uniquely. Allows the HOST to identify each track
- Firm—the firmness of the track (essentially a measure of the consistency of correlation of the track)
- α—a tracking coefficient for the X positional values derived from a least squares fit tracking algorithm

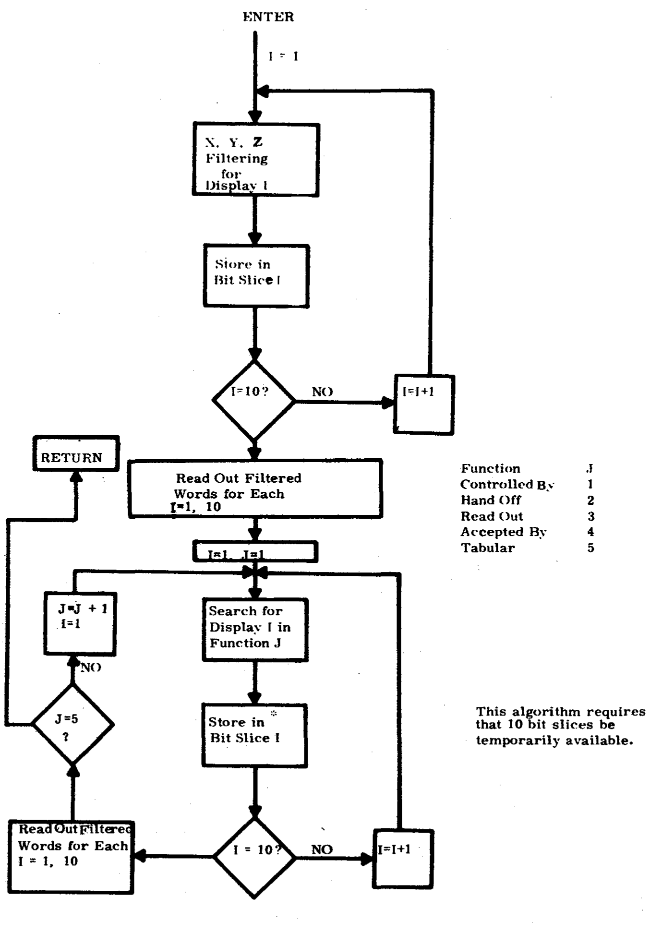

ENTER
I = 1
X, Y, Z Filtering for Display I
Store in Bit Slice I
I=10?
NO
I=I+1
RETURN
Read Out Filtered Words for Each I=1, 10
I=1, J=1
J=J + 1 I=1
NO
J=5 ?
Search for Display I in Function J
Store in Bit Slice I
Read Out Filtered Words for Each I = 1, 10
I = 10?
NO
I=I+1
Function J
Controlled By 1
Hand Off 2
Read Out 3
Accepted By 4
Tabular 5
This algorithm requires that 10 bit slices be temporarily available.

Figure 8—Display filtering algorithm

- β—a tracking coefficient for the Y positional values derived from a least squares fit tracking algorithm
- Temp—temporary storage fields
- CB—controlled by field. The number in this field designates the display that is controlling this track
- AB—accepted by field. The number in this designates the display that has accepted the track if it was being handed off from one display to another.
- HO—hand off to. This field designates the display the track is being handed off to.
- RO—read out by. This field designates the number of the display reading out tracks other than those under its control.
- TAB—Tabular—this field designates the number of the display on whose tabular list this track appears
- Update Flag—Designates tracks that correlated, but have to have their positions corrected and predicted.
- Conflict Detection Flag—designates tracks that need a conflict detection check
- X_p—Predicted X position
- Y_p—Predicted Y position
- θ_p—predicted azimuth
- ρ_p—Predicted ρ position
- X_c—corrected X velocity
- Y_c—corrected Y velocity

Figure 9 shows the manner in which the fields of each word have been allocated. When the system is first initialized, there will be a momentary bottleneck because a lot of data will have to be put into the RAM; however, this bottleneck should be less than a bit slice processor. After this initialization has been accomplished the number of changes in the information in the RAM will be small. The fields were distributed between the AM and the RAM in order to minimize output from the RAM. None of the fields in the RAM portion of the system are read out and sent to the HOST. They are either fields that change slowly and have to be written into the memory from data received from the HOST (CB, AB, HO, RO, TAB) or fields that are calculated and never have to be read out to be sent to the HOST (Flags, X_p, Y_p, θ_p, ρ_p, X_c, Y_c).

In the associative memory, we have the data which require that a quick output capability be available. Data fields that we would like to be able to read out in word parallel, such as X_c, Y_c, Z, BC and GPID, have been included in the AM. Also, data fields that need a word parallel read and write capability, such as Firmness Factor (FIRM), α, and β, have been included in the AM. This organization gives as the best speed solution to the input problem by overcoming the bit serial input output problems of the non-distributed logic approach and the slow bit slice read of the distributed logic approach.

Timing and I/O Data Estimates

The following timing and I/O estimates were made assuming that (1) 512 tracks are contained in the associative processor; (2) 128 tracks must be correlated per second (512 tracks per radar scan); (3) the updating routine is processed eight times a second; (4) the conflict detection algorithm is processed on each track two times per scan (that is 256 conflict detection checks are made every second); (5) all display information is updated twice per second; and (6) a software organization as discussed in the next section is used.

In order to time the conflict detection algorithm it was assumed that the maximum number of responses to the XYZ filtering was 6 for the small square and 45 for the large square. Because of the accuracy required it was decided that the trignometric functions would be done fastest by table look from the ROM in the controller.

Under the above assumptions it is estimated that with the speeds in Table II, the performance of the air traffic control problem for 512 tracks will utilize 50 percent of the processors capability. The 50 percent use includes all overhead and bookkeeping functions. Comparable estimates were made for a bit slice processor and a distributed logic processor. Using the speeds given in Table II, both of these processor require approximately 65 percent of the processor's capabilities. Therefore, the hybrid processor can handle approximately 30 percent more processing than either of the other two processors.

Input to the AP and its controller is estimated at 1200 words per second. Output to the ARTS III is estimated at 30,000 words per second. Therefore an approximate total of 31,000 words of I/O per second are anticipated for worst case operation. In the air traffic control problem *as formulated here*, I/O does not seem to present a major problem.

Software organization

A very simple organization of the AP and interface is assumed. The HOST can transmit only data or one of four instructions to the AP. The instructions are as follows:

- Correlate (number of tracks)
- Update
- Conflict Detection Probe (number of tracks)
- Display

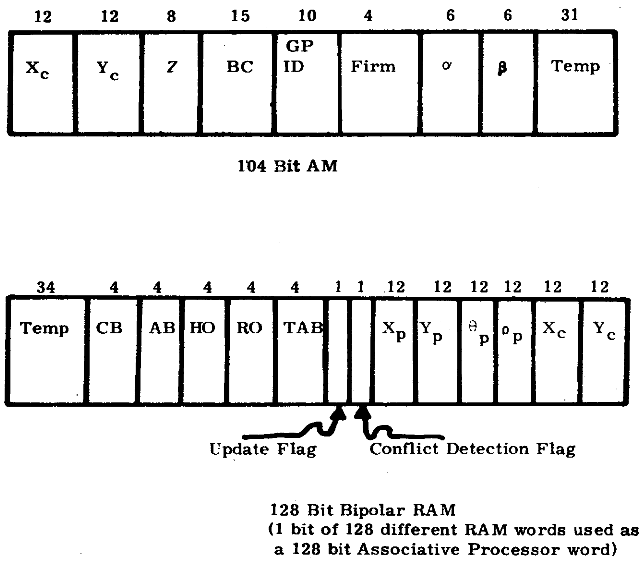


Figure 9—Allocation of fields in the memory word

The AP only transmits the results of the performance of the above functions to the HOST. The programs for the instructions are stored in the 2000 word read/write memory in the AP controller. These programs are written in terms of the AP's macroinstructions (multiply, add, less than search, etc.) which in turn call for the execution of the appropriate microprogram to be executed from the controller's ROM. The microprograms are written in terms of the basic machine instructions such as equality search, bit slice read, word read, etc.

CONCLUSION

A new type of an associative processor has been designed. This processor combines the best properties of the bit slice and distributed logic associative processors. The processor provides the flexibility that will enable it to out perform either of the other two processors on most applications. In typical applications, the processor can handle 30 percent more processing then either of the other two types of processors.

In general, the processor has the I/O and equality search capabilities of a distributed logic associative processor combined with the bit slice and arithmetic processing capabilities of a bit slice processor, thus making it more effective than any other associative processor. This processor overcomes the main drawback of current associative processors, i.e., I/O problems.

The processor was applied to the air traffic problem. It was sized for 512 tracks which corresponds to a 1975 traffic load for most terminal areas with a 64 mile radius. A microprogrammed controller was used to provide future flexibility. The processor was about 50 percent loaded (for 512 tracks) considering overhead functions. This processor can provide a viable solution for the ATC problem.

The air traffic control system used memory speeds that are available from current MOS associative memories and off the shelf bipolar RAM's. The processor is built from a combination of a distributed logic associative memory and a bipolar RAM. The processor has one word per track (104 bits AM and 128 bits RAM per word) or 512 words of memory. Each word has a serial adder plus associated registers. Several tables (ROM) are needed because certain functions will be performed by table look up.

ACKNOWLEDGMENT

The author wishes to thank L. D. Wald and D. C. Gunderson for their assistance and pateienc in helping the author gain an understanding of associative techniques. Thanks are also given L. D. Wald for the details of the control unit for the associative processor. The author wishes to thank the following personnel of the FAA for their help in understanding the air traffic control problem: Lawrence Shoemaker, James Dugan, John Harrocks, and Jack Buck.

REFERENCES

1 L D WALD
MOS associative memories
The Electronic Engineer August 1970 pp 54-56
2 L D WALD
An associative memory using large scale integration
National Aerospace Electronics Conference Dayton Ohio May 1970
3 A G HANLON
Content-addressable and associative memory systems—A survey
IEEETEC Volume EC-15 No 4 1966 pp 509-521
4 J A GITHENS
A fully parallel computer for radar data processing
National Aerospace Electronics Conference Dayton Ohio May 1970
5 J C MURTHA
Parallel processing techniques in avionics
National Aerospace Electronics Conference Dayton Ohio May 1970
6 J W BREMER
A survey of mainframe semiconductor memories
Computer Design May 1970 pp 63-73
7 R E LYONS
The application of associative processing to air traffic control
1er Symposium International Sur La Re'gulation du Trafic, Trafic Ae'rien Versailles June 1970 pp 6A-31 to 6A-40

8 J A RUDOLPH et al
With associative memory, speed is no barrier
Electronics June 22 1970
9 N A BLAKE J C NELSON
A projection of future ATC data processing requirements
Proceedings of the IEEE March 1970

APPENDIX—DESCRIPTION OF AN ASSOCIATIVE MEMORY

An associative memory (AM) is a device that combines logic at each bit position along with storage capacity. A *n* word AM with *p* bits per word can store *n* binary words of *p* bits. In addition, certain logic operations can be performed on the words stored in the AM. In particular, search operations can be performed *simultaneously*, over all words. These operations can identify words in the memory that are related to the externally supplied test word. For this reason AM's are sometimes referred to a content addressable memories (CAM). The types of operations that can be performed are:

- Fully parallel maskable equality search
- Bit serial inequality searches
- Bit serial incrementation of fields
- Bit serial maximum (minimum) search (identifies the maximum or minimum stored word)

A brief example is given to illustrate the use of an associative memory. An eight-word associative memory, with four three-bit fields, is shown in Figure 10. In addition to the memory that stores the words, an AM must have a search register for storage of the word to be compared with the stored words, a mask register to designate which of the bit positions of the search word are to be included in the search operation, a results register for storing the results of the search, and a word select register to select the words to be searched over. For the example, word seven has not been selected as shown by the contents of the word select register in Figure 10. In Figure 10, the contents of the mask register show that only the first field of the search register is to be included in the search. An equality search operation in the above associative memory will result in the simultaneous comparison of the contents of the first field of the search register with the contents of the corresponding field of all stored words. It can be noted that only stored words three and six satisfy the search and are therefore identified by 1's in the results register after the search. Word seven would have satisfied the search; however, it was not in the set of words designated for performance of the search by the word select register.

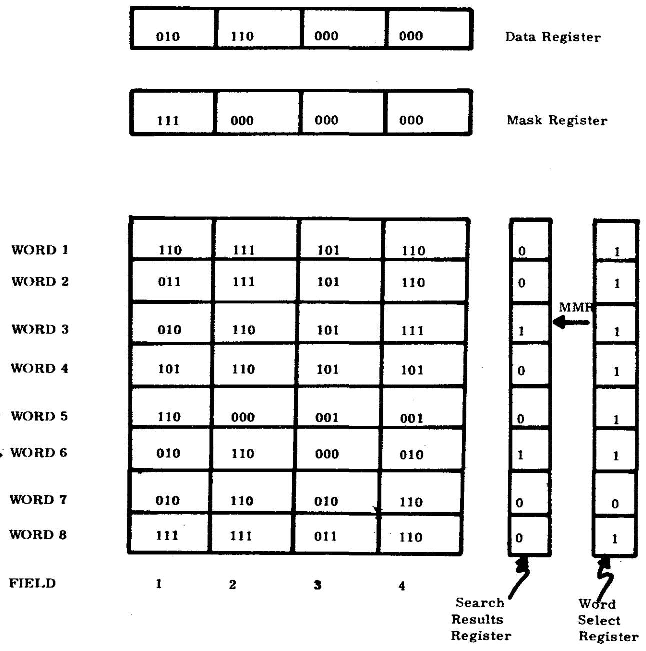


Figure 10—An associative memory

In many associative memory applications, such a search operation would normally be followed by a readout operation (whereby the identified words are sequentially read out) or another search operation (in which case the search results register would be transferred into the word select register). One notes that a series of searches can be performed and the results ANDed together if the results in the search results register are used as new contents of the word select register.

A multiple match resolver (MMR) is also an integral part of the memory. This is indicated by the arrow in Figure 10. The MMR indicates the "first match" in the memory if there were any matches.

A computer aided traffic forecasting technique—The trans-Hudson model

by EUGENE J. LESSIEU

The Port of New York Authority
New York, New York

INTRODUCTION

The transportation problems of the New York Metropolitan region are many and diverse and there are several major governmental agencies concerned with and working towards solutions to these problems. Among these problems is that of planning, providing and maintaining transportation facilities across the Hudson River between the states of New York and New Jersey. Although the trips across the river constitute only a small part of the total regional travel, they amount to over one million trips a day.

Over the past years, the Port Authority has collected and analyzed much data on the volume of traffic crossing the river by all modes. It has also conducted origin and destination surveys to study many of the characteristics of this trans-Hudson traffic. Through the years, the region has grown, the data have become more voluminous, and the analysis more complex. It was becoming more difficult to do comprehensive research and analysis with the data and it was apparent that some sort of formalized information system was necessary to research the trend changes in the volume and the pattern changes in the O and D.

The purpose of traffic research is primarily for forecasting. If one understands the reasons for traffic changes as they occur, then one can more reliably predict future traffic changes based on these reasons. Traffic may shift to a new facility because it makes travel faster. Traffic may grow at one facility and not at another because it is a lower cost facility, or because rapid development is taking place in the market area of one facility and none in the other.

With this in the background, the Port Authority embarked on the development of a system of traffic data handling that would be aimed at researching traffic patterns and their influences, and forecasting traffic based on this research program. Because of the large amount of data available and complex research techniques applicable only to computer solution, the use of a high speed computer as a tool was mandatory.

REVIEW OF AVAILABLE TOOLS

There have been many urban transportation studies in the past decade. Techniques have varied but in most cases new methods are built upon old ones. When the Port Authority decided to embark on the trans-Hudson study, it was natural to review all existing processes. It was discovered that the focus of most other studies was generally to depict and forecast all traffic patterns in an entire region with perhaps a special focus on the Central Business Districts (CBD). With the Port Authority's major focus being on only the Hudson River Crossings, it was decided that much of the theory and many of the techniques applied were inappropriate for our problem.

There are many theories of movement; gravity model, intervening opportunity, etc. Most of these however, are strongest in describing the phenomenon that most trips are short trips—as distances increase between zones less trips occur. In the traffic that crosses the river, there are few short trips. Most of the trips across the river are major trips—not simply going down the street for a quick shopping trip (which by the way makes up a considerable part of the region's total travel). The feeling was that if we were to isolate these major trips from total trips we would have to have additional theories and a different system from that used by others.

With regard to techniques, most of the other studies' end product was a traffic assignment on each link (representing a transportation system segment) of a multi-link network (sometimes thousands). With only the few links that cross the Hudson River of interest to us, many of the efficiencies of the existing techniques would be wasted on our problem.

There was, however, one technique which we considered indispensable and that was the general network tracing and least-path calculating process. There were several programs available that used the Moore's algorithm or some adaptation thereof, that we could count on using.

There was available in house a computer, magnetic tapes containing all the trip data from our O and D surveys, and the rudiments of a computerized data bank. This bank had a data matrix of 180×180 cells with space for 50 pieces of information in each cell. There were programs available to put the O and D data and other data into the bank. There were programs to modify the data once in the bank, and there were programs to extract and manipulate the data so that they could be fed into other standard analytical programs. We decided to use this data bank and modify it to our needs.

Of equal importance to us was our finding that an existing multiple regression program was available that would accept our data in both size and format, had the flexibility to manipulate the data easily and produced printed results sufficient for analysis.

It seemed, then, that we had sufficient tools to put a whole system together and that we could start, get results, and improve the system as we went along. Some interesting comments regarding this assumption are related later in the paper.

DESIGN OF THE RESEARCH AND FORECASTING SYSTEM

Knowing the tools available, the system was designed around them. It was necessary, of course, to review and organize the input data and to specify the output requirements. Further demands on the system were that there had to be a complementary flow of data through the system for research and testing and forecasting, and the system should be designed to provide for continuous use and change as later data become available.

Our output requirements were specified, of course, by the job we set out to do—forecast trips across the Hudson River by facility. Exactly which process to use to get down to the level of facility traffic forecasting was considered in depth. Standard metropolitan transportation studies had usually developed the system by a three stage-process—(1) Trip generation or interchange; (2) Modal split and (3) Traffic assignment. Trip interchange concerns the total number of person trips between zones. Modal split describes the process of determining what share of the total trips will be made by each mode. Traffic assignment is the term used to describe which route or which specific transportation facility will be used once the mode of travel is chosen.

As mentioned earlier, we had collected a great deal of origin-destination data on the various modes of transportation across the river. After a long study of trip data it was decided that, in order to get an explainable group of trips, the trips should be segregated into several sets. First, peak period travel and off peak travel were known to exhibit entirely different characteristics particularly with regard to modal choice, but also with regard to associating travel times and costs to the trips since congestion is greater in the peak. The second separation deemed necessary was a classification by residence, since trips from *A* to *B* would

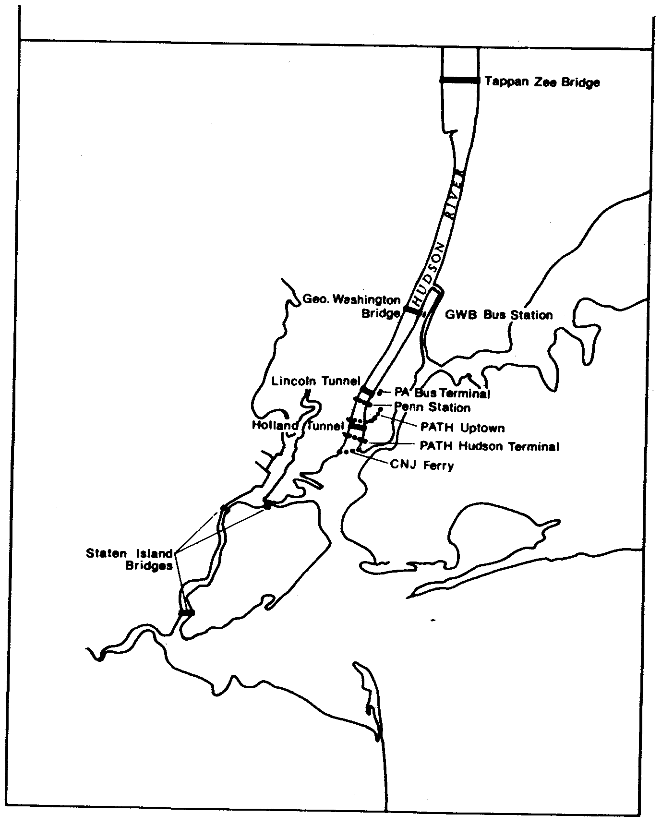


Figure 1—Hudson River crossings

have different modal choice and different trip generating characteristics depending on whether the home based end were *A* or *B*.

To describe mode and facility classification of these trips, a little geography of this region is necessary. The map shows the river crossings available. There were seven vehicular crossings: The Tappan Zee Bridge, George Washington Bridge, Lincoln Tunnel, Holland Tunnel and three Staten Island bridges. There were three rail facilities: PATH downtown (to Hudson Terminal), PATH uptown and the Pennsylvania Rail tunnel. There were two railroad passenger ferries and there were two locations where major flows of interstate buses occurred.

Because of space limitations in the data banks and because analysis of the system revealed that specific definition of some crossings was unnecessary for our future forecasting requirements, it was decided to collapse the crossings into the following mode and facility groups:

Auto mode—	Tappan Zee Bridge
	George Washington Bridge
	Lincoln Tunnel
	Holland Tunnel
	three Staten Island Bridges
Bus mode—	P. A. Bus Terminal (at Lincoln Tunnel)
	George Washington Bridge Bus Station
Rail mode—	Penn Station
	PATH downtown (Hudson Terminal)
	PATH uptown
	CNJ ferry

Traffic using the other ferry (Erie Lackawanna Rail passenger ferry) was included with the PATH downtown traffic because the two crossings were parallel and served an identical market and it was known that the ferry service was soon to be eliminated.

Up to now, we have eleven facilities within three modes and four classifications of trips (Peak, Off-peak, residence east, residence west). In order to get data to explain why trips might be made over one facility or another or one mode versus another, travel network characteristics data had to be collected. The items of data we felt would be important, could be collected, and could be forecasted were travel time, travel cost and number of transfers.

The trip interchange part of the forecasting problem is probably the most difficult in deciding what information is needed to study trip interchange characteristics and attempt to forecast future trip volumes. Many variables can be included in the study part of it in developing relationships that explain differences in trip making. But it must be remembered that only those explanatory variables that themselves can be forecast can be used to explain trips if one wishes to forecast as well as explain. With these restrictions we chose population, employment and area (so that densities could be used) and some description of proximity. It was the latter item that established the basis for the construction of the master program that ties together the entire forecasting system.

The data needed to cover the complete range of studies and models planned had to be placed in a data bank so that it was readily accessible for both developing the models and using them for forecasting. A data bank is nothing more than an arrangement of information stored (on tape) in some meaningful indexed form. In the system developed, the index was geographical zones.

The data bank programs that had already been developed had space for a 180×180 zone classification, but we used only part of this. We classified 100 zones west of the Hudson as "*i*" zones and 80 zones east of the river as "*j*" zones. The reference index then contained 8,000 $i-j$ cells that could be referenced by an $i-j$ number. A map of these zones is shown in Figure 2.

Within each of the cells we had space for 50 different data items. With reference to the earlier description of data it can be seen that we had eleven facilities and four classifications of trips (residence east, residence west, peak and off-peak). We also had time, cost and transfer data for each of the facilities and population and employment, and area data for each of the zones.

Summing these up:

5 auto facilities ×2 network variables	=10
6 transit facilities ×3 network variables	=18
11 facilities×trips for 2 residence classes	=22
4 demographic variables	= 4
space for new facilities in forecast years	=10
	64

It can be seen that the 50 data item spaces of a single bank were easily exceeded, and it was necessary to devise a method to utilize more than one bank. Such a method was developed which in essence, simply keyed to the fact that a single bank was only critical when using it as input to the forecasting system. In the model development stages, separate banks could be used for each of the models—assignment, modal split and trip interchange. A listing of the data in the various banks developed is shown in Figure 3.

Considering the data bank limitations and the fact that peak and off-peak traffic differ in many respects,

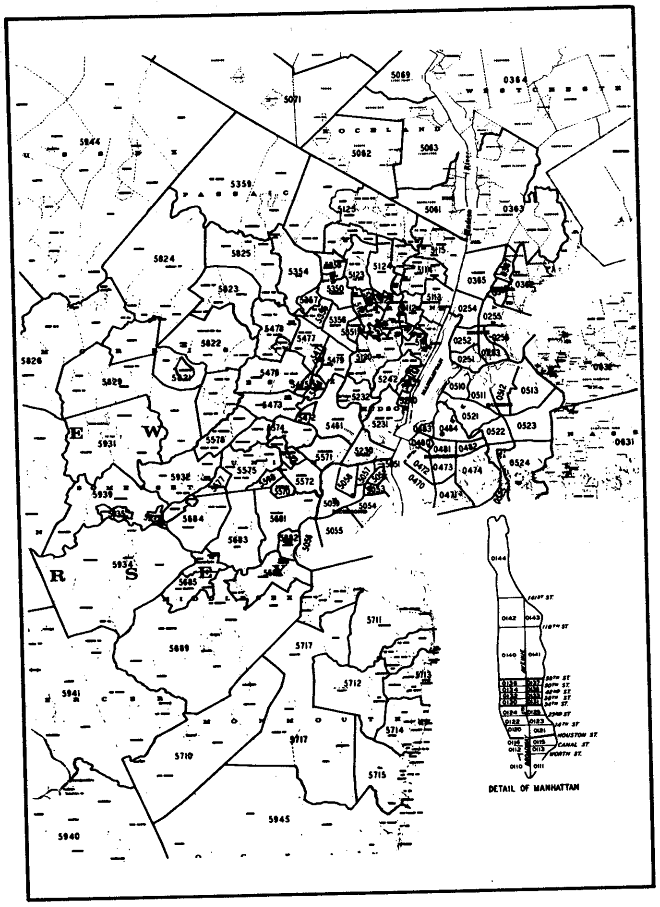


Figure 2—Port Authority analysis zones

it was naturally decided to approach the peak and off-peak as two separate and distinct efforts, and to forecast each time period independently. A set of banks of similar format but with entirely different data is used for the off-peak. The only similarities between the peak and off-peak is the process used and the demographic data.

From inspection of the data bank listing it can be seen that the first Data Bank (1964 I) contains the most fine grained data; facility network data and facility trips. This bank is used for developing the assignment model. A secondary Data Bank (1964 II) was developed (it could have been placed in the same bank except for lack of space) by collapsing the facility network data to mode network data through the concept of weighted average of the facility network data using the existing trips as the weighing factor. The second bank also contains weighted average total network data developed by a similar concept of using the existing modal trips as weighing factors. This bank is used for developing the modal split and trip interchange models and therefore also contains population, employment, area data.

For testing the models a data bank similar to Data Bank I is used since the fine grain facility detail is necessary. Modal total trips are also in this bank (1964 M) so that the assignment and modal split models can be tested on some existing base total. Forecasting is done with a similar Bank (1985 M) which contains estimated future network characteristics and zone populations and employments.

The process is more fully explained under the discussion of the master program.

FLOW OF INFORMATION THROUGH THE SYSTEM

Figure 4 shows how the basic data is gathered and made use of within the system. The source data has been discussed in general earlier.

Item #	1964I	1964II	1964M	1985M
1	At1 (GWB)	AAt	At1	At1
2	At2 (LT)	AAc	At2	At2
3	At3 (HT)	AVE a	At3	At3
4	At4 (SIB)	AVW e	At4	At4
5	At5 (TZB)	ABt	At5	At5
6		ABc		At6 (new)
7	Ac1 (GWB)	AVE a	AC1	AC1
8	Ac2 (LT)	BVW e	Ac2	AC2
9	Ac3 (HT)	ARt	Ac3	Ac3
10	Ac4 (SIB)	ARc	Ac4	Ac4
11	Ac5 (TZB)	RVE	Ac5	Ac5
12	AV1 (GWB)	RVW		Ac6 (new)
13	AV2 (LT)	Pi	Pi	Pi
14	AV3 (HT)	Ei	Ei	Ei
15	AV4 (SIB)	Area i	Area i	Area j
16	AV5 (TZB)		Pi	Pi
17	Bt1 (GWB)	Pj	Bt1	Bt1
18	Bt2 (PABT)	Ej	Bt2	Bt2
19	Bc1 (GWB)	Area j	Bc1	Bc1
20	Bc2 (PABT)		Bc2	Bc2
21	BV1 (GWB)	PC	Ej	Ej
22	BV2 (PABT)	ABF	Area j	Area j
23	Rt1 (P. Sta)	ARF	Rt1	Rt1
24	Rt2 (HT)	.	Rt2	Rt2
25				Rt3 (new)
26	Rt4 (PUP)		Rt4	Rt4
27			PC	PC
28	Rt6 (CNJ)		Rt6	
29	Rc1 (P. Sta)		Rc1	Rc1
30	Rc2 (HT)		Rc2	Rc2
31				Rc3 (new)
32	Rc4 (PUP)		Rc4	Rc4
33				
34	Rc6 (CNJ)		Rc6	
35	RV1 (P. Sta)		AVEa	reserved
36	RV2 (HT)		AVWe	for
37			BVEa	output
38	RV4 (PUP)		BVWe	↓
39			RVEa	
40	RV6 (CNJ)		RVWe	
41				
42				
43	BF1 (GWB)		BF1	BF1
44	BF2 (PABT)		BF2	BF2
45	RF1 (P. Sta)		RF1	RF1
46	RF2 (HT)		RF2	RF2
47				RF3 (new)
48	RF4 (PUP)		RF4	RF4
49				
50	RF6 (CNJ)		RF6	

A = auto	t = travel time	
B = bus	c - travel cost	PC = parking cost
R = rail	V = trip volume	i = zones west of river
AA = avg. auto	F = transfers	j = zones east of river
AB = avg. bus	P = population	Ea = residence east
AR = avg. rail	E = employment	We = residence west

Figure 3—Data bank formats

The determination of the proper values to be placed in the data bank merits some attention. Auto trip data were taken from the continuous sample origin-destination surveys taken at the Port Authority's facilities and the Tappan Zee Bridge. Bus trip data were based on origin-destination surveys taken at the two bus terminals. Rail trip data, including the PATH system were synthesized from a PATH origin-destination survey, origin-destination surveys of those rail lines involved in the Aldene Plan (Central Railroad of New Jersey; Pennsylvania Railroad—Shore Branch), from various railroad conductor counts, and from the Manhattan Journey-to-Work Surveys taken in 1961-1962.

For auto times and costs, it was necessary to build peak and off-peak link and node networks. Travel time for each facility was calculated along the minimum time path with all of the other trans-Hudson facilities removed from the system. Travel distances were found by skimming over those paths. Costs were based on over-the-road costs of 2.8¢ per passenger mile, plus tolls and average parking costs. The 2.8¢ figure was developed by an independent study and was based on out-of-pocket vehicle costs divided by average vehicle occupancy.

The bus and rail time, cost, and transfer matrices were developed by adding rows and columns for what might be called a common point network. Travel times, were determined from each zone west of the Hudson to a Manhattan terminal (Penn Station for example). Then travel times were determined from that terminal to each zone east of the Hudson. The same was done for costs and transfers. This depicted, quite naturally, how a bus or rail trip is made, and it was necessary only to add the rows and columns to determine the full $i-j$ matrix of all time, cost, and transfer data.

Population, employment and area data were extracted and updated from Federal and State Census Data.

It is interesting from an information handling aspect, to describe some of the trials and tribulations of manipulating the various pieces of input data from their original form to a finally completed data bank.

First, a guiding decision was made in the development stages that all attempts would be made to use "in house" computer services. The Port Authority had for internal use both an IBM 7070 and an IBM 360-40. A data bank had already been built for some similar work using the 7070. We had also done some earlier work on the auto networks using the Control Data Corporation's Tran-Plan Programs and some further work using the Bureau of Public Roads Transportation Planning programs run on an IBM 7094.

To make a long story short, we decided to utilize as many of the existing programs as possible to build

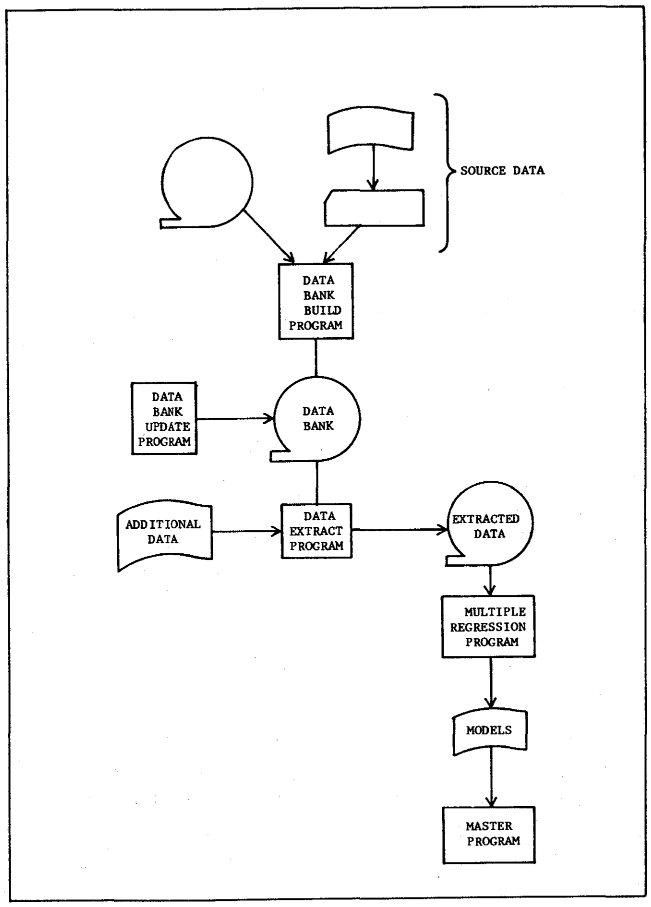


Figure 4—Model development process

the data bank. The auto network data was run on the CDC 3600 and converted on the 3600 to input compatible for the 7070 programs. The bus and rail network data was constructed from punch cards on the IBM 7094 with the BPR program and converted to the input format with that same CDC 3600 program. The trip data was transferred to the desired input format from its original state by an IBM 360 program and converted to proper input format for the data bank, with the IBM 7070. The population, employment and area data was directly input to the bank from punched cards.

The data extract program was written originally for the 7070 and we used it in the early stages to extract data compatible for input to a known regression program run on an IBM 7094.

In the course of the work all new programs were written for use on the IBM 360. Further, all the programs originally written for the 7070 were rewritten for the IBM 360 and made more flexible in the process. We also hunted up regression programs and altered them for our needs, so that regressions too could be run on the IBM 360. During the progress of the work, the Port Authority in-house computer was changed to an IBM 360-75 and all programs were modified where necessary to operate on it.

As is the case with all information handling systems, errors can be expected in processing data from one stage to another. It is necessary then to have some means of correcting the errors. A flexible data bank update program was developed to accomplish this. Since the amount of data in the bank is so large, the program not only provides for correcting individual pieces of data, but also provides for massive changes with a few simple instructions. If, for example, it was discovered that the bus travel time on the west side of the river was five minutes too short from one zone it would mean that travel time from that zone to all zones on the other side would be wrong. Correcting this can be done with three punched cards. This update program is also used extensively to create forecast year network data and to change this data to describe many different alternate transportation systems for study.

Once the data is in the data banks and verified as correct, it is necessary to extract this data, in certain pre-determined groups and in certain formats in order to perform the regressions to develop the models. For each of the models, the data is extracted in a different form.

For the assignment model, we used a rating system for each of the facilities and a rating had to be calculated and placed on the extract tape along with the time, cost and transfer difference for each of the facilities. The modal split being done in two stages required two separate extracts. Further, a classification system was used in the Bus vs. Rail modal split and each class required a separate data extract. The trip interchange also used a classification system that required many separate extracts to properly group the necessary data.

Since multiple regression programs vary greatly as to their capacity and flexibility, it is necessary to formulate the data in the extract stage so that it can be easily used by the regression program. We have used a number of regression programs, none of which we can get to work on our own computers with the flexibility we would like. Both the extracting of data and the preparation of the instruction to the regression program are tedious jobs. They require rigorous attention to detail. Ratios cannot be calculated by dividing by zero; natural logs cannot be taken of negative numbers, etc.

Model development also is not a simple undertaking. If a hypothesis is stated such that trips$=k+a$ population$+b$ employment$+c$ travel time, and the regression shows that this is not a linear relationship then other forms of the variables might be tried such as logs, ratios, powers, etc. If this shows a poor fit then data might be reclassified so that different groups will be included. If this proves negative, then new variables have to be sought to try to explain the variation in trips. The latter approach also presents problems for, as explained earlier, every explanatory variable used for forecasting must be forecastable itself. This means that each search for a new variable must be thorough to the point of satisfying this condition.

A brief description of the models that have been developed to date and the theory behind their development is contained in the following section.

Assignment models

The assignment technique employed is based on the concept that each crossing facility within a mode of travel competes with all others for the trips to be made within that mode between each origin-destination pair. While it is true that there is also competition between modes, as well as between facilities within a mode, considerable literature is in existence that indicates there are different factors that govern choice of mode. These factors might not easily be handled in an allocation method that does not specifically identify the mode. The technique considered for allocation within mode does not necessarily identify the facility, *per se*, in its concept.

The assignment model is based on a rating system first introduced by Cherniack. The concept assumes that the traveler compares the travel time, travel cost, and, in the case of bus and rail, the numbers of transfers for the alternatives he can choose from. In evaluating the alternatives, the traveler perceives the fastest facility and compares that time to the times of the other facilities; he perceives the least expensive facility and compares that cost to the costs of the other facilities; he perceives the most convenient alternative and compares it to the others; or, more realistically, he perceives some combination of all factors. He then rates the alternate facilities and gives the highest rating to the one that he perceives to have the best combination of time, cost, and convenience and a lesser rating to those he perceives to not have these advantages. Conversely, if the use of each facility is based on the cumulative rating of all users, then each facility could be given a rating based on its traffic volume compared with the traffic volume of all other competing facilities. The facility with the highest volume gets the highest rating and others, comparatively lower ratings.

Using multiple regression techniques the relationship between these three factors and the comparative usage of the facilities was explored for each mode. The function considered can be expressed as follows:

$R1 = T1/TH = f\ (t1 - ts,\ c1 - cc,\ F1 - Ff)$ where,
$R1$ = rating of facility 1; the ratio of trips via facility 1, $T1$, to trips via facility most heavily used, TH.
$t1$ = door to door travel time via facility 1,
ts = door to door travel time via the fastest facility
$c1$ = travel cost via facility 1,
cc = travel cost via the least expensive facility,
Ff = number of transfers via the facility with the fewest transfers,
$F1$ = number of transfers via facility 1

The R value or rating will equal 1.0 if the facility in question is the most heavily used and will be less than 1.0 for all lesser used facilities. Also, the differences will equal zero if the facility in question is the best for the particular transportation variable. The ratings and the differences (to be known as Δt, Δc, ΔF, for time, cost, and transfer differences, respectively)

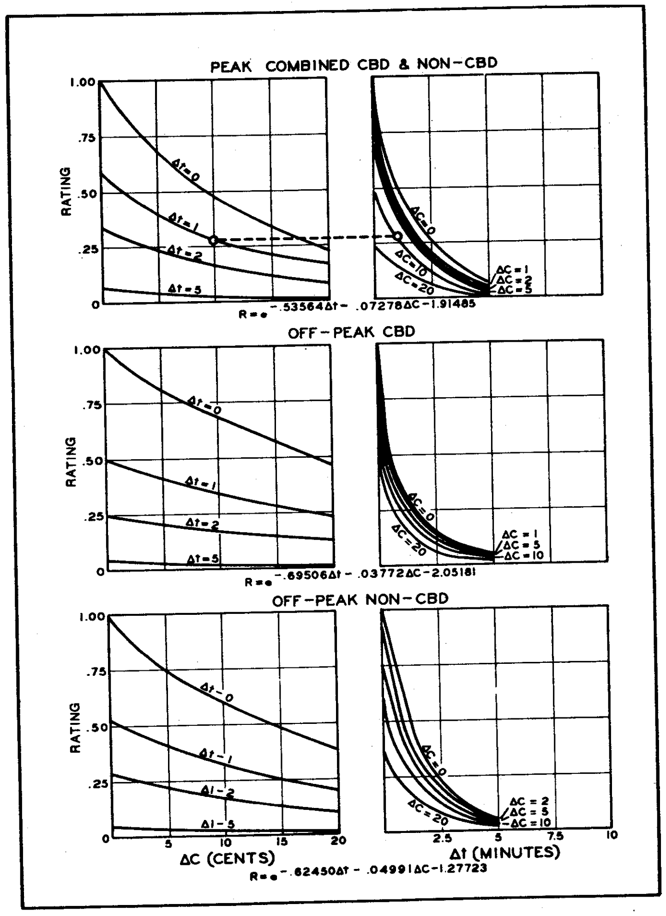


Figure 5—Auto assignment models

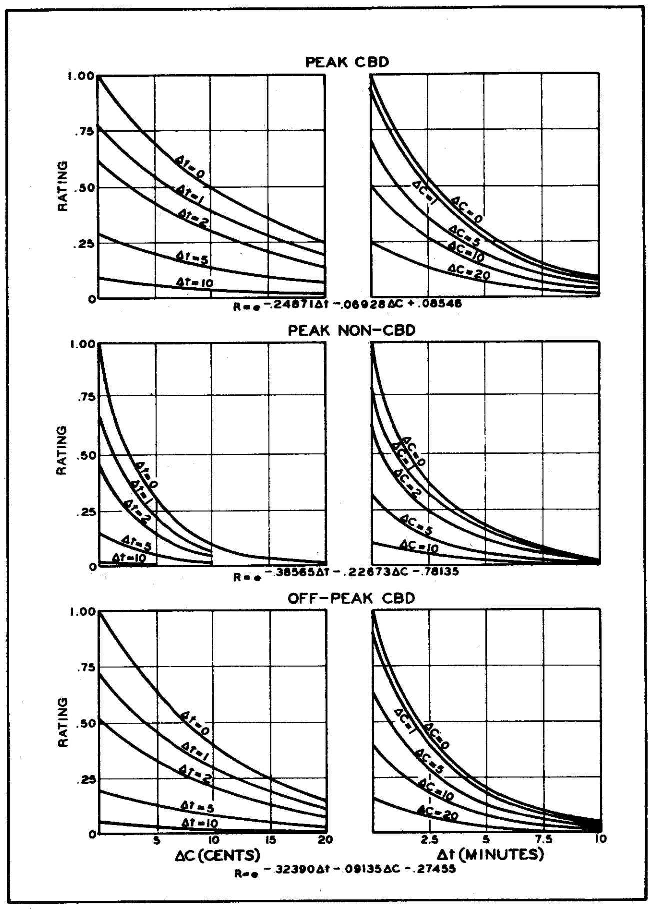


Figure 6—Bus assignment models

are calculated for each facility within each origin-destination pair for each mode. Thus, for the automobile allocation model, where five auto crossings are considered, each origin-destination pair can theoretically contribute five data points. In this study, each origin-destination pair contributed fewer since only those facilities that were within twenty minutes of the fastest were deemed worth considering. Needless to say, few if any trips were found in that excluded category.

When using the model to forecast facility usage, it is not necessary to find the most heavily used facility. The rating for each facility, being the dependent variable, is determined by the time, cost, and transfer differences. The share of the total traffic for each facility is the ratio of its rating to the sum of all the ratings. Graphs of the models are shown in Figures 5, 6 and 7.

Modal split models

Having studied many approaches to modal split as well as having tried a few ourselves, it was decided to

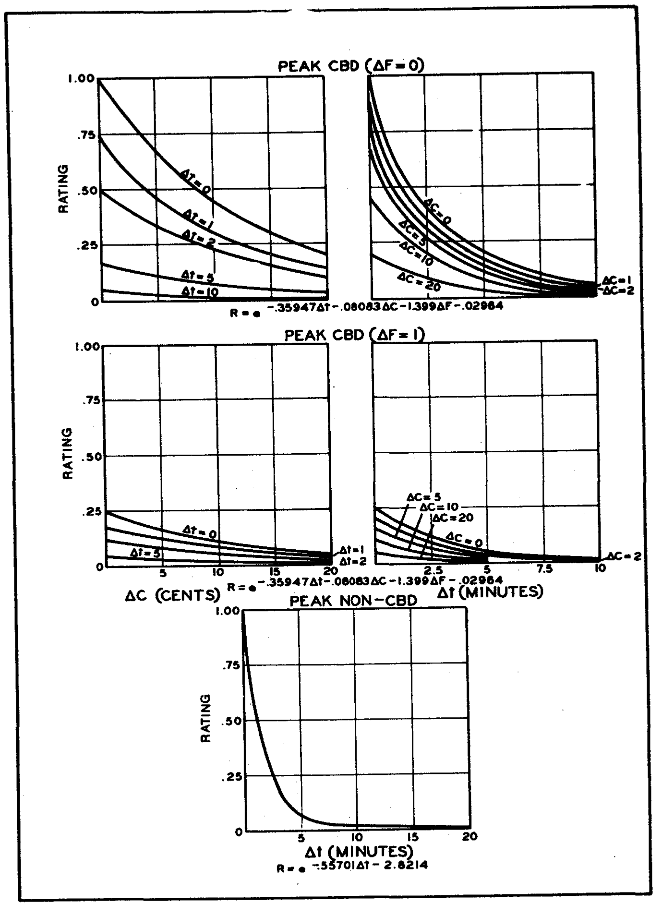


Figure 7—Rail assignment models

attempt to develop the modal split models in two stages—first, would be a split between the two forms of public transport, bus and rail; then a split between public transport and auto. The reason this approach was taken was that the motivations for using auto or public transport seemed to us entirely different from a choice between two different means of public transport. The regression runs at least partially proved us correct.

The bus versus rail model was derived by regressions using travel time, travel costs and number of transfers. Early attempts at deriving meaningful models indicated that we were not explaining nearly enough of the variation with just those variables. It was then decided to investigate some sort of service index. Considering the problems of forecasting an exact service index we chose instead to classify areas according to a frequency of service ratio. In that way, we could be reasonably sure we could approximate this classification for forecasting purposes. The bus vs. rail models were subsequently grouped into four groups and modeled separately. These groups were bus predominant, competitive, rail predominant and PATH areas. Each zone was classified into one of these groups based on the ratio of service frequency of bus and rail. The latter group was separated out because PATH was a rail service that had a frequency of service more like a bus service, and did not fit within the definition of the classification index.

Trial runs of the early bus vs. rail models indicated that certain zones on the trip destination end, particularly The Manhattan CBD, were being systematically over or under estimated. A search for reasons indicated that the Lower Manhattan area—the focus of most of the NJ rail service—exhibited entirely different characteristics than the remainder of the CBD. When we separated this area and ran separate models, the explanation of the variation was much higher and conversely the reactions to the remaining variables were much lower.

The following were the equations finally derived for the bus versus rail modal split:

MODAL SPLIT MODELS
Bus vs. Rail

Bus zones—

$B/B+R = .80 + .00373\ (tr - tb) + .337\ (Cr - Cb)$ $(R = .55)$

Competitive zones—

$B/B+R = .231 + .0064\ (tr - tb) + .522\ (Cr - Cb)$
downtown $(R = .41)$

$B/B+R = .436 + .0139\ (tr - tb) + .65\ (Cr - Cb)$
other CBD $(R = .73)$

Rail zones—

$B/B+R = .261 + .1705\ (tr/tb) + .1868\ (Cr/Cb)$
downtown $(R = .36)$

$B/B+R = .928 + .745\ (tr/tb) + .4829\ (Cr/Cb) + .038$
$(Fr - Fb)$ $(R = .65)$

PATH zones—

$B/B+R = .220 + .0055\ (tr - tb) + 1.01\ (Cr - Cb) + .097$
$(Fr - Fb)$ $(R = .94)$

$B/B+R$ = ratio of bus trips to total transit trips

tr = rail travel time (minutes)
tb = bus travel time
Cr = rail travel cost (dollars)
Cb = bus travel cost
Fr = number of rail transfers
Fb = number of bus transfers
R = multiple correlation coefficient

The second stage of the modal split process was the auto versus public transit split. In the earliest attempts at deriving this set of models, we had assumed that since the percentage auto usage to the CBD was much lower than to the remaining areas east of the river, we would try to derive a separate model for the CBD. The variables included in the trials were employment density east of the river, population density west of the river, travel time difference or ratios, travel cost differences or ratios and parking costs.

The resultant regression equations explained very little of the variation in the percent auto, however, the analysis of the results compared to existing trip patterns showed that where bus was the predominant public transit mode, there was a larger percentage of auto trips than where rail was the predominant transit mode. This finding indicated that while the choice between bus and rail might be a different one from the choice between auto and public transit, there also appears to be a difference in the choice of auto versus bus and auto versus rail. Rather than establish three sets of equations (auto vs. bus, auto vs. rail and bus vs. rail), which would have to be normalized to sum to 100 percent, it was decided to try the percent bus of total public transit as a variable in the auto vs. public transit modal and still depict the difference in choice between auto and the two public transit modes. When the percent bus was entered as a variable, the regression equation proved to be dominated by this variable, but still did not explain enough of the variation in percent auto in the CBD. Similar trials with non-CBD traffic only had even less explanation.

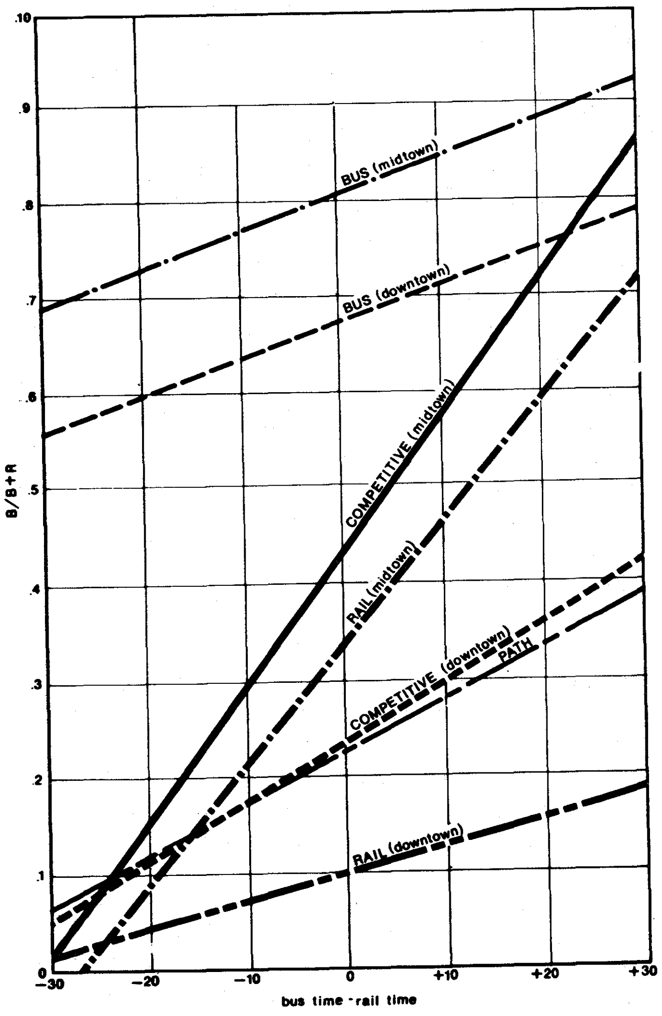


Figure 8—Bus vs rail modal split CBD destinations

Observation of the range of values of the independent variables led us to discover that while the range of values of many of the variables did not explain the large difference between percent auto to CBD and to the non-CBD, two of the variables, employment density and parking cost, did seem to be highly correlated with the percent auto if the CBD and non-CBD trips were combined. A regression run using all the observations showed a relatively high degree of explanation. It did not seem to go far enough towards explaining differences between percent auto within the CBD and those within the non-CBD. In order to correct this, the finding that the percent bus variable was highly explanatory, in the CBD, was combined with the general equation derived for both CBD and non-CBD observations, and the model proved reasonably successful in depicting the general pattern of auto as a percent of total traffic.

The model as finally established was:

$$A/A+B+R = .65 - .091 \ln (Ej/Aj) - .033 \ln (Pi/Ai) - .0068 \ln (PC) + .1279 \ln (tp/ta) + .175\ (B/B+R)$$

Multiple correlation coefficient = .76

Where Ej/Aj	= Employment density of east of Hudson zones
Pi/Ai	= Population density of west of Hudson zones
PC	= Parking cost
tp/ta	= ratio of transit travel time to auto travel time
$B/B+R$	= ratio of bus trips to total transit trips.
ln	= natural log

A graph of the model is shown in Figure 9. It was developed using the (Ej/Aj) variable as the basic variable and shows the effect of the other variables as they extend to their maximum value range. For example, if a zone interchange were between a zone with

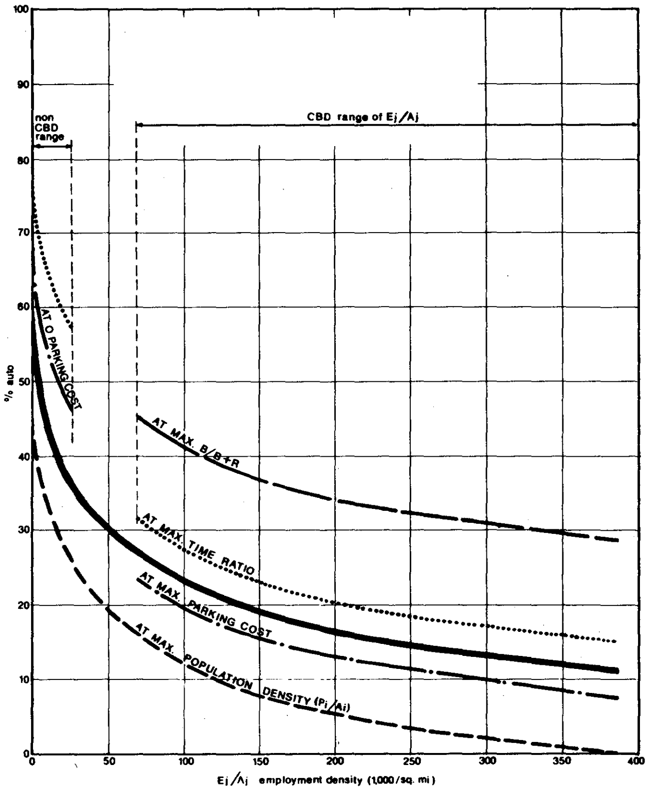


Figure 9—Auto vs public transit modal split

(Ej/Aj) of 300,000 jobs per square mile and one of the highest (Pi/Ai) zones it would have $13-11=2$ percent auto. Including the remaining variables would drop it below 0 percent if it had the highest parking cost, raise it back up by 4 percent if it had a high (tp/ta) ratio and raise it an additional 17.5 percent if all the transit trips were by bus. That zonal interchange would then be predicted to have about 20 percent of its trips by auto.

Also shown on the graph is the range of values for (Ej/Aj). It can be seen why that variable explains so well the percent auto since the CBD employment density is so much higher than that of the other areas.

Trip interchange models

In this model we were attempting to develop relationships that describe total trips between a zone on one side of the river and a zone on the other side. Previous studies in this subject have concentrated more on a concept of trip generation and trip attraction where the trips from the sending zones are estimated separately from the trips to the receiving zones and then a balancing of trip interchanges is made. This process lends itself to a gravity concept that postulates that trips are generated by population, attracted by employment and vary by some function of the distance between zones. The function is usually a decay function and short trips predominate in the model description of trip patterns—which is quite true to life.

The trans-Hudson trip market is only a very small portion of all trips taken in the region and it is a portion that includes mostly longer trips, Therefore for our approach we tried to go directly to trip interchange and we developed a method that is based on segregation of geographic areas on each side of the river that, by earlier study, exhibited different trip patterns.

The general theory behind this approach to trip interchange is that communities change as they age, and there are several directions of change that they can take. This can best be explained by discussing the types of areas we used for our classification systems. There were 5 separate area types west of the Hudson; (2) urban core: old densely developed areas near the river, (4) urban self-sufficient: also older areas but further from the river with more or less their own economic base, (6) stable suburban: old areas originally developed as bedroom communities with little economic base of their own, (5) mid-suburban: newer areas fast-growing in the recent years with a mixed orientation, (1) emerging suburban: sparcely settled areas now with growth expected in the future. Further, there were 3 separate area types east of the Hudson: (3) Manhattan CBD, (7) urban areas—includes most other New York City zones, (8) suburban areas—New York suburban counties.

From our past studies it has been shown that each of these interchange groups exhibited different characteristics as to trips interchanged with zones on the other side of the river as well as different socio-economic and demographic characteristics.

Further analysis of the characteristics of these areas indicated that they could be classified into separate groups by study of economic and demographic data relating to each of the zones. The classification of these areas is presently done with a non-rigorous method of observation, but we have just begun using the statistical technique of descriminant analysis for a more precise classification system and it seems to be working well. It has proved our original classification to be accurate in most cases and has given us further insight to troublesome zones.

While the models we have developed so far still have many shortcomings, they appear to verify the general theory and the variables used are logical ones. It can be seen from the list of equations that are now being used that the effect of the predominant trip-producing variable (population) varies considerably between area type groupings. Further the gravity theory of trips varying with distance between zones (represented by travel time) is maintained with the inclusion of the time variable. The CBD models are shown in a graph form in Figure 10. The scales of the graphs should be noted since the lines plotted indicate the range of the population and trips within each of the area types. The slope of the lines indicates the effect of population on trips and the brackets at the end of each line indicate the range of the effect of employment on trips. It can be seen that Area Type 4 zones produce a small amount of trips per capita and do not react very much to employment attractions on the east side of the river. These are the "self contained areas." Area Type 6 zones, on the other hand, have higher trips per capita

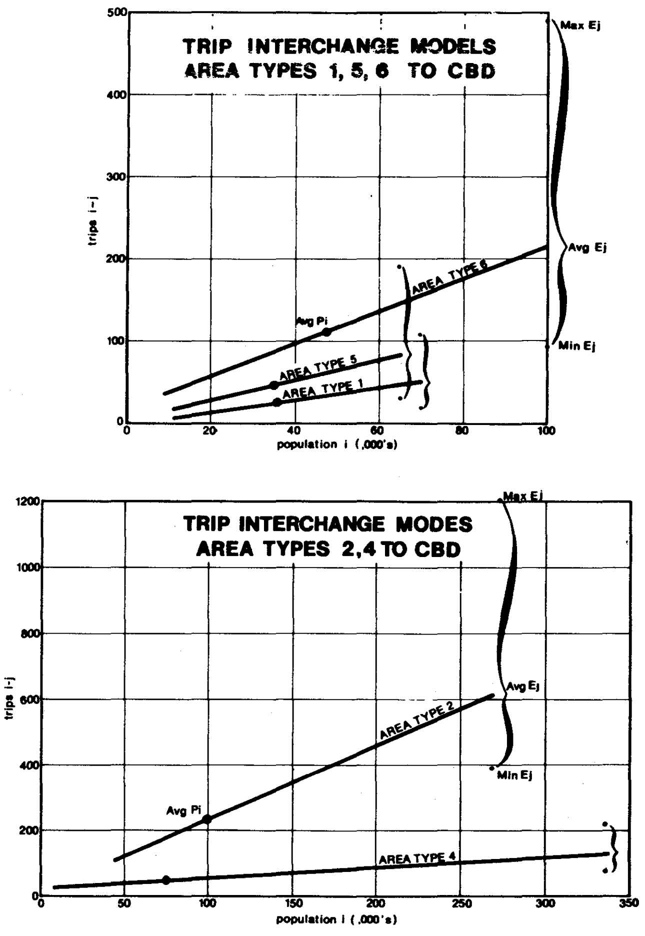


Figure 10—Trip interchange models

Table I—Trip Interchange Models

Area Types	CBD Zones	
2-3	Tij = −125+2.36 Pi+3.16 Ej−.487 tij	(R = .65)
4-3	Tij = +5+.307 Pi+.514 Ej−.059 tij	(R = .71)
5-3	Tij = −159+44.3 ln Pi+.596 Ej−.059 tij−25 (Ei/Pi)	(R = .71)
6-3	Tij = −60+.197 Pi+1.509 Ej−.131 tij	(R = .75)
1-3	Tij = −3+.682 Pi+.338 Ej−.037 tij	(R = .69)
	Non CBD Zones	
2-7	Tij = −10.4+.124 Pi+.108 Ej+1.12 (Ej/Aj)	(R = .66)
4-7	Tij = 5.0+.118 Pi+.076 Ej+.48 (Ej/Aj) −.028 Atij	(R = .74)
1-7	Tij = 191+6.66 ln Pi+3.60 Ln Ej−0.50 ln (Ej/Aj)−33.57 ln Atij	(R = .74)
5-7	Tij = 167+8.79 ln Pi+7.93 ln (Ej/Aj)−3.59 ln (Ei/Ai)−32.9 ln Atij	(R = .69)
6-7	Tij = 250+13.24 ln Pi+8.17 ln (Ej/Aj) −48.2 ln Atij	(R = .76)
2-8	Tij = 33+4.22 ln Pi−6.5 ln tij	(R = .46)
4-8	Tij = 15+3.82 ln Pi+1.25 ln Ej+1.87 ln (Ej/Aj)−4.74 ln tij	(R = .58)
1-8	Tij = 13+1.9 ln Pi+.4 ln Ej−2.66 ln tij	(R = .44)
5-8	Tij = 67−6.11 ln (Ei/Ai)+7.53 ln (Ej/Pj) −8.2 ln tij	(R = .60)
6-8	Tij = 2.3+16.2 ln Pi+11.2 ln (Pi/Ai)+30.7 ln (Ej/Pj)−6.4 ln tij	(R = .63)

Tij	= total trips between i zone and j zone
Pi	= population (000) in i zone
Ei	= employment (000) in i zone
Ai	= area (square miles) of i zone
Pj	= population (000) in j zone
Ej	= employment (000) in j zone
Aj	= Area (square miles) of j zone
tij	= weighted average travel time from i to j
Atij	= weighted average auto time from i to j
ln	= natural logarithm
R	= multiple correlation coefficient

and much higher attraction to employment. These are the stable suburban, or bedroom communities. The highest trip producers are the Area Type 2 zones which are closest to the river. The trip values for the graphs were calculated using the average travel time for each area type.

Improvements in the models are currently being sought through investigations of additional explanatory variables such as competitive employment opportunities on the same side of the river and employment classification. Further investigations are also being made with different forms of the dependent variable.

With models being developed for the purpose of forecasting, some method must be utilized to judge their forecasting quality. One method is to carefully inspect the way that the models perform in reproducing the basic traffic patterns from which they were developed. The process we developed for this is best described by discussing the master program.

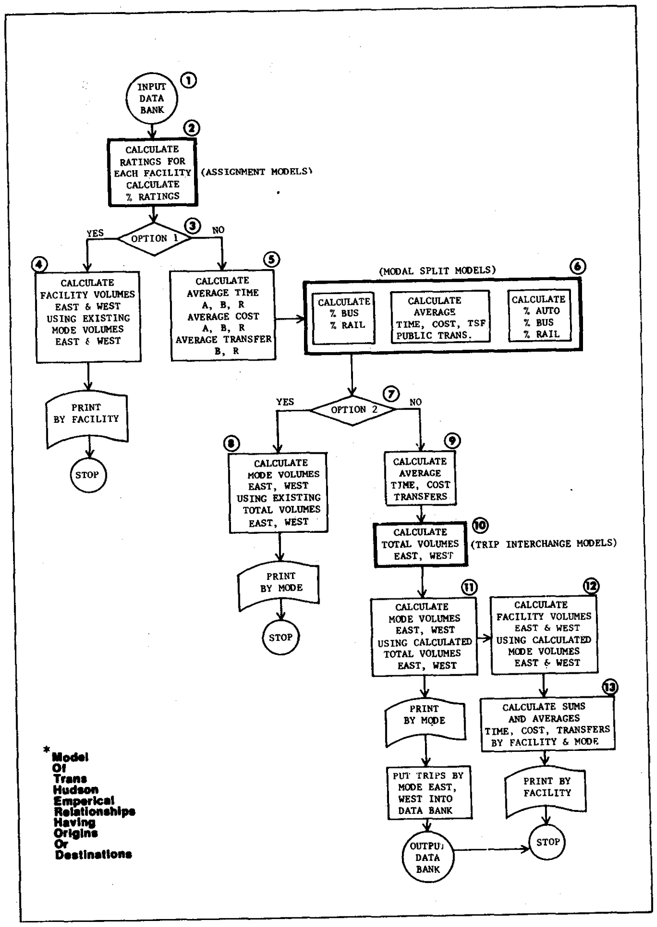

INPUT DATA BANK
1
2
CALCULATE RATINGS FOR EACH FACILITY CALCULATE % RATINGS
(ASSIGNMENT MODELS)
3
OPTION 1
YES
NO
4
CALCULATE FACILITY VOLUMES EAST & WEST USING EXISTING MODE VOLUMES EAST & WEST
5
CALCULATE AVERAGE TIME A, B, R AVERAGE COST A, B, R AVERAGE TRANSFER B, R
(MODAL SPLIT MODELS)
6
CALCULATE % BUS % RAIL
CALCULATE AVERAGE TIME, COST, TSF PUBLIC TRANS.
CALCULATE % AUTO % BUS % RAIL
PRINT BY FACILITY
STOP
7
OPTION 2
YES
NO
8
CALCULATE MODE VOLUMES EAST, WEST USING EXISTING TOTAL VOLUMES EAST, WEST
9
CALCULATE AVERAGE TIME, COST TRANSFERS
10
CALCULATE TOTAL VOLUMES EAST, WEST
(TRIP INTERCHANGE MODELS)
PRINT BY MODE
STOP
11
CALCULATE MODE VOLUMES EAST, WEST USING CALCULATED TOTAL VOLUMES EAST, WEST
12
CALCULATE FACILITY VOLUMES EAST & WEST USING CALCULATED MODE VOLUMES EAST & WEST
13
CALCULATE SUMS AND AVERAGES TIME, COST, TRANSFERS BY FACILITY & MODE
PRINT BY MODE
PUT TRIPS BY MODE EAST, WEST INTO DATA BANK
PRINT BY FACILITY
OUTPUT DATA BANK
STOP
*Model Of Trans Hudson Emperical Relationships Having Origins Or Destinations

Figure 11—Trans-Hudson model—Motherhood Program

MASTER DATA PROGRAM—"MOTHERHOOD"

A system designed for forecasting must have some method of utilizing the developed models to produce a listing of the forecasted trips in the desired detail. The design of the forecasting program was, of course, controlled by the concept of the entire system.

The program developed has been named "Motherhood." This is an acronym representing "Model of of Trans-Hudson Emperical Relationships Having Origins or Destinations." It was dreamed up for us by one of those fellows who has nothing better to do with his spare time other than dream up acronyms. It has done well by us since who, in the long run, can criticize Motherhood. It has also created some interesting dialogue with the programmers who run the system, since many of them are young girls.

Figure 11 is a block diagram of the information flow in the "Motherhood" program. It is indexed by the numbers in circles for easy reference.

The program is designed to accommodate each of the several model stages (assignment, modal split and trip interchange) separately or combined. The thread of continuity in the combined operation is the network data. The program takes the basic facility network data from the data banks, uses it for the assignment model and then reduces the facility data to mode and total travel data for the modal split and trip interchange models in much the same fashion as was done to build Data Bank II for development of the models.

Starting with the input in the data bank where travel times and costs are referenced to a specific facility the program calculates (box 2) the percent of each mode's total traffic that will be assigned to each facility within that mode. A separate equation (model) is used to calculate the percentages within each mode. The program then uses these percentages to calculate (box 5) the weighted average time and cost of each mode. It is these mode averages that are then considered as input to the modal split models. The modal split models (box 6) operate in two stages. First, the percent split between bus and rail is calculated from a model (or set of models) then the weighted average time and cost of public transport is calculated and a percent split is calculated between public transport and auto. The last step is then to calculate a single weighted average time and cost (box 9) so that it can be used as input to the trip interchange model (box 10). It can be seen that the eleven times and costs in the data bank (one for each of the facilities) are now transferred to a single time and cost (representing a weighted average of all facilities) by flow through the program. The trip interchange model using this averaged value of time or cost then calculates total trips between zones. The program, having previously calculated and saved the percent split value on modes and facilities, simply uses these

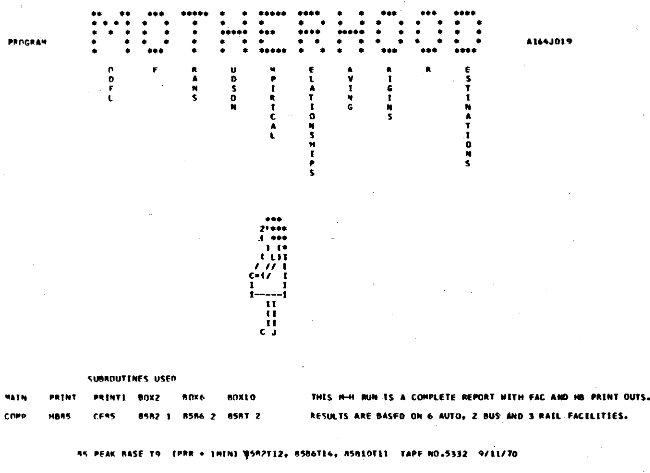

```
PROGRAM                                                        A164J019

              SUBROUTINES USED
MAIN    PRINT   PRINT1  BOX2    BOX6    BOX10      THIS M-H RUN IS A COMPLETE REPORT WITH FAC AND MB PRINT OUTS.
COMP    HBR5    CFR5    B5B2 1  B5B6 2  B5BT 2     RESULTS ARE BASED ON 6 AUTO, 2 BUS AND 3 RAIL FACILITIES.

        B5 PEAK BASE T9  (PRR + 1MIN) B5B2T12, B5B6T14, B5B10T11  TAPE NO.5332  9/11/70
```

Figure 12—Print out example

```
                JOB  A164 J019              FACILITY  VOLUMES  F + W  FOR  OPTION  1     85    COMP         85B2 1  85B6 2  858T 2

                          85 PEAK BASE T9  (PRR + 1MIN) 85B2T12, 85B6T14, 85B10T11  TAPE NO.5332  9/11/70

I     J ALL MODES ---------------- AUTO ----------------  -------- BUS --------  -------------- RAIL --------------
        GR. TOT    TOT    GWB   LT    HT   STB    TZ   NEW    TOT   GWBS  PABT           TOT   PRR   HTR  PUPR
81    1     174     62     60    0     0     0     0     2     29     27     2     0      83     0    82     0     0     0
81    2     306    113    110    0     0     0     0     3     50     46     3     0     143     0   139     3     0     0
81    3      32     17     16    0     0     0     0     1      9      8     1     0       6     0     4     2     0     0
81    4      22     13     11    0     0     0     0     2      5      4     1     0       4     0     3     1     0     0
81    5      34     19     18    0     0     0     0     1      8      7     1     0       7     0     0     7     0     0
81    6      27     17     13    0     0     0     0     4      6      5     0     0       4     0     0     4     0     0
81    7      76     45     42    1     0     0     0     1     14     13     1     0      17     0     0    17     0     0
81    8      57     37     28    0     0     0     0     8     10      9     2     0      10     0     0    10     0     0
81    9      89     45     43    1     0     0     0     1     20     17     3     0      24     0     0    24     0     0
81   10      60     34     26    0     0     0     0     7     13     11     2     0      12     0     0    12     0     0
81   11      80     42     38    3     0     0     0     1     23     21     2     0      15     0     0    15     0     0
81   12      75     35     26    1     0     0     0     7     22     18     4     0      19     0     0    19     0     0
81   13      64     31     26    3     0     0     0     1     21     15     6     0      12     0     0    12     0     0
81   14      63     33     24    1     0     0     0     7     18     15     3     0      12     0     0    12     0     0
81   15     118     61     55    3     0     0     0     2     58     46    12     0       0     0     0     0     0     0
81   16     106     54     42    1     0     0     0    12     49     39    10     0       2     0     0     2     0     0
81   17     122     70     67    0     0     0     0     2     52     47     5     0       0     0     0     0     0     0
81   18     142     69     53    1     0     0     0    15     68     58    10     0       5     0     1     4     0     0
81   19     155     86     83    0     0     0     0     3     70     68     1     0       0     0     0     0     0     0
81   20     123     63     48    0     0     0     0    15     57     52     5     0       3     0     0     3     0     0
SUB TOTAL
     20    1924    945    832   17     0     0     0    96    600    526    74     0     378     0   231   147     0     0
81   21     115     76     74    0     0     0     0     2     39     39     0     0       0     0     0     0     0     0
81   22      94     65     51    0     0     0     0    14     29     29     0     0       0     0     0     0     0     0
81   23     111     78     75    0     0     0     0     4     33     33     0     0       0     0     0     0     0     0
81   24     108     81     67    0     0     0     0    14     27     27     0     0       0     0     0     0     0     0
81   25     120     92     89    0     0     0     0     3     27     27     0     0       0     0     0     0     0     0
81   26      99     79     58    0     0     0     0    21     20     20     0     0       0     0     0     0     0     0
81   27     117     93     74    0     0     0     0    19     25     25     0     0       0     0     0     0     0     0
81   28      96     83     30    0     0     0     0    53     13     13     0     0       0     0     0     0     0     0
81   29     111     99      2    0     0     0     0    97     12     12     0     0       0     0     0     0     0     0
81   30     100     93      0    0     0     0     0    93      7      7     0     0       0     0     0     0     0     0
81   31      57     55     20    0     0     0     0    35      2      2     0     0       0     0     0     0     0     0
81   32     569    569      0    0     0     0     0   569      0      0     0     0       0     0     0     0     0     0
81   33    1003   1003      0    0     0     0   403   600      0      0     0     0       0     0     0     0     0     0
81   34     383    383      0    0     0     0    48   335      0      0     0     0       0     0     0     0     0     0
81   35     131    131      0    0     0     0     0   131      0      0     0     0       0     0     0     0     0     0
81   36     136    136      0    0     0     0     0   136      0      0     0     0       0     0     0     0     0     0
81   37     341    341      0    0     0     0     0   341      0      0     0     0       0     0     0     0     0     0
81   38       8      6      4    0     0     0     0     2      2      2     0     0       0     0     0     0     0     0
81   39       2      1      1    0     0     0     0     1      1      1     0     0       0     0     0     0     0     0
81   40      16     12      9    0     0     0     0     3      3      3     0     0       1     0     0     0     0     0
81   41       4      3      2    0     0     0     0     1      1      1     0     0       0     0     0     0     0     0
81   42       1      1      0    0     0     0     0     0      1      1     0     0       0     0     0     0     0     0
81   43      17     13      9    0     0     0     0     4      3      3     0     0       1     0     1     0     0     0
81   44      12      9      2    0     0     0     0     7      2      2     0     0       1     0     1     0     0     0
81   45      16     12      3    0     0     0     0    10      3      2     0     0       1     0     0     0     0     0
81   46      32     20     15    0     0     0     0     5      9      9     0     0       3     0     3     0     0     0
81   47      26     19     14    0     0     0     0     5      7      7     0     0       1     0     0     0     0     0
81   48      33     24     17    0     0     0     0     7      9      8     1     0       1     0     0     1     0     0
81   49      21     18     13    0     0     0     0     5      4      4     0     0       0     0     0     0     0     0
81   50      22     19      1    0     0     0     0    18      3      3     0     0       0     0     0     0     0     0
81   51      16     15      0    0     0     0     0    14      1      1     0     0       0     0     0     0     0     0
```

Figure 13—Print out example

percentages to distribute the total trip volume into trip volumes by mode and facility.

You can note also on the diagram that there are optional routes through the program. These are for model testing purposes so that each of the models can be tested and calibrated separately. A test of the assignment model can be made by running the program with option 1 on (box 3) where we can take the given number of trips on each mode and simply assign them to facilities. In order to test the modal split model, the program must have both the assignment model and the modal split models in so that the weighted averages of the times and costs can be made to input data in the modal split models. And lastly, a test of the trip interchange model must be made with all three models (box 2, box 6 and box 10) in the program, so that the average times and costs can be calculated by the two previous models as input to the trip interchange model.

There are several print options in the program. The major option is a facility print or mode print where residence east trips and residence west trips are separated (see Figures 12, 13 and 14). These options are used so that the analyst may review the forecast in the manner most appropriate to his investigation. The residence split being most important in the trip interchange review and the facility print most important in the assignment and in the review of performance of the overall system.

Additional print options are available that collapse and sum the trip forecasts. The basic print is in an $i-j$ format, i.e., each cell is printed. One summation is by county groups, another sum is by rail corridor groups

```
               JOB  A164  J019            MODE  VOLUMES  EAST  AND  WEST  FOR  OPTION  2  85    COMP       85B2 1  85B6 2  85BT 2

                             85 PEAK BASE T9  (PRR + 1MIN) 85B2T12, 85B6T14, 85B10T11  TAPE NO.5332  9/11/70

I   J                                      --------- AUTO ---------    ---------- BUS ----------    --------- RAIL ---------
         GR. TOT   TOT. E   TOT. W       TOTAL     EAST     WEST        TOTAL    EAST     WEST        TOTAL     EAST     WEST
81   1      174       2      172          62        1       61           29       1       28           83        0       83
81   2      306       2      304         113        1      113           50       1       49          143        0      143
81   3       32       2       30          17        1       16            9       1        8            6        0        6
81   4       22       2       20          13        1       13            5       1        4            4        0        4
81   5       34       2       32          19        1       18            8       1        7            7        0        7
81   6       27       2       25          17        1       16            6       1        5            4        0        4
81   7       76       2       74          45        1       44           14       1       13           17        0       17
81   8       57       2       56          37        1       36           10       1       10           10        0       10
81   9       89       2       87          45        1       44           20       1       19           24        0       24
81  10       60       2       58          34        1       33           13       1       12           12        0       12
81  11       80       2       78          42        1       41           23       1       22           15        0       15
81  12       75       2       73          35        1       34           22       1       21           19        0       19
81  13       64       2       62          31        1       30           21       1       20           12        0       12
81  14       63       2       61          33        1       32           18       1       17           12        0       12
81  15      118       2      116          61        1       60           58       1       57            0        0        0
81  16      106       2      104          54        1       53           49       1       48            2        0        2
81  17      122       2      120          70        1       69           52       1       51            0        0        0
81  18      142       2      140          69        1       68           68       1       67            5        0        5
81  19      155       2      153          86        1       85           70       1       68            0        0        0
81  20      123       2      121          63        1       62           57       1       56            3        0        3
SUB TOTAL
    20     1924      38     1885         945       18      927          600      20      580          378        0      378
81  21      115       4      111          76        2       74           39       2       37            0        0        0
81  22       94       3       91          65        1       63           29       2       27            0        0        0
81  23      111       4      107          78        2       76           33       2       31            0        0        0
81  24      108       3      105          81        1       80           27       2       25            0        0        0
81  25      120       4      116          92        2       90           27       2       26            0        0        0
81  26       99       2       96          79        1       78           20       1       18            0        0        0
81  27      117       4      113          93        2       91           25       2       23            0        0        0
81  28       96       3       94          83        1       82           13       1       12            0        0        0
81  29      111       2      108          99        1       98           12       1       10            0        0        0
81  30      100       2       98          93        1       92            7       1        6            0        0        0
81  31       57       2       55          55        1       54            2       1        1            0        0        0
81  32      569       1      568         569        1      568            0       0        0            0        0        0
81  33     1003       1     1002        1003        1     1002            0       0        0            0        0        0
81  34      383       1      382         383        1      382            0       0        0            0        0        0
81  35      131       2      129         131        2      129            0       0        0            0        0        0
81  36      136       2      134         136        2      134            0       0        0            0        0        0
81  37      341       1      340         341        1      340            0       0        0            0        0        0
81  38        8       2        6           6        1        5            2       1        1            0        0        0
81  39        2       1        1           1        1        1            1       1        0            0        0        0
81  40       16       2       14          12        1       11            3       1        2            1        0        1
81  41        4       2        2           3        1        2            1       1        0            0        0        0
81  42        1       1        0           1        1        0            1       1        0            0        0        0
81  43       17       2       15          13        1       12            3       1        2            1        0        1
81  44       12       2       10           9        1        8            2       1        1            1        0        1
81  45       16       3       13          12        1       11            3       1        1            1        0        1
81  46       32       1       31          20        1       20            9       1        8            3        0        3
81  47       26       3       23          19        2       17            7       2        5            1        0        1
81  48       33       3       31          24        1       22            9       1        7            1        0        1
81  49       21       2       19          18        1       17            4       1        3            0        0        0
81  50       22       2       19          19        1       18            3       1        2            0        0        0
81  51       16       2       14          15        1       14            1       1        0            0        0        0
```

Figure 14—Print out example

and the last sum is by area type groups. An additional feature is built into the program that converts auto passenger trips to auto trips so that analysis can be made on a vehicular basis.

Another feature of the master program is that it can be used to read and print out the data within any of the data banks. This is a necessary procedure when one considers the massive amount of data in the banks and the necessity of maintaining accuracy when changes are made.

Perhaps the most important feature of the master program is that the model sections (boxes 2, 6 and 10) are completely flexible. They are branches within the master program and can be changed simply by rewriting the FORTRAN statements within each branch. This allows the user to change the entire model or parts of it. It allows us to use the master program for the peak and off-peak models with equal ease simply by replacing the model sections of the program. Of great importance is the fact that this master program can be used to test models developed at a later time based on some new data and new research.

CURRENT USE OF THE SYSTEM

The peak period models have already been developed to a point where we think they can be used for forecasting. The basic models were derived from 1964 data and they were tested by running them through the "Motherhood" program. Because a multiple regression program does not provide one with a perfect fit of the data, the test runs provide for analysis of the output. These analyses point to areas where additional variables could be used, where network errors might have been made, and where certain phenomenon simply cannot be explained adequately. Correcting for these items is called calibration.

The 1964 models were calibrated in several stages. First, the assignment models were run and calibrated. Then the assignment models were used along with trial modal split models and the two stage runs were calibrated. Finally, the calibrated assignment and modal split models were run with the trip interchange models in the full three stage run. This way, the entire set of models could be considered calibrated to reproduce the 1964 data in the same procss that the 1985 forecast runs would be made.

Many data summaries were made to make sure that traffic patterns derived from the models were reasonably in line with those found in the original trip data. Areas that were checked were those where specific transportation improvements were to be made or where alternate systems were to be tested. The following table shows some of the calibrated data summaries made on the 1964 trial runs.

For the forecast year 1985 an initial transportation network had to be constructed. The network data was derived by an analysis of the plans and programs of all the transportation agencies in the region. The time, cost and transfer effect of new or improved facilities that would be in place and operating by 1985 were coded into the networks to depict a base network. These included new highways, improved rail service and some improved bus service resulting from new highways.

TABLE II—1964 Peak Period Trans-Hudson Model Calibration Facility Comparisons

	Actual	Assignment Model Only	Assignment and Modal Split Models	Assignment Modal Split and Trip Interchange Models
Grand Total	159 438	159 398	159 394	158 659
Auto	45 909	45 909	46 199	46 230
GWB	23 560	23 675	22 847	23 091
LT	11 159	11 677	12 618	12 466
HT	3 214	3 269	3 474	3 586
SIB	2 770	2 540	2 534	2 672
TZ	5 205	4 749	4 726	4 416
Bus	58 845	58 833	58 142	57 195
GWB	11 268	10 270	10 860	10 765
PABT	47 577	48 563	47 282	46 430
Rail	54 684	54 658	55 056	55 235
PS	7 593	7 449	7 543	7 504
HT	26 060	25 652	25 282	25 199
PUP	13 153	12 885	13 521	13 954
CNJ	7 878	8 671	8 712	8 578

Additional data necessary to run a 1985 trial forecast was the forecasted independent variable data—population, employment, area types for the trip interchange model, and service classification for the rail vs. bus part of the modal split model.

Trial forecasts for a basic assumed transportation system have been run and are being analyzed. Several alternate systems have also been coded into the network and run and we are analyzing the reaction to the changes caused by the various alternates. Further work in this area includes reassessing the 1985 population and employment forecasts based on later census data.

The off-peak models based on 1964 data are still under development. Because off-peak traffic exhibits a great amount of variation, it is difficult to "zero in" on what causes the variation. First, the market is split, being part work oriented travel and part non-work. Another problem is that 21 hours of traffic are included in the data some hours of which may differ from others. Still another is that public transport usage is extremely variegated in this off-peak market perhaps being influenced by income in some areas or by entertainment centers in other areas, etc. Unfortunately, few of these types of influences can be accurately forecast.

While we are working on the off peak models, we continue to search for more usable explanatory variables for the peak models. In this assignment model, we had difficulty duplicating representative patterns for users of the two bus terminals and we are working on a variable that will attempt to define more clearly the difference in access to the different bus routes on the New Jersey side. This same variable and one like it for access to the rail service might help explain more of the variation in the bus vs. rail Modal Split models. We are also experimenting with variables that might better describe the differences in distribution on the New York side, particularly in the CBD. We are now using travel time, cost and number of transfers involved in the entire trip, and it seems as though these variables do not focus strongly enough on the attractiveness of a trip that requires no additional mode for distribution from the terminal to the final destination.

In the trip interchange model we are currently using total trips as the dependent variable but we are trying other forms such as trips per capita. We are also introducing new variables such as competing employment opportunities within 20 minutes of the origin zone and managerial and professional employment. We must be careful with the latter since it may be difficult to forecast.

Earlier in the paper it was stated that one of the aims of the system was to allow for updating and using new data. In meeting this requirement, we are now building a new data bank to represent the base year 1968.

The trip data for that year is available from numerous O and D surveys. Auto data comes from the continuous O and D sample and O and D surveys were taken that year at the two bus terminals, on the Penn Central Railroad and on the PATH system. Only minor adjustments to the original trip data programs are necessary to format these data for use in the model systems.

Population data for 1968 will be estimated from the 1970 census of population and employment data will be extracted from continuous sources of employment data available from New York and New Jersey State agencies.

The travel time, costs, transfers and other network variables will be updated from sources of permanent records available such as service schedules, fare tariffs and sample travel time runs. With the utility available from the battery of programs that have been written and from the design of the systems we now have a capability long sought after in urban transportation studies. We can update on a short time cycle so that we can integrate time series analysis with the new forecasting system. We can now derive new models with a 1968 base and compare them with those derived from the 1964 base. We can attempt to forecast 1968 from the 1964 models and analyze the differences from actual performance. With this technique we hope to have greater insight to the causes of traffic pattern changes both for short run analysis and long range forecasting.

Computer graphics for transportation problems*

by DAN COHEN and JOHN M. McQUILLAN

Harvard University
Cambridge, Massachusetts

INTRODUCTION

The central problem in designing transportation systems and networks is determining the optimal control techniques for given transportation facilities. For example, it is essential to find the best strategy for handling the traffic in a given airspace, in a given highway system, or in a network of city streets. The other side of the problem is to determine for given or predicted traffic conditions, the optimal transportation facilities. Urban planners must solve these problems when designing new developments; similarly it is important to determine how many airways and airports will be required to handle the air traffic in the 80's. From the answers to such questions one can decide how to allocate funds, for example, to improve the radar systems to allow smaller separation or to put more navigation aids in order to increase the number of airways.

It is not just a mere coincidence that in many languages the word "see" and "understand" are synonyms. In many cases to see is to understand, and this is what computer graphics is all about.

Computer graphics is used mainly as an interface between the man and the machine. Problems which inherently require display of output or have graphically oriented input are the clearest beneficiaries of computer graphics. Graphical output gives the ability to display arbitrary shapes quickly. Graphical input provides the ability to define shapes and the ability to identify things naturally by pointing to them. Transportation systems are often best represented graphically. For these reasons we have found that the application of computer graphics techniques to the solution of transportation problems is most fruitful.

In this paper we discuss the philosophy behind our approach, and illustrate it with examples taken from specific programs. A ten minute film will be shown to demonstrate the application of interactive computer graphics for urban traffic problems.

COMPUTER APPROACHES FOR TRANSPORTATION SYSTEMS

It is often the case that practical problems deal with system behavior, rather than behavior of a single particle or a single element. Describing and dealing with systems is manyfold more complex than working with a single element. Often one can describe very precisely the exact mathematics which govern the behavior of a single element. However, it is very seldom that one can find equations which describe a system completely, and still be consistent with the behavior of each of its elements.

Simulation of urban traffic, or air traffic, are examples of this difficulty. One can describe very precisely the motion of a single car or of a single airplane. If the motion of the car is unrestricted, then its behavior is simple to explain. When more than one element is introduced into the system, the interaction between them adds a new dimension to the problem. The complexity of the interactions might grow as the square of the number of objects in the interaction. In general, one can solve situations where few vehicles are involved. However, any practical problem involves too many objects for a human being to solve without a computer.

In many transportation problems, there is a system of many particles moving concurrently in the same space, obeying some interaction restrictions. These restrictions are usually in the form of separation criteria (for cars, airplanes, ships, etc.), staying in some corridors (like highways, airways, etc.) sharing some navigation facilities and so on. Such system problems lend themselves very well to computer use. In order to solve these transportation systems on a computer, one can

* This research was supported in part by ARPA Contract F19628-68-C-0379.

use simulation techniques, rather than integrating equations into system-behavior. A computer can perform the tedious job of simulation particle by particle and make local decisions about each of the particles. In some systems these decisions are based only on local information, as observed by each particle. In other systems these decisions may depend on global information about the state of the system.

If the behavior of each individual particle is non-deterministic and some distribution and probabilities are involved in the description of each particle, then the behavior of the entire system is non-deterministic; in order to simulate it properly one has to simulate the distributions. These non-deterministic simulations have to be repeated many times in order to average the behavior and the distribution to get meaningful results. Clearly, it is appropriate to use a computer for such simulations.

Computer graphics lends itself very well to this kind of simulation. After every updating cycle, one can display the state of the system. For example, if one simulates the air traffic in a given space based on some known rules, one would like to observe the dynamics of the system changes. The visual display of this information, at a rate meaningful for the viewer, might introduce new understanding of the behavior of the system. In the case of traffic simulation, whether it is urban traffic or air traffic, one can learn a great deal by viewing the intermediate steps through which the system is going.

For example, one might observe that due to some latency in traffic lights, some cars happen to jam an intersection, which in turn might cause a total breakdown of the traffic flow. If the conditions of cause and effect are not known in advance, global measures are not enough to explain this kind of behavior. The only way to understand the system is by viewing it, and recognizing its behavior patterns. These patterns, which are not known before they are observed, rely very highly on the intelligence of the human being and his ability to recognize patterns. If the behavior patterns are already known, one might assign the computer to look for them, measure them, and report them. This can be done off-line (batch processing, for example) and interactive graphics is not needed for it. However, in many cases the internal behavior patterns are not known and one has no idea what to look for, and cannot assign the computer to search for and measure them. The dynamic graphic simulation allows one to see and recognize behavior patterns which he never expected to find, and watch them develop. This recognition leads to an improved understanding of the system.

INTERACTIVE COMPUTER GRAPHICS FOR SIMULATION PROBLEMS

Under many circumstances, the best use of a computer simulation is an interactive one. There may be so many variables that the only way to understand their interdependence is to study the problem in real-time simulation, seeing how it is affected by various changes. There may be such uncertainty in the model itself that the parameters should be altered as the simulation proceeds. Using this approach, one can quickly gain a good understanding of the model's strengths and weaknesses, its suitability to certain situations, and its sensitivity to incremental changes of many kinds. Such an intuitive appraisal of a model is frequently more valuable than extensive numerical evaluations. The conditions of a smoothly flowing dialogue are decidedly more conducive to thought than the use of a computer merely as a calculating machine.

We felt that an interactive system would be desirable in view of the nature of problems in traffic flow. They are infinite in variety, yet they can be formulated intuitively. We have daily experience with many of these problems, and we know how traffic behaves under many conditions. Since these perceptions are often difficult to include in a precise model, it is to our advantage to exercise a model in an interactive way, and supply it with our reactions as the simulation takes place. We are then employing the computer where it is most useful in the problem-solving process.

Interaction with a computer simulation becomes much easier for a man, as well as a more valuable technique, when results are supplied quickly and clearly in picture form. Pictures carry immediate meaning; details and patterns can be recognized easily, and factors of cause and effect are evident. When he changes the conditions of the problem, he gets meaningful results right away and is therefore in an excellent position for further interaction. He may continuously change the parameters and see the sensitivity of the system to these incremental differences. This kind of continuous dialogue, uninterrupted by technical details, is a powerful and valuable method of investigation. A man is thus able, with computer assistance for computation and communication, to solve many problems beyond the scope of a man alone or an off-line computer program.

Just as graphical output is the natural form for the machine-to-man communication, graphical input is the natural form for the man-to-machine communication. Many transportation problems require a specification of a map and associated parameters. This can be done initially in a digital form; however, it is much more convenient and natural to input this information graphically, by using stylus-like devices. Furthermore, during run-time the need often arises to identify particular objects, which may be in motion. It is most natural and efficient for a man to do so simply by pointing at them with his stylus.

It is most important to provide the transportation-engineer with natural means for communicating with the computer. He should be able to concentrate solely on solving the specific transportation problem, rather than concerning himself with the details of computer operations.

AN EXAMPLE OF URBAN TRAFFIC

Mr. John M. McQuillan, then a senior at Harvard College, constructed a program to simulate urban traffic, based on the principles of interactive graphics discussed above. The user of the program begins by specifying the street map to be considered. This is accomplished by means of a stylus and a tablet. The user specifies the position of the streets, their direction and number of lanes, and the program draws in the streets and the intersections for him. He defines whether an intersection is controlled by a traffic signal, a stop sign, a yield sign, and so on. These symbols are drawn for him automatically. This definition process is interactive, allowing the user to edit the map at any time. He may reposition portions of the map, delete and add sections as he wishes. After the street map is drawn, the user can specify automatic settings for the traffic signals. He does this by drawing a bar graph of the times during a fixed-length cycle when the light is to be green and when it is to be red. He also has the facility of assigning the same setting to other signals, or the same setting with a fixed delay time. He may also specify that certain signals are to be given the same settings and then perform the above operations on groups of signals rather than single ones. In this way, it is relatively easy to construct a strategy of traffic signal settings for a complex network of intersections.

Next, the user directs the program to enter the simulation phase. In this stage, one CRT shows the street map, with cars travelling through it, obeying the traffic laws and signal settings. Each car moves through the streets, turns, pauses, switches lanes, etc., according to information based on the surroundings. This is illustrated in Figure 1. Meanwhile, another CRT shows the automatic signal settings in a form of bar graphs. A cursor moves along these bar graphs, indicating the signal changes as they happen. This is illustrated in Figure 2. At the same time, a third CRT shows a

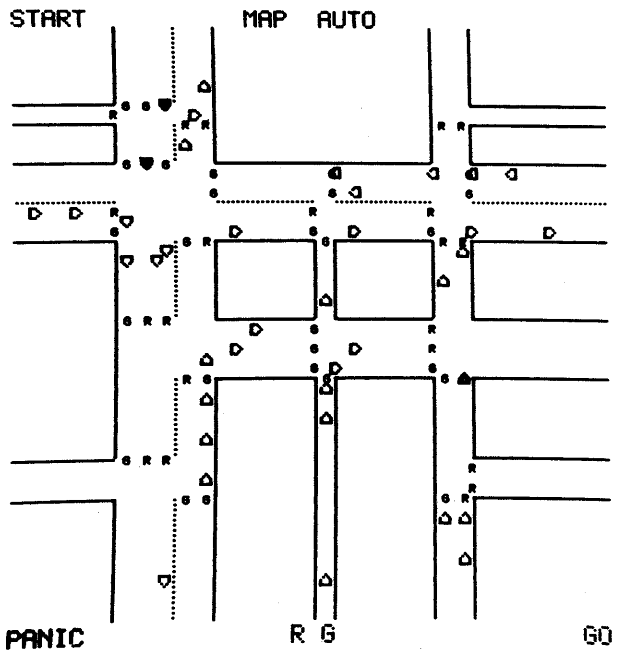


Figure 1—Urban traffic simulation—Cars move in all directions through a user-specified street map. Traffic lights at the intersections are represented by the letters R and G. Words at the edges of the picture are control functions

control panel with bar graphs which govern several parameters. These parameters include traffic density in each direction and some other characteristics. As the simulation proceeds the user may change any parameter merely by pointing to the bar graph with his stylus. This control panel is illustrated in Figure 3. In addition, this third CRT displays instantaneous and cumulative statistics, such as number of cars inside the map, average speed and so on.

The program is designed so that it is natural to use interactively. After specifying one map, the user can try different signal settings under different traffic conditions to find appropriate means of control. He sees the effects of these changes in real time, as traffic flows through the network. He may return to draw in a new map and alter his strategies further, all in an interactive manner. We have found this approach to be a very valuable one in formulating and solving problems in urban traffic.

AN EXAMPLE OF AIR TRAFFIC

The air traffic control problem is a unique problem in the sense that it involves a very complex system of many airplanes sharing the same air space concurrently. In order to describe the system of the air traffic, one needs a dynamic tool which enables him to describe in

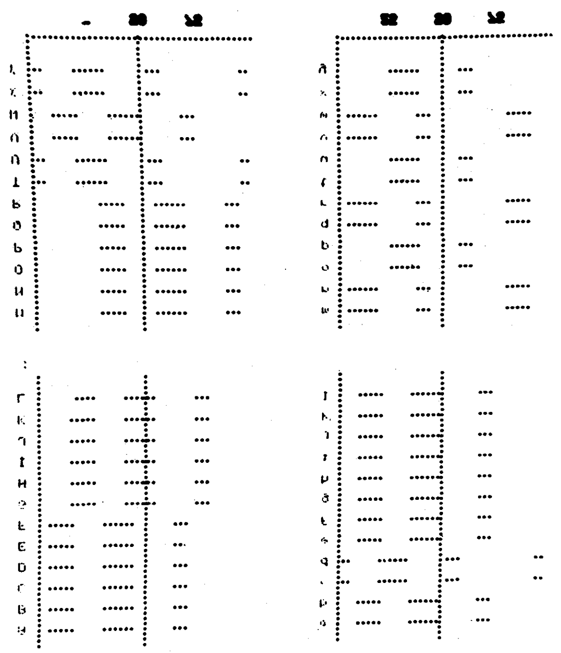


Figure 2—Signal settings for traffic simulation—This is a display of the time settings of the traffic signals (letters on the graphs identify individual signals). For each traffic light, a bar graph represents those times that the light is green and the absence of a bar indicates when it is red, during a 100-second cycle. The vertical bar is a cursor which moves across the graphs in real time

real-time the current positions of many airplanes which move in different directions at the same time. There is no way but graphically to describe the state of the system at any time. In real life the way the air traffic system is described is by graphical means, the radar which is used by the controllers. The control information which is issued is in the form of instructions to the airplanes telling them positioning and timing information, issuing "vectors," instructions for turning, and so on. Because of the nature of the problem, it is desirable to have facilities that enable one to communicate with the system graphically for input control information and to receive the state of the system at any point. For example, a controller should be able to define a route for an airplane merely by drawing the route on the face of a scope rather than verbally describing it. Collision hazards should be represented to the controller by showing him two airplanes whose routes tend to merge, and perhaps flashing some warning lights to attract his attention to this fact. The interaction between the controllers and the real airplanes should benefit from the use of the graphics as well. The controller should be able to point to an airplane rather than calling it verbally. This assumes, of course, that the system behind the graphics is aware of which airplane is where, and can automatically issue some communication to this airplane upon graphical request of the operator. In order to demonstrate these ideas and to provide a training environment, a computer program was developed in our laboratory.

In the first phase of the program, which was written by Mr. Geoffrey A. Modest, then a junior at Harvard College, one can define the map of the area in which he wants to operate, and assign it any arbitrary shape. One can define the shape and the position of the airports to be included in this area. One can define the "Victor" and the "Jet" airways intersecting this area, and can define standard holding patterns. Navigation aids can be introduced into the map in the shape of triangles and squares. This definition state is, of course, interactive. One can change his mind during the definition stage or later by editing the map, changing it, deleting obsolete objects, and adding new features to it.

After the definition phase, the operation phase begins. This section was written by Mr. W. B. Barker, a graduate student at Harvard University. This operation phase requires two people to operate it. One simulates the air traffic controller and the other one simulates concurrently all the pilots of all the airplanes in the area. The "pilot" can issue routing instructions

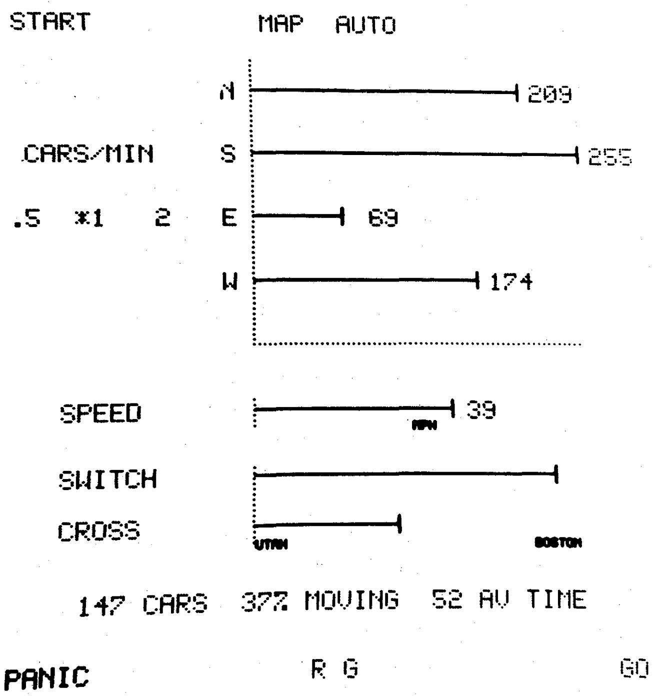


Figure 3—Control panel for traffic simulation—This display provides graphical output and input of traffic density, speed, and driver characteristics. Not only do the graphs show the current value of each parameter, but they may be changed, merely by pointing to an endpoint and moving it to a new position. Statistics describing traffic flow are displayed dynamically at the bottom of the picture

to each airplane in graphical form. The routing instruction may have the form of "climb and maintain flight level 200," and "follow victor 20, turn to victor 16 at station x," etc. The "controller" can see on his scope the position of each airplane and can interrogate these airplanes graphically, requiring information about altitude, speed, identification, and so on. Ideally, the controller should be able to express instructions to the airplanes graphically. However, in order to simulate closely today's systems, the program does not automatically carry the graphical instructions of the "controller" to the airplanes, but the "controller" has to issue them verbally, as if he were talking on radio to the pilot. The "pilot" then can apply these instructions to the airplanes, exactly according to the "controller's" instructions, or he may deviate from them. This way the "pilot" can simulate misunderstandings between the air traffic control and the pilot in the air. The only way that the "controller" can find about these misunderstandings is by noticing, on his "radar," that some airplanes do not follow the instructions that he had issued before. All communication with the airplanes either by the "pilot" or the "controller" is very natural. In order to specify an airplane all they have to do is to "touch" the airplane on the scope with a stylus. All control information is requested graphically, and the flight paths of the aircraft on the radar screen provide the necessary feedback.

FUTURE APPLICATIONS OF COMPUTER GRAPHICS IN TRANSPORTATION

We hope that the programs we have developed will be prototypes for future practical systems. Many different aspects of transportation problems could benefit from the introduction of computer graphics tools. Initially, we have been concerned with the design problems that city planners and others face, and the resource allocation and management questions that arise in the creation or expansion of transportation facilities. It is a tremendous saving in time and money for the design engineer to be able to experiment with alternate approaches by computer simulation rather than by actual experiment.

Just as major costs can be avoided by graphics simulation in the planning of a new airport or highway, minor modifications to existing facilities can be accomplished with far greater ease. Here too, the advantages of different approaches can be evaluated carefully ahead of time. The manager can get a clear picture of the effectiveness of various proposals from the simulation, and weigh this against other factors of cost and feasibility. Indeed, he need not wait until he is forced to expand or alter the available facilities before he turns to a computer graphics simulation. He could keep an up-to-date model of his facilities for computer use, and periodically test this model under varying conditions. In this way, problem areas may be diagnosed before they become dangerous or expensive, or both. Computer simulation is obviously superior to actual measurements and experiments in examining future loads on a transportation system. The air traffic control program can simulate anything from private planes to SSTs not yet developed. Of course, the manager could also concentrate on getting the best performance out of the existing facilities. Using the computer graphics method, he can satisfy himself that a certain system of routing and control is optimal before he tries it out. It should be noted that the practical experiences of the people using the graphics system can be continuously applied in a feedback loop to improve the quality of the computer simulation.

Another aspect of a highly interactive graphics system is its suitability for educational use. Traffic engineers can receive a great depth of training from a realistic simulator. Air traffic controllers can learn about many emergency situations and alternate strategies to employ. Watching a dynamic model of a transportation system is an excellent way to learn about its behavior and how to control it effectively. Not only is it a good introduction to a particular situation, but it provides a means of studying subtle problems that may otherwise be impossible to observe. This power comes from the man's ability to control the scale and focus and speed of the simulation interactively, as it proceeds, ignoring routine patterns, and closely examining critical decision areas.

For educational use, design analysis, and practical decision-making, the interactive graphical simulation promises to be a useful tool in the field of transportation.

Real time considerations for an airline

by JOHN LOO, B. T. O'DONALD and I. R. WHITEMAN

Eastern Airlines, Inc.
Miami, Florida

INTRODUCTION

In the 60's the airlines developed and became the recognized leaders of real time applications of the computer. These applications have been primarily commercial in nature and characterized by small amounts of processing on large amounts of data. In the 70's the direction and effort will be expanded to include large scale sophisticated mathematical models within the processing.

In no industry are the problems of scheduling as omnipresent as within transportation and particularly within the airlines. In particular the crew scheduling problem has been solved many times using many varied techniques. It has never, however, been completely solved to the satisfaction of all airlines, and certainly not to the degree of rigor that the term solution would imply to a mathematician. The solutions as they exist today represent a varied collection of heuristic approaches to large problems and rigorous approaches to restricted problems.

The problem is a commercial one and constitutes a real time application yet today the solutions remain cumbersome and slow. The real time response to these large computational problems awaits the development of more suitable heuristics and computers with larger capacities and higher speeds. Large scale combinatorial problems depend upon a step by step analysis and nano second speeds are just too slow for the number of steps involved.

REAL TIME APPLICATIONS

The prime impetus to the real time system has been the passenger reservations system. Today there is no commercial airline that does not live off of this system. It is a random request system accessible from a large number of users located at any of a large number of stations. The response time of the system is measured in terms of an unbroken conversation between the prospective customer and the ticket agent.

Development of the communication peripherals have opened the doors to a number of computer systems which possess obvious import to the airlines. Some of the more important at Eastern Airlines include:

Baggage Tracing—This system is so successful that Eastern Airlines handles this function for some 34 other carriers. This also includes a claims central file geared to detect repetitive and possibly fraudulent claims.

Flight Watch—This system collects and displays to dispatchers the position and movement of every flight.

Crew Management—This system collects and displays the activities and events in the histories of crew members. Of particular interest at any time is their eligibility to fly.

In general the characteristics of these information systems are relatively straightforward. In most cases the information deals with inventory. Is some inventory available? Where is the inventory located? The software is not complicated because of the operations performed upon the raw data, but because of the communications and the security aspects of the data. The consequences associated with loss of files and with not being able to access those files in a fast moving environment are obvious.

Of those real time systems which have come into being some possess structured mathematical models One such is Computer Flight Planning. In producing flight plans, the system takes into consideration the following: (1) altitudes, (2) equipment types, (3) fuel flows, (4) gross weights, (5) Mach numbers, (6) scheduled flight times, and (7) weather data. The vagaries of the weather necessitate real time. Any flight plan which does not meet the performance specifications of the equipment manufacturer is rejected by the computer. In addition, plans which do not allow the flight to arrive on time are rejected unless no plans meet the scheduled time. Under those circumstances those plans

FEDERAL AIR REGULATIONS

- No duty during any rest period
- Do not exceed 30 flying hours in any 7 days
- No more than 8 scheduled hours in any 24 hours without at least 16 hours rest after flying 8 hours
- Deadheading is not considered a rest
- Minimum of 24 hours of consecutive rest during any 7 days

ADDITIONAL FACTORS

- Elapsed time
- Absolute time
- The airport—Rest facilities
- The proximity of airports
- Geography—Customs and clearances
- Connections

PAIRING PAYMENT

- f—Flying time
- d—Duty time
- e—Elapsed time
- t—Tour guarantee
- h—Deadheading

Pairing Payment $= p(f, d, e, t, h)$

Figure 1—The factors of scheduling

which come closest to meeting the scheduled arrival time are selected.

Yet awaiting introduction to the real time system is the crew scheduling problem. The problem is representative of a class of combinatorial problems in which elements of a set are to be ordered or grouped according to some criterion. It is characterized by a large number of possible solutions and is marked by factorial growth in the amount of computation required to carry out that enumeration as problem size increases. The approach taken by most airlines is that of integer programming with 0-1 variables. In contrast, the approach taken by Eastern, the CREATION program is heuristic. The problem is commercially important. The costs associated with flying the flight schedule are difficult to assess. It is necessary to take into account the flight schedule in its entirety. At first thought it would appear that the cost of manning the flights would be a linear combination of the number of flight hours and the unit costs of flying the various types of aircraft. But there is more to it than just that! There are credits which are applicable to the pairings formed and these cannot be determined without completion of the allocation in its entirety. These non-productive costs are appreciable and constitute a major cost of manning the flights.

The flight schedule is constantly changing and at any time there are many flight schedules under investigation. Formulation of the allocation involves specification of the flight schedule and continual liaison between the participating operational groups. The needs and the time responses of each of these participating groups vary at each stage of development and present a formidable real time application.

THE CREW SCHEDULING PROBLEM

The crew schedule is the assemblage of all pairings which satisfy the flight schedule. The pairings are the trips flown by the crews from the time they depart their home base until the time they return home. These pairs must be formulated such that they satisfy governmental regulations and contractual requirements. Collectively the allocation should be executed at least cost to the airline.

There are many factors which must be taken into account in the formation of the pairings as set forth in Figure 1. These are worthy of mention, not because they present an insurmountable challenge, but because together they are indicative of the detail which must be written into the program.

The schedule and time are synonymous. The hours governing the actions of the crew are completely prescribed by the Federal Air Regulations. The principles of safety dictate that the crew not be scheduled for excessively long periods of duty and flying and that the periods of rest are adequate and sufficient. No pairing, no assignment of flying, is acceptable if it does not satisfy these requirements.

Additional requirements are set forth in the contract. The schedule is predicated upon elapsed time, but it is also affected by absolute time. The amounts of permissible duty depend upon the time of day with fewer hours allowable over the evening hours. When periods of rest are prescribed there must be adequate rest facilities available. Some airports possess suitable rooms, some do not. If facilities are not adequate, sufficient additional time must be allotted to travel to proper quarters.

The proximity of some airports, from the viewpoint of scheduling, means they may be treated as one, as co-terminals. It is possible to fly into one airport and depart the other. If the pairing includes such an arrival and departure, sufficient time must be provided to allow for transportation between the airports.

Scheduling depends upon the geography flown. Flights which return from outside of the continental United States must return through customs and an additional time must be provided to permit this clearance. In the case of connecting flights there must be sufficient time provided to make the connection. And so it goes. Each station, each condition, each special facility, each time zone must be uniquely identified within the program in order to assemble a pairing which satisfies all requirements, a legal pairing.

Rules and regulations set forth the requirements governing the formulation of the pairing. There are other contractual specifications which set forth how the crew is to be paid. The costs associated with the allocation of flying depend upon the actual amount of flying and the non-productive time associated with trips flown. At the end of the pairing the pilot has accumulated an amount of flying time and credits of different forms. In a sense, from the time the pilot starts a trip until the time the trip is completed, a number of clocks are kept in terms of these respective credits. These assure him of some minimum amount of flying for the pairing, a credit for the time he is away from home, and if he is required in the course of his duty to fly as a passenger, a credit for deadheading. Pay for the pairing is a complex formulation which involves all of these factors.

These are some of the considerations which must be taken into account in the formation and evaluation of the single pairing. There are yet others which come into play in the formation of the allocation. For the pairings to be manned there must be a sufficient number of trained personnel to fly the scheduled pairings at each of the designated domiciles. In the Eastern system there are currently 6 domiciles. These are the only locations from which crews can be scheduled to fly and these are the 6 cluster areas in which crews live. Each of these domiciles service some number of different types of aircraft, but not necessarily all of them. In general each type of aircraft is serviced by 2 to 5 domiciles. An allocation to be acceptable must have the assignment of flying in consonance with the domicile apportionment.

These then are the considerations which must be taken into account in the formation of an allocation. As arduous and as complex as the associated bookkeeping may be the heart of the problem is to secure that allocation which can be executed at least cost, with a minimum of nonproductive time to the airline.

The size and scope of the crew allocation problem can be seen in the detailed considerations which go into the formulation of the single pairing, in the many ways in which a pairing can be formed, in the balance of manpower requirements among the domiciles, and in the many, many alternative solutions to the final allocation. The number of possible ways of formulation is indeed so large that it is simply not possible to investigate the entire space of solutions.

THE CREATION PROGRAM

The CREATION program is heuristic and assembles allocations through use of controlled Monte Carlo selections. Within the computer emphasis is upon the allocation in its entirety. At computer speeds thousands of allocations are generated and studied to arrive at the least cost solution.

Input specifications

All data required for the CREATION program are specified in the input. No data are included in the program. A typical data set is shown in Figures 2, 3 and 4. The largest portion of the input consists of the flight segments; there must be an entry for every flight flown.

A review of the input shows the detail previously described concerning the formation of the pairings. Information must be provided concerning the equipment, the allocation of manpower between the contributing domiciles, description concerning the coterminals and the stations. The controls governing the computer run must be specified as well as all of the parameters as specified in the contract. The types of output can be elected.

The program

The CREATION program is shown in Figure 5. The program is heuristic and follows the pattern pursued by the equipment specialist who forms the allocation manually. The procedure is a sequential one. Starting with some flight, additional segments are attempted. If the flight departs the previous arrival site and if it is legal it is added to the sequence. If not another is attempted. This process is continued until the trip finally returns home and can then be considered a complete pairing.

Early in the processing of an allocation there are many degrees of freedom available in the formation of a pairing. As the allocation builds up the degrees of freedom decrease until at the end there may be only one way, if any at all, of putting the last pairing together. In the process of putting the allocation together it is necessary to honor the domicile apportionment and to observe all of the legality checks.

The equipment specialist attempting a solution may pursue several courses. Upon completion of an allocation and noting its quality, he may break up some of the pairings and then reassemble in an effort to improve upon the allocation. He may feel that it is worthwhile to preserve some and to recombine others in a

```
                                        RUN DESCRIPTION

RECORD
 CODE

  00     CREATION 1 DYNAMIC TEST PHASE :   TEST NUMBER 1200


                                        EQUIPMENT DATA

RECORD     PRIME         EQUIPMENT ONE    EQUIPMENT TWO    EQUIPMENT THREE
 CODE    EQUIPMENT        TYPE   CODE      TYPE   CODE      TYPE   CODE

  10       DC8            DC8    D8


                                        ALLOCATION DATA

RECORD   EQUIPMENT      CITY/STA ALLOC  CITY/STA ALLOC  CITY/STA ALLOC  CITY/STA ALLOC  CITY/STA ALLOC  CITY/STA ALLOC
 CODE      TYPE           NAME     %      NAME     %      NAME     %      NAME     %      NAME     %      NAME     %

  20       DC8            NYC    040.     FLA    060.


                                        CO-TERMINAL DATA

RECORD   CITY         CO-TERMINALS               A TO B                    A TO C                    B TO C
 CODE    NAME      STA A  STA B  STA C    DUTY    D/H     TRAN     DUTY    D/H     TRAN     DUTY    D/H     TRAN

  30      CHI      ORD    MDW             01:45   00:18   02:00
  30      FLA      MIA    FLL             01:00   00:19   01:30
  30      NYC      JFK    EWR    LGA      01:30   00:16   02:30    00:45   00:12   01:30    01:15   00:14   02:00
  30      WAS      DCA    BAL    IAD      01:15   00:20   02:00    01:00   00:20   01:30    01:45   00:40   03:30


                                        STATION DATA

RECORD   STATION    CONVERSION         EXTENSION TO       INTL     PRE-CLEARANCE     CUSTOMS
 CODE     NAME     FACTOR FOR GMT      BREAK TIME         CODE         TIME            TIME

  40      ABE        + 05:00             00 00                        00 00           00 00
  40      ABY        + 05:00             00 00                        00 00           00 00
  40      ACA        + 06:00             00 00              1         00 15           00 15
  40      AGS        + 05:00             00 00                        00 00           00 00
  40      ATL        + 05:00             00 00                        00 00           00 00
  40      AVP        + 05:00             00 00                        00 00           00 00
  40      BAL        + 05:00             00 00                        00 00           00 00
  40      BDA        + 04:00             00 00              1         00 00           00 15
```

Figure 2—Input audit listing

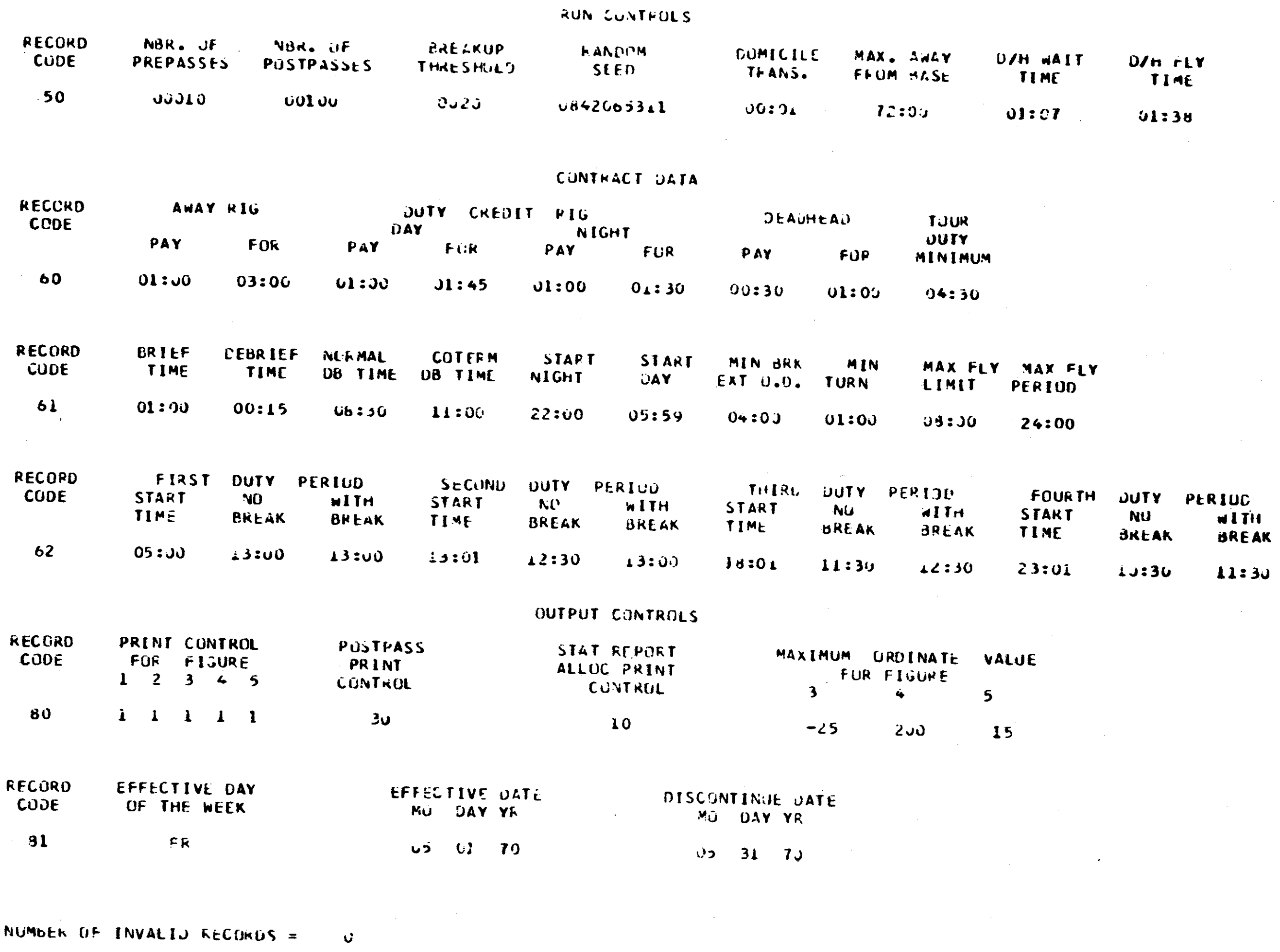
more favorable fashion. The effects can not be determined without trial. He can pursue this action time and time again. After some succession of such attempts the equipment specialist may feel that the overall combination has been exhausted. He may elect to start the allocation from scratch if he feels that there is little to be gained through minor perturbations of the originating allocation.

The equivalences of all of these actions can be seen in the flow chart. Prepass indicates the formation of an allocation from scratch. Postpass indicates the reduction of the previous allocation in accordance with prescribed selection criteria. All pairing is based upon random selection from the remaining unassigned flights.

As each allocation is generated, its Figure Of Merit is compared with those of the allocations previously generated. If it is not considered to be a good candidate, it is discarded. If it is considered to be a good candidate, it is saved and the worst of those previously saved is discarded. Throughout the processing some fixed number of allocations is preserved representing the best of the allocations generated. Final selection of the best allocation is made from that reservoir.

The outputs

There are a number of different outputs available from the CREATION program. The basic output is,



```
                                              RUN CONTROLS
RECORD   NBR. OF     NBR. OF      BREAKUP      RANDOM       DOMICILE  MAX. AWAY  D/H WAIT  D/H FLY
 CODE   PREPASSES  POSTPASSES    THRESHOLD      SEED         TRANS.   FROM BASE    TIME      TIME

  50      00010       00100        0025       0842065311     00:01      72:00     01:07     01:38

                                              CONTRACT DATA
RECORD       AWAY RIG               DUTY  CREDIT  RIG                  DEADHEAD        TOUR
CODE                               DAY                 NIGHT                            DUTY
        PAY        FOR        PAY        FOR        PAY        FOR      PAY      FOR    MINIMUM

 60     01:00      03:00      01:00      01:45      01:00      01:30    00:30    01:00   04:30

RECORD  BRIEF   DEBRIEF   NORMAL    COTERM     START     START   MIN BRK    MIN    MAX FLY  MAX FLY
CODE    TIME     TIME    DB TIME   DB TIME    NIGHT       DAY   EXT D.D.   TURN    LIMIT    PERIOD

 61     01:00    00:15    06:30     11:00     22:00      05:59   04:00     01:00   08:00    24:00

RECORD    FIRST   DUTY   PERIOD     SECOND   DUTY   PERIOD     THIRD   DUTY   PERIOD     FOURTH   DUTY   PERIOD
CODE    START     NO      WITH     START     NO      WITH     START     NO      WITH     START     NO      WITH
        TIME     BREAK   BREAK     TIME     BREAK   BREAK     TIME     BREAK   BREAK     TIME     BREAK   BREAK

 62     05:00    13:00   13:00     13:01    12:30   13:00     18:01    11:30   12:30     23:01    10:30   11:30

                                             OUTPUT CONTROLS
RECORD   PRINT CONTROL        POSTPASS           STAT REPORT          MAXIMUM  ORDINATE  VALUE
CODE      FOR  FIGURE           PRINT            ALLOC PRINT                 FOR FIGURE
          1  2  3  4  5        CONTROL             CONTROL            3           4          5

 80       1  1  1  1  1          30                  10              -25         200        15

RECORD   EFFECTIVE DAY            EFFECTIVE DATE           DISCONTINUE DATE
CODE     OF THE WEEK                MO  DAY YR                 MO  DAY YR

 81          FR                     05  01  70                 05  31  70


NUMBER OF INVALID RECORDS =     0
```

Figure 3—Input audit listing

of course, the best allocation. The other outputs are optional and relate to the characteristics of all allocations generated within the run. And, of course, there are numbers of various sorts for the convenience of the concerned operating groups.

In Figure 6 is shown a sample printout of a portion of an allocation. All pairings are represented in this common format. Basically the information consists of an identification, a listing of all segments which make up the pairing, departure and arrival data, and a breakout of the respective times and credits. The total pay time and the specific credit, if there is one, are shown.

A typical pairing is ALLNO=1 PR NO=12. This pairing departs JFK and is completed in the co-terminal EWR. This pairing extends over two days with a duty break in MIA. The amount of flying is 13:32. Note that the time away is 42:00. There is an away credit of 1 hour of flying for 3 hours away. This indicates a payment of 14:00, and hence the credit of 28 minutes. In Figure 7 is shown a summary sheet of the pertinent statistics governing the best allocations.

Timing considerations

The CREATION program runs on an IBM 360/65. The amount of time required to generate an allocation depends upon the size and characteristics of the fleet. It can be noted that the number of pairings in an allocation may be in the hundreds and the number of allocations attempted in an effort to secure a feasible least cost solution may be in the thousands. A thorough study can involve hours of machine time.

```
 CARD  EQUIPMENT      DEPARTING          ARRIVING        FLIGHT     NUMBER      FLIGHT     TURNS TO   LOCAL
 UMBER   TYPE       STATION  TIME     STATION  TIME      NUMBER    OF STOPS     TIME       FLIGHT    DEP-MIN

  14     DC8          ATL     600       CLT     646        88          0          46          83       360
  27     DC8          ATL    1825       EWR    2128        82          1         216         917      1105
  63       INTERMEDIATE STOPS      1928 1-RDU   2015
  38     DC8          ATL    1000       LAX    1116        83          0         416          84       600
  34     DC8          ATL    1845       LAX    2001        87          0         416          88      1125
  29     DC8          ATL    2055       LAX    2211        89          0         416          82      1255
  36     DC8          ATL    2055       MCO    2211        84          0         116         699      1255
  19     DC8          ATL    1405       MCO    1520       125          0         115          87       845
  32     DC8          ATL    2055       MIA    2233       245          0         138          12      1255
   2     DC6          ATL    2055       MIA    2233      1245          0         138        8888      1255
  50     DC8          BDA    1420       JFK    1520       810          0         200          27       860
  26     DC8          BDL    2320       JFK       4       527          0          44          33      1400
  57     DC6          BDL     900       SJU    1504       947          1         425         924       540
  66       INTERMEDIATE STOPS      1001 1-BAL   1040
  37     DC8          CLT     800       ATL     901        83          0         101          83       480
  58     DC8          DTW    1100       MIA    1341       953          0         241         953       660
  54     DC8          DTW    1100       MIA    1341      1953          0         241        1953       660
  28     DC8          EWR    1630       ATL    1958        89          1         238          89       990
  64       INTERMEDIATE STOPS      1750 1-RDU   1840
   3     DC8          EWR     830       MIA    1055         7          0         225           6       510
  46     DC8          EWR    2330       SJU     403       917          0         333         916      1410
  60     DC8          EWR    1215       SJU    1651       967          0         336         946       735
  51     DC8          JFK    1000       BDA    1256       807          0         156         810       600
   1     DC8          JFK     720       BDL     800       528          0          40         947       440
  12     DC8          JFK    1400       MIA    1635        21          0         235          28       840
  16     DC8          JFK    1700       MIA    1935        27          0         235         410      1020
  24     DC8          JFK    1100       MIA    1335        33          0         235          24       660
  10     DC8          JFK    2100       MIA    2335       401          0         235          44      1260
  44     DC8          JFK     730       SJU    1156       923          0         326         922       450
  41     DC8          JFK    1830       SJU    2309       927          0         339         923      1110
  52     DC8          JFK    1030       SJU    1508       985          0         338         952       630
  22     DC8          JFK    2035       YUL    2128       924          0         123         985      1205
  15     DC8          LAX    1020       ATL    1716        82          0         356          82       620
  35     DC8          LAX    1300       ATL    1956        84          0         356          84       780
   9     DC8          LAX    2200       ATL     500        88          0         400          83      1320
  33     DC8          MCO    1628       ATL    1745        87          0         117          87       988
  20     DC8          MCO    2250       MIA    2334       699          0          44         915      1370
  31     DC8          MIA    1148       ATL    1259       246          0         111        8888       708
  40     DC8          MIA    1855       DTW    2137       952          0         242         953      1135
  53     DC8          MIA    1855       DTW    2137      1952          0         242        1953      1135
  17     DC8          MIA    1230       EWR    1450         6          0         220          89       750
  11     DC8          MIA    1000       JFK    1225        12          0         225          21       600
  21     DC8          MIA    1600       JFK    1825        24          0         225         401       960
```

Figure 4—A listing of accepted segment records

THE CONTRIBUTIONS AND TIME RESPONSES OF THE CREATION SYSTEM

There are a number of distinct types of requests that must be satisfied by the CREATION program from the time a flight schedule is initiated until the time that the flight schedule is finally implemented. The detail of information and accuracy of information, and the time response in which the response must be made, continually change over the course of the development of the flight schedule. These changes are dictated by the degree of interaction between the participating groups concerned with formulation of the flight schedule.

During the course of development there are four primary groups concerned with production of the ultimate schedule. Fundamental responsibility falls within the Flight Schedule group; theirs is the need of formulating the best schedule from the viewpoint of the traveling public and the operations of the airlines. And for the flights to be flown there must be an airplane available for every scheduled flight; these considera-

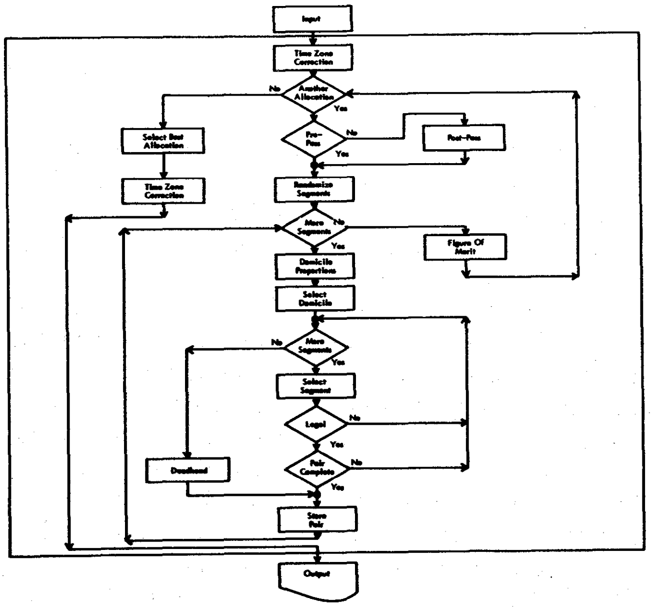


Figure 5—Creation flow chart

```
                                  NYC  EFF 05/01/70 THRU 05/31/70

                                  LOCAL         SCHED        DAILY             SCHED            AWAY           DUTY C
                     CR           TIME      S    SEG   ADJ    F.T.     LAY      ON     DUTY     FROM   AWAY    TOUR C  ROUTE
FREQ     EQUIP FLT   ML   FR  TO  DEP   ARR T   F.T.   NITE SEQ TOT   OVER     DUTY   CREDIT    BASE  CREDIT CREDIT R  QUAL

         SEQ NR           ALLNO =   1  PR NO =  11

         D801 0027   D    JFK MIA 1700 1935 0   2 35                  2 25                                              JFK MIA
         D801 1410        MIA PHL 2200 0015 0   2 15          4 50   18 05     8 30    5 06                             MIA PHL
         D801 1043        PHL MIA 1820 2035 0   2 15                  1 25
         D801 0400        MIA JFK 2200 0025 0   2 25   1 23   4 40             7 20    4 40
                                                                                                32 40
           LINES                                       1 23   9 30                     9 46           10 53 TOT
           FLOAT DATES                                 12.7%                                           1 23 CR
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
         SEQ NR           ALLNO =   1  PR NO =  12

         D801 0528        JFK BDL 0720 0800 0     40                  1 00                                              JFK BDL
         D801 0947   L    BDL SJU 0900 1504 1   4 25                  1 36                                              BDL SJU
         D801 0952        SJU MIA 1640 1805 0   2 25          7 30   20 50    12 00    7 30                             SJU MIA
         D801 0953   D    MIA SJU 1456 1817 0   2 22                  1 38
         D801 0960   D    SJU EWR 1955 2235 0   3 40     28   6 02            10 25    6 10                             SJU EWR
           COTERMINAL                                                          1 30
                                                                                                42 00
           LINES                                         28  13 32                    13 40           14 00 TOT
           FLOAT DATES                                  3.3%                                             28 CR
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
         SEQ NR           ALLNO =   1  PR NO =  13

         D801 0927   DS   JFK SJU 1830 2309 0   3 39                  1 21                                              JFK SJU
         D801 0928        SJU JFK 0030 0301 0   3 31          7 10             9 46    7 10
                                                                                                 9 46
           LINES                                              7 10   TOT               7 10
           FLOAT DATES                                    %
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
         SEQ NR           ALLNO =   1  PR NO =  14

         D801 0985   L    JFK SJU 1030 1508 0   3 38                  1 22                                              JFK SJU
         D801 0924   D    SJU JFK 1630 1913 0   3 43          7 21             9 58    7 21
                                                                                                 9 58
           LINES                                              7 21   TOT               7 21
           FLOAT DATES                                    %
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
         SEQ NR           ALLNO =   1  PR NO =  15

         D801 0007   B    EWR MIA 0830 1055 0   2 25                  1 35                                              EWR MIA
         D801 0006   L    MIA EWR 1230 1450 0   2 20          4 45             7 35    4 45
                                                                                                 7 35
           LINES                                              4 45   TOT               4 45
           FLOAT DATES                                    %
```

Figure 6—Bid information sheet

tions are treated by the non-manpower groups. On the manpower side, there is the allocations group concerned with the development of the most efficient allocation. And there is the manpower group with the responsibility of assuring that there is sufficient trained manpower available to fly the flights.

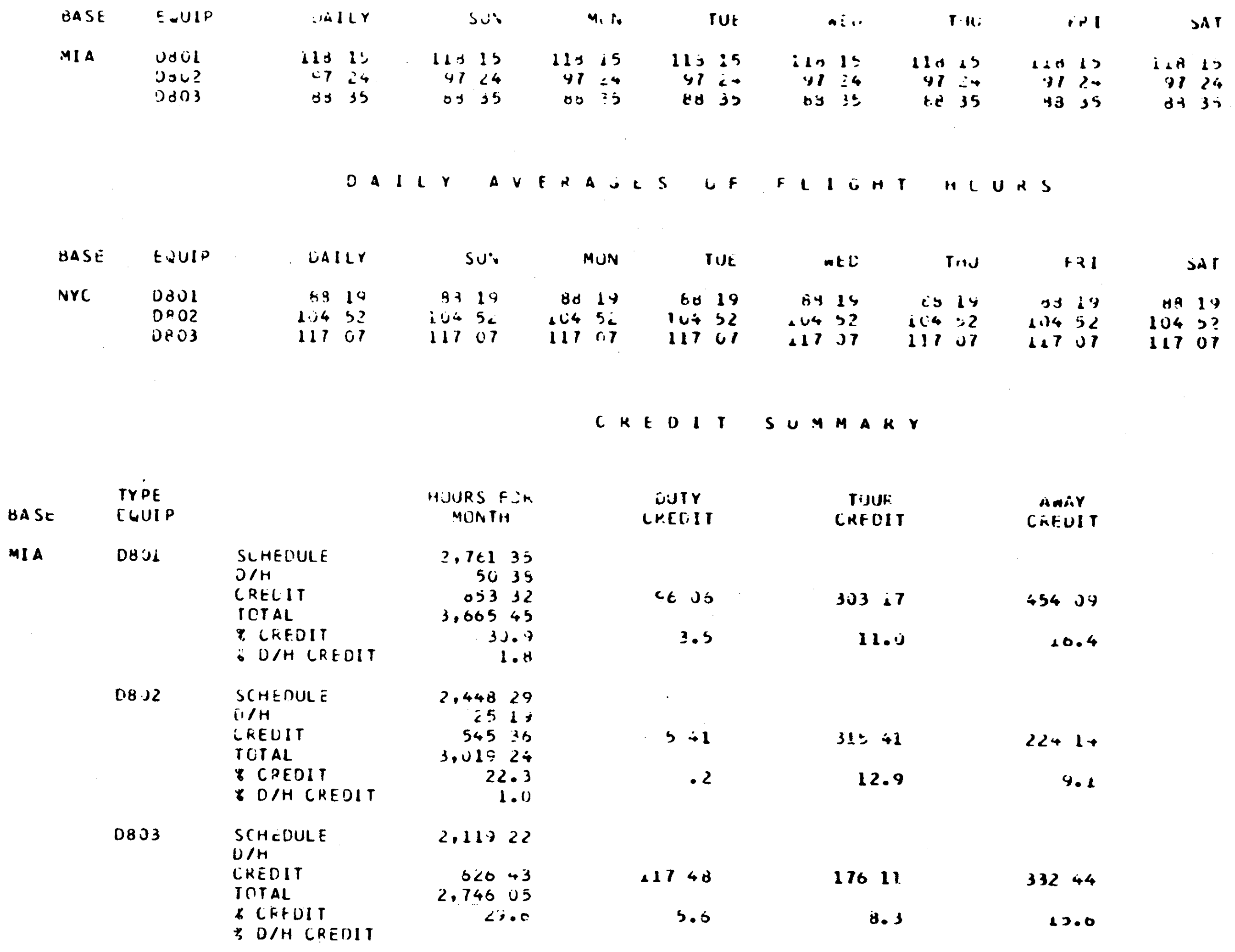
There is a great deal of interaction among these participating groups. Each would be pleased if the schedule could be altered to accommodate his own specific needs. Each interact with each other through the flight schedule, and through additional interactions as in the case of manpower requirements defined in the domicile apportionment.

The total cost of manpower depends upon training and transfers of crews at the respective domiciles and the costs of credits. Both of these steady state and transient costs are related to domicile apportionment. Each change to the flight schedule can produce far reaching consequences to the participating groups and may require a complete analysis of the allocation process.

The allocation can be sensitive to the alteration of but one minute in one flight. Of course, it is possible that a flight can be advanced or delayed a number of minutes and that no change at all will result. But it is also possible that the change may result in the breaking of a pair which will result in the breaking of another until every pair is affected and altered.

The closeness of the relationship of the quality of the flight schedule to the allocation, to the non-manpower requirements, and to the manpower requirements means that each step along the way in the formulation of a new schedule different requirements for information are created which must be satisfied within some permissible time response.

Figure 8 shows a simplified flow chart of the stages



```
BASE    EQUIP         DAILY        SUN        MON        TUE        WED        THU        FRI        SAT

MIA     D801         118 15     118 15     118 15     118 15     118 15     118 15     118 15     118 15
        D802          97 24      97 24      97 24      97 24      97 24      97 24      97 24      97 24
        D803          88 35      88 35      88 35      88 35      88 35      88 35      88 35      88 35

                      D A I L Y   A V E R A G E S   O F   F L I G H T   H O U R S

BASE    EQUIP         DAILY        SUN        MON        TUE        WED        THU        FRI        SAT

NYC     D801          88 19      88 19      88 19      88 19      88 19      88 19      88 19      88 19
        D802         104 52     104 52     104 52     104 52     104 52     104 52     104 52     104 52
        D803         117 07     117 07     117 07     117 07     117 07     117 07     117 07     117 07

                                         C R E D I T   S U M M A R Y

        TYPE                              HOURS FOR       DUTY          TOUR          AWAY
BASE    EQUIP                              MONTH         CREDIT        CREDIT        CREDIT

MIA     D801       SCHEDULE             2,761 35
                   D/H                     50 35
                   CREDIT                 653 32         96 06         303 17        454 09
                   TOTAL                3,665 45
                   % CREDIT                30.9           3.5           11.0          16.4
                   % D/H CREDIT             1.8

        D802       SCHEDULE             2,448 29
                   D/H                     25 19
                   CREDIT                 545 36          5 41         315 41        224 14
                   TOTAL                3,019 24
                   % CREDIT                22.3            .2           12.9           9.1
                   % D/H CREDIT             1.0

        D803       SCHEDULE             2,119 22
                   D/H
                   CREDIT                 626 43        117 48         176 11        332 44
                   TOTAL                2,746 05
                   % CREDIT                29.6           5.6            8.3          15.6
                   % D/H CREDIT
```

Figure 7—Summary of flight hours

taken in the formulation of a new schedule. During all of these stages the Allocations group makes some contribution to the formulation and ultimately it generates the final allocation which becomes the basis for assignment of crews to the pairings.

The first step is shown in Stage 1, in which scheduled development originates at the uppermost levels of the planning division. The principal concern at this time is to provide an air transportation service pattern which meets the needs of the traveling and shipping public. Some of the detailed considerations are certificate requirements, market research forecasts, political and civic pressures, competitors' schedules, equipment availability and support capacity; i.e., terminal and ground handling facilities. Considerations must be given to aircraft procurement and to the size of the pilot pool.

During this stage the schedule is but loosely defined. What is known is the number of flights, where they fly, the number of hours to be flown, and some idea of the equipments to be flown. Less well defined is the departure time to the hour and the minute and the arrival time to the hour and the minute. Yet in the allocation process it is precisely this detailed information that is required.

At this stage of development the requirements imposed upon the Allocations group are for estimates concerning the approximate amounts of non-productive time generated by the proposed schedule. Generation of the initial allocation is more a matter of extrapolating from overall statistical characteristics as opposed to the generation from the detailed flight specifications. Consistent with the availability of schedule decisions, the need for information response may be measured in weeks or months.

In Stage 2 the objectives of the airline have been reasonably formulated in terms of the flights, the number of equipments flown and the numbers of pilots. It is at this stage that work proceeds to put together a schedule which satisfies these objectives. At this time the emphasis is upon a real detailed schedule, one with realistic departure times and arrival times flown by specific aircraft.

The objective at this time is a real schedule which can be released to the operational groups approximately 6 months prior to the effective date of the schedule. Prior to the release of this advance schedule, whatever is formulated is extremely fluid, and little reliance can be placed upon any of the portions which make up the preliminary estimates. Without some degree of solidification, there is little that can be accomplished definitively by the Allocations group. Again participation by the Allocations group is confined to generalized projections based upon the best though insufficient information available. The time frame for response by the group is not unlike that of Stage 1 and is measured in terms of weeks and months.

Stages 3 and 4 mark the availability of the advance schedule. For the first time the operational groups have made available to them a reasonably fixed schedule that they consider from a detailed point of view. The two stages are marked by the release for publication date. When this time is reached the schedule is deemed fixed. It is during these two stages that the contributions of the Allocations group become most meaningful though in different ways.

In Stage 3 the operational groups examine every detail of each flight in terms of the specific demands imposed upon them. During this time the pace of interchange of information between all of the operational groups and the flight schedulers increases dramatically. There is a constant demand for minute changes to the schedule to bring about important efficiencies to specific groups. Tentative changes must be released as soon as possible to determine if there are adverse effects upon the other participating groups. The result is that changes are released daily.

It is during Stage 3 that the need for detailed information from the Allocations group increases immeasurably. It is at this time that the final schedule begins to take shape and begins to solidify. Each change to the flight schedule must be judged in terms of how it affects the allocation. The evaluation cannot be made in terms of the flights considered by themselves. It is

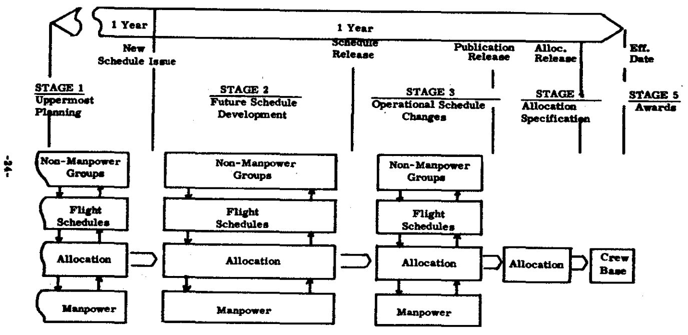


Figure 8—Flight schedule development

possible that a change may affect the allocation. But it also may be such that it will break a pairing and that the effect will cascade through the allocation in its entirety. With changes taking place daily it is essential that it be possible to generate the allocation daily.

During this Stage 3 the need of the Allocations group is not merely to respond to the demands and changes imposed by the other operational groups. The group itself is the source of changes to the schedule. The schedule must be evaluated in terms of how changes can be secured which can bring about important reductions to the allocation. It is perhaps at this stage that the most important economies can be introduced to the allocation through changes made to the flight schedule.

It is during this stage that the tempo of response increases tremendously and the time frame of management response changes from weeks to a day to day response. It is this stage that makes mandatory computer speeds to permit response in the time frame of management. It is during this stage that there is need for numbers of short bursts upon the computer to evaluate the constantly changing schedule. The emphasis is upon accurate assessments of the effect of the individual changes upon the allocation. The need is for rapid evaluation of not only what is good, but the detection of what is bad in time to permit corrective action in reduction of the overall cost of the final emerging allocation.

Once Stage 3 is closed, as the schedule is released to publication approximately 3 months before the effective date, there is little that can be done to introduce changes to the schedule. At this time the viewpoint of the Allocations group again takes on a new outlook. The objective is to secure the best allocation possible for the now fixed schedule.

In Stage 4 the time response is not so short. The schedule is relatively fixed and the Allocations group is not deluged with a constant flow of changes. Whereas the emphasis in the preceding stage was to identify pairings which were poor, and to introduce changes to the schedule to ameliorate these bad conditions, the emphasis now is to secure the very best allocation consistent with the now fixed schedule. It is during this stage the final allocation is formulated and comes into being. Each and every pairing is completely specified.

In Stage 5 the allocation is released to the crew bases. At these locations the actual assignments of the specific crews to fly the pairings are made. During this stage the schedule and its implementation is firmly committed and there is little further participation required from the Allocations group.

THE FUTURE

A problem basic to the airlines is the generation of schedules. Flight schedules are in a constant state of flux to accommodate the changing patterns of passenger traffic, the changes of seasons, and the acquisition and retirement of equipment. The problem is compounded due to schedules being predicated upon schedules. The schedules that the crews fly depend upon the flight schedule, the schedules that the equipment fly depend upon the flight schedule, and both crew and equipment schedules depend upon each other.

At present every major carrier is attempting to solve the crew scheduling problem. At best, satisfactory and sufficient solutions are 2 to 3 years away. There are other problems which are composites of the allocation. These involve domicile apportionment and contractual studies. Whereas the generation of the allocation involves hours of machine time these problems involve shifts of machine time. Inclusion of these problem sets into the real time network requires better problem solving capability. The difficulty is not in the processing of information from terminal to computer and from computer to terminal. The bind is within the computer itself.

What is required are computers with larger capacities and higher speeds complemented by the development of better heuristics. Combinatorial problems confined to step by step solutions are so large that increases in speed and capacity alone are not sufficient. Algorithms must be developed which cut across customary methods of solution. Until that time the management information real time system will be sluggish at best.

A computer simulation model of train operations in CTC territory

by DAVID T. BORCH

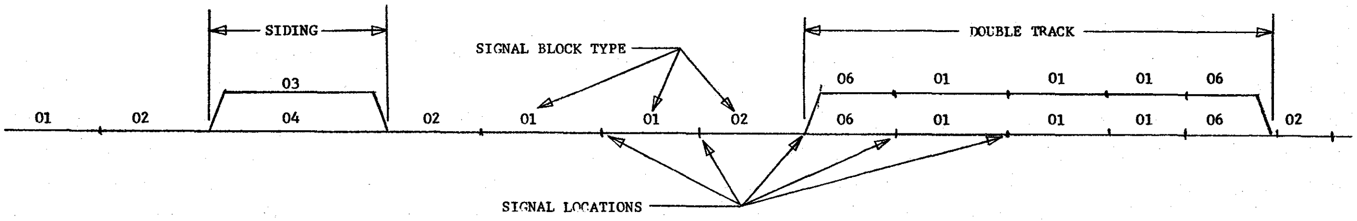
CP Rail
Montreal, Quebec, Canada

CENTRALIZED TRAFFIC CONTROL (CTC) is a system which controls the movement of trains. One train dispatcher, located at a central point, can manipulate the position of track switches and the indication of track signals through the use of a series of push buttons. It is the position of the track switches and the signal indications which govern the progress of the trains through the CTC system. A schematic of a section of single track with a siding and a section of double track is shown in Exhibit I. The CTC Simulation is a model of the train movements through a CTC system.

The design of the computer program commenced in 1961, and up to its latest application was still being fine-tuned. The simulation was programmed for an IBM 7080 computer in Autocoder language with one Fortran subroutine. The running time varies with the number of trains and the length of territory being simulated. The computer running time to produce the attached output was approximately twenty minutes.

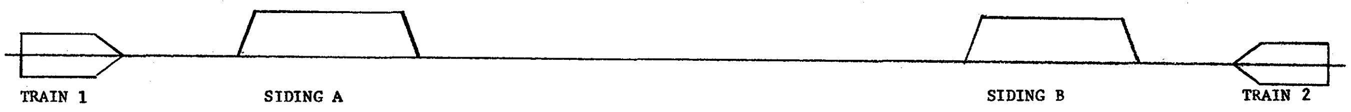
The input to this program is comprised of three types:

1. Track Profile Record—for a section of track up to one mile in length, this record contains the resistance due to grade and curvature (in lbs./tons) in each direction and the track speed limit for freight and passenger trains in each direction,
2. Signal Block Record—contains the mileage of the signal at the beginning of the signal block, the length of the signal block, the mileage of spur tracks, and a code describing the type of the signal block,
3. Train Data Records—(a) contains the number of diesel locomotives, the dispatching priority code, the departure mileage and time, the arrival



A SIGNAL BLOCK is the section of track between two signals

Exhibit I



Delay to TRAIN 1 if meet takes place at siding A = 15 minutes

Delay to TRAIN 2 if meet takes place at siding B = 18 minutes

Exhibit II

mileage, the gross weight of the train, the horsepower, and the schedule or standard running time of the train over the territory being simulated, (b) contains the scheduled, or work stop, delay information, such as mileage and duration of the delay, any change in the consist of the train (adding or deleting cars and diesel locomotives), and whether the delay is clearing or non-clearing.

The inputs for the Track Profile and the Signal Block Records are obtained from Engineering drawings. The input for the Train Data Records is obtained from dispatcher train sheets, existing train schedules, and Transportation planning personnel.

Appendix I contains the generalized logic diagram of the simulation. After the Track Profile and Signal Block Records have been edited and stored in the computer memory, a reservoir of the first sixteen trains, in chronological order, is created. Throughout the simulation, this reservoir of trains is scanned, and the train with the lowest clock time is selected and progressed through the system to its next event or decision point. As the trains move through the system, the information describing each train is actually moved from signal block to signal block in the computer memory. The simulator treats each signal location as a decision or event point, where the relative position of each train can be evaluated in order to determine whether they should stop, proceed, slow down, or enter a siding.

If the selected train is originating, or returning from a clearing work stop, the simulator must decide whether to allow the train to enter the system. In real life, a train dispatcher must scan the track in the vicinity of the train to determine if the track is clear before deciding whether to permit the train to enter the system. This is precisely what the simulator does. If the track is clear, the simulator places the train in the signal block and calculates the time it will take the train to move to the next signal location. This time is then added to the train clock time and the train returned to the reservoir. If the track is not clear, the train is not placed in the signal block. One minute is added to the train clock time before the train is returned to the reservoir.

When the selected train is one that has arrived at its terminating point, a statistical record is created for that train and the train is removed from the system. The signals for that signal block are reset to clear. A new train is then added to the reservoir.

When the selected train is one that has arrived at a scheduled or work stop delay point, the duration of the train's delay is added to the train clock time. A work stop may be of two types—clearing and non-clearing. A clearing work stop is one in which the train leaves the CTC system, whereas a non-clearing work stop is one in which the train remains in the CTC system. Therefore, if the train has a clearing work stop, the simulator removes the train from the CTC system, cancels all meets and passes for that train, resets to clear the signals for the signal block the train is in, and returns the train to the reservoir. If the train has a non-clearing work stop, the simulator returns the train to the reservoir.

Before moving a train to the end of a signal block, the

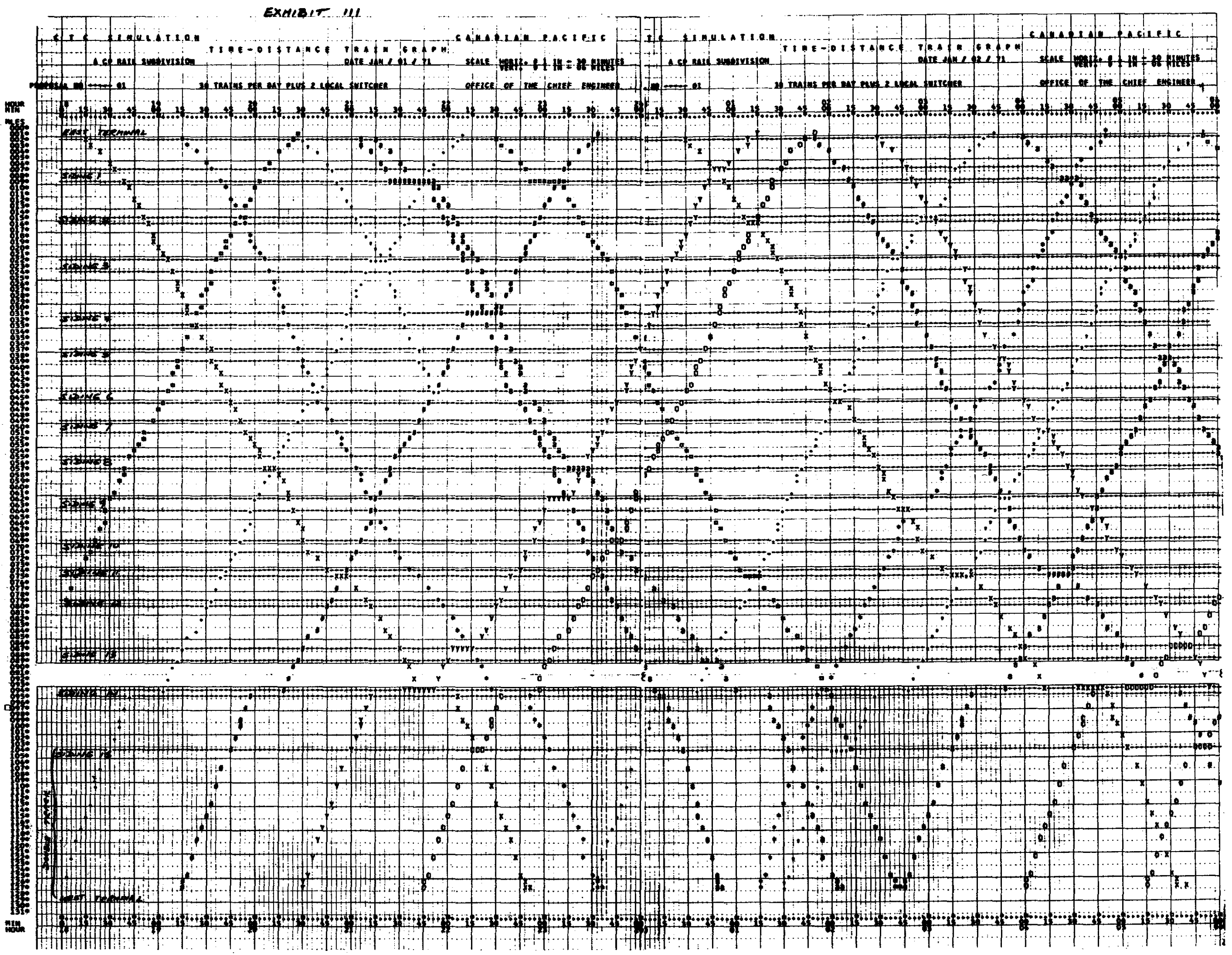


Exhibit III

simulator must first determine if the train terminates in this signal block or has a work stop in the signal block. If either of the above is the case, the time for the train to move to the terminating or work stop point from its present location is calculated. Otherwise, the time for the train to move to the next signal location from its present location is calculated. Appendix II is the logic diagram for the routine which calculates the train running time over a specified distance. The dynamic characteristics of the train are developed by the Newtonian force equation $F = MA$, where F, the net force available for acceleration, and M, the mass of the train, are known and A, the acceleration, is calculated. The net force is the algebraic sum of the forces acting on the train, that is, the 'pulling' force or 'tractive effort' of the diesel locomotives minus the 'retarding' forces or the train resistance due to grade, curvature, flange and bearing friction, and still air resistance. The train is progressed in small increments of distance of between 100 and 260 feet until the train has travelled the required distance. The times for the train to move over all the small distance increments are summed and added to the train clock time. The train is then returned to tbe reservoir.

When the train has reached the end of a signal block, and hence, is about to enter the next signal block, the simulator must first evaluate the position of the train relative to other trains in the vicinity of this train. All

```
                                              EXHIBIT IVa                                             2
*        C T C   S I M C L A T I C N                                       C A N A C I A N   P A C I F I C

                                     S T A T I S T I C A L   R E P C R T

               A CF RAIL SUBCIVISION                                       PERIOC JAN  1 TO JAN  3 , 1971

   PRCPCSAL AC ----- 01               38 TRAINS PER CAY PLUS 2 LCCAL SWITCHER         OFFICE  OF  THE  CHIEF  ENGINEER

                                  I  PERFORPANCE  SUPPARY  OF  INDIVICLAL  TRAINS

  T R A I N      NC  DEPARTURE  POWER  NC    TONS    ELAPSEC TIPE   TIPE     D E L A Y   T I M E     NO  AVG
 NB SPB TYP CLS  CF    TIPE      IN    OF            RUN CC RUN 01  SAVED   WK STP CNFLCT  TOTAL    OF  SPD
                DSLS DY HR PIN   H/P  CARS           HR MIN HR MIN  HR PIN  HR PIN HR PIN  HR MIN CNFLCT

 1C10 ¤  F  E  08   01 16 30  24000  114  04695   4 3C   3 57    33    15     02     17   2  31.8
 1C91 ,  F  I  04   01 20 30  06800  100  0360C   1 3C   1 25    05                         1  30.3
 2C02 .  F  E  08   01 18 00  24000  114  04695   4 3C   4 10    20    15     14     29   5  30.1
 1C11 )  F  C  06   01 18 15  18000  114  15990   4 45   4 37    08    15     15     30   4  27.2
 2C04 #  F  E  08   01 19 15  24000  114  04695   4 3C   4 10    12    15     22     37   5  29.2
 1C13 *  F  C  06   01 19 15  18000  114  1599C   4 45   4 22    23    15            15   4  28.8
 1C01 8  P  A  03   01 21 05  05000  012  01200   3 4C   3 47    07-   02     26     28   4  33.2
 2FE2 Y  F  F  08   01 20 30  24000  114  04695   4 3C   4 45    15-   15     47   1 02   5  26.4
 2FE4 C  F  F  08   01 21 45  24000  114  04695   4 3C   4 05    25    15     10     25   5  30.8
 1C15 a  F  C  06   01 21 15  18000  114  1599C   4 45   4 52    07-   15     31     46   5  25.8
 1EW3 ¤  F  F  02   01 22 30  06000  079  02365   4 3C   4 13    17           28     28   5  29.8
 2C06 .  F  E  08   01 23 30  24000  114  04695   4 3C   4 11    19    15     13     28   5  30.0
 2C02 *  P  A  03   02 01 15  05000  012  01200   3 4C   3 37    03    11     04     15   4  34.7
 1GB1 X  F  D  06   02 00 30  18000  114  15990   4 45   5 08    23-   15     46   1 01   5  24.5
 2C08 ,  F  E  08   02 01 30  24000  114  04695   4 3C   4 20    10    15     23     38   4  29.0
 1C17 #  F  C  06   02 01 30  18000  114  1599C   4 45   4 54    09-   15     32     47   5  25.6
 2C1C 8  F  E  08   02 02 45  24000  114  04695   4 3C   3 56    34    15            15   5  31.9
 1GB3 Y  F  D  04   02 02 30  12000  114  10452   4 45   4 34    11    15     14     29   5  27.5
 1FB3 2  F  D  06   02 04 00  18000  114  15990   4 45   4 37    08    15     15     30   5  27.2
 2FE6 C  F  F  08   02 04 00  24000  114  04695   4 3C   4 57    27-   15   1 00   1 15   5  25.4
 2GE2 0  F  F  08   02 05 15  24000  114  04695   4 3C   4 10    20    15     14     29   5  30.1
 2C01 ¤  F  C  06   02 05 40  18000  114  1599C   4 45   4 58    13-   15     37     52   5  25.3
 1FB5 X  F  D  06   02 06 40  18000  114  15990   4 45   4 23    22    15     02     17   4  28.7
 2C12 .  F  E  08   02 06 30  24000  114  04695   4 3C   4 35    05-   15     37     52   5  27.4
 2C90 Y  F  I  04   02 09 15  06800  1C0  0635C   1 3C   2 01    31-          09     09   3  21.3
 2902 *  F  B  06   02 07 45  18000  114  07725   4 15   4 20    05-   15     15     30   6  29.0
 2965 #  F  B  03   02 08 15  09000  114  0715C   4 0C   4 28    28-   15     20     35   6  28.1
 2GE4 ,  F  F  06   02 09 00  18000  114  04305   4 3C   4 45    15-   15     48   1 03   5  26.4
 2948 C  F  B  06   02 11 00  18000  114  07725   4 15   4 05    10    15            15   5  30.8
 2C03 8  F  C  06   02 09 30  18000  114  15990   4 45   5 35    50-   15   1 13   1 28   6  22.5
 2C05 &  F  C  06   02 10 20  18000  114  15990   4 45   5 00    15-   15     37     52   7  25.1
 2C14 ¤  F  E  08   02 11 30  24000  114  04695   4 3C   4 17    13    15     21     36   6  29.3
 2C07 0  F  C  06   02 11 20  18000  114  1599C   4 45   4 39    06    15     18     33   5  27.0
 2EB2 X  F  F  02   02 13 15  06000  1C8  0309C   4 30   4 20    10           25     25   5  29.0
 2C09 .  F  C  06   02 13 00  18000  114  1599C   4 45   5 08    23-   15     44     59   6  24.5
 2EW1 *  F  F  03   02 13 40  09000  114  04507   4 3C   4 37    07-          36     36   5  27.2
 2LU2 #  F  B  06   02 14 15  18000  085  10095   4 15   4 44    29-   15     31     46   6  26.5
 2C16 Y  F  E  08   02 15 15  24000  114  04695   4 3C   4 12    18    15     15     30   5  29.9
 2901 ,  F  B  03   02 14 50  09000  114  0715C   4 0C   4 48    48-   15     39     54   5  26.2
 2018 8  F  E  08   02 16 30  24000  114  04695   4 3C   4 16    14    15     20     35   4  29.4


   PROPCSAL AC ----- 01               38 TRAINS PER DAY PLUS 2 LOCAL SWITCHER         OFFICE  OF  THE  CHIEF  ENGINEER

                                  I  PERFORPANCE  SUMMARY  OF  INDIVIDUAL  TRAINS

  T R A I N      NC  DEPARTURE  PCWER  NO    TONS    ELAPSEC TIPE   TIPE     D E L A Y   T I M E     NO  AVG
 NC SPB TYP CLS  CF    TIPE      IN    OF            RUN CC RUN 01  SAVED   WK STP CNFLCT  TOTAL    OF  SPD
                DSLS DY HR PIN   H/P  CARS           HR MIN HR MIN  HR PIN  HR MIN HR PIN  HR MIN CNFLCT

 2FB1 &  F  D  06   02 16 50  18000  114  1599C   4 45   4 22    23    15            15   5  28.8
 3C02 C  F  E  08   02 18 00  24000  114  04695   4 3C   4 09    21    15     12     27   6  30.3
 2C91 X  F  I  04   02 20 30  06800  1C0  0360C   1 3C   1 42    12-          16     16   2  25.2
 2C11 0  F  C  06   02 18 15  18000  114  1599C   4 45   4 37    08    15     15     30   6  27.2
 3C04 .  F  E  08   02 19 15  24000  114  04695   4 30   4 14    16    15     18     33   6  29.7
 2C13 ¤  F  C  06   02 19 15  18000  114  1599C   4 45   4 46    01-   15     25     40   4  26.4
 2C01 #  P  A  03   02 21 05  05000  012  0120C   3 4C   3 21    19    02            02   5  37.5
 3FB2 *  F  F  08   02 2C 30  24000  114  04695   4 3C   4 45    15-   15     48   1 03   5  26.4
 2C15 ,  F  C  06   02 21 15  18000  114  1599C   4 45   4 29    16    15     08     23   4  28.0
 3FE4 Y  F  F  08   02 21 45  24000  114  04695   4 3C   4 57    27-   15   1 00   1 15   6  25.4
 2EW3 8  F  F  02   02 22 30  06000  079  02365   4 3C   4 31    01-          44     44   5  27.8
 3CC6 a  F  E  08   02 23 30  24000  114  04695   4 3C   4 06    24    15     10     25   5  30.6
 3G02 C  P  A  03   03 01 15  05000  012  01200   3 4C   3 52    12-   18     14     32   5  32.5
 2GB1 C  F  D  06   03 00 30  18000  114  1599C   4 45   5 08    23-   15     45   1 00   6  24.5
 3C08 .  F  E  08   03 01 30  24000  114  04695   4 3C   4 12    18    15     15     30   4  29.9
 2C17 ¤  F  C  06   03 01 30  18000  114  1599C   4 45   4 54    09-   15     32     47   5  25.6
 3C1C *  F  E  08   03 02 45  24000  114  04695   4 3C   3 59    31    15     02     17   4  31.5
 2GB3 X  F  D  04   03 02 30  12000  114  10452   4 45   4 54    09-   15     37     52   5  25.6
 3FE6 ,  F  F  08   03 04 00  24000  114  04695   4 3C   4 11    19    15     16     31   2  30.0
 2FB3 #  F  D  06   03 04 00  18000  114  1599C   4 45   4 2C    25    15            15   4  29.0

 ----           --            -----  ---  -----   - --   - --  -- --  -- --  -- --  -- --  ----  -----
 80/ 57  AVRGES  06            18088  1C5  08584   4 3C   4 28    02    13     23     37  4.9  28.4

                       AVERAGE  DELAY  PER  TRAIN  PER  CCNFLICT  IS ----   4.77 MINUTES
```

Exhibit IVa

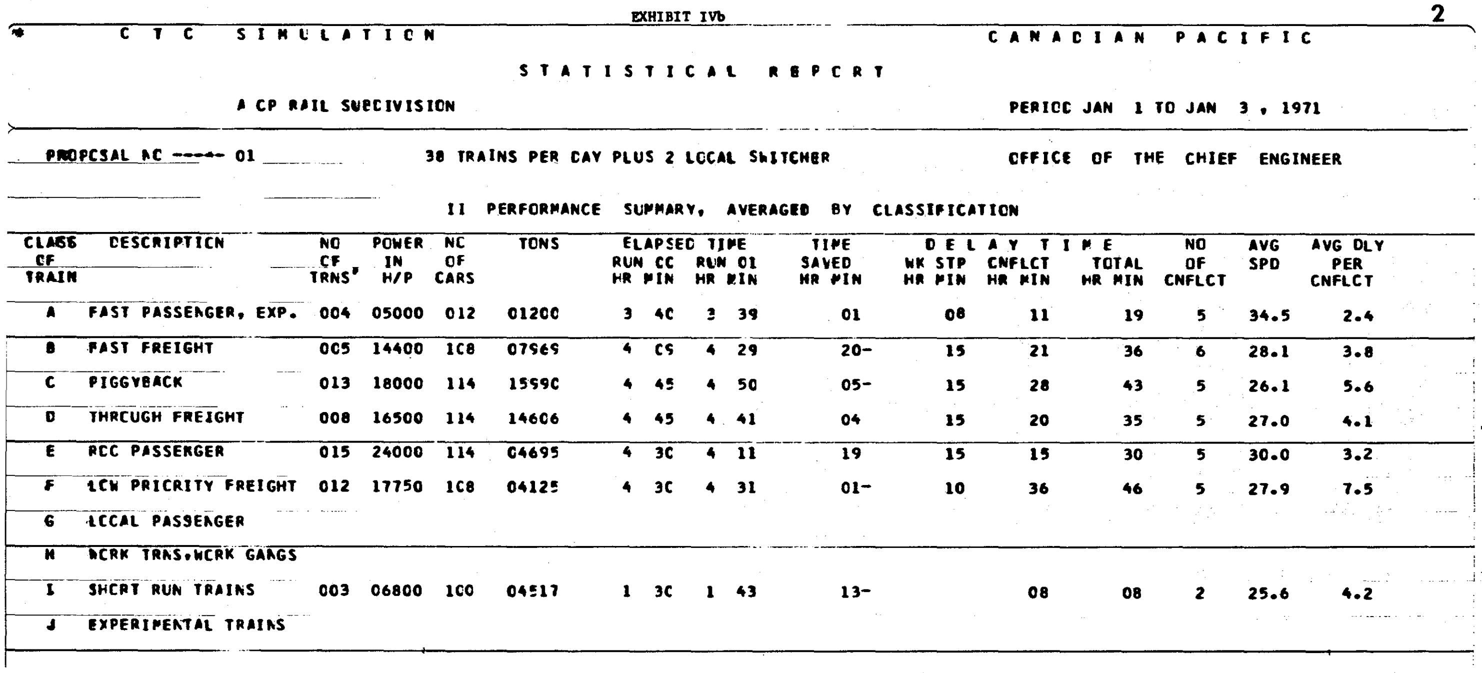


EXHIBIT IVb 2

C T C S I M U L A T I C N C A N A C I A N P A C I F I C

S T A T I S T I C A L R E P C R T

A CP RAIL SUBCIVISION PERICC JAN 1 TO JAN 3 , 1971

PROPCSAL NC ----- 01 38 TRAINS PER CAY PLUS 2 LCCAL SWITCHER CFFICE OF THE CHIEF ENGINEER

II PERFORMANCE SUMMARY, AVERAGED BY CLASSIFICATION

CLASS CF TRAIN	DESCRIPTICN	NO CF TRNS	POWER IN H/P	NC OF CARS	TONS	ELAPSEC TIME RUN CC HR MIN	ELAPSEC TIME RUN 01 HR MIN	TIME SAVED HR MIN	D E L A Y T I M E WK STP HR MIN	D E L A Y T I M E CNFLCT HR MIN	D E L A Y T I M E TOTAL HR MIN	NO OF CNFLCT	AVG SPD	AVG DLY PER CNFLCT
A	FAST PASSENGER, EXP.	004	05000	012	01200	3 4C	3 39	01	08	11	19	5	34.5	2.4
B	FAST FREIGHT	0C5	14400	1C8	07969	4 C9	4 29	20-	15	21	36	6	28.1	3.8
C	PIGGYBACK	013	18000	114	1599C	4 45	4 50	05-	15	28	43	5	26.1	5.6
D	THRCUGH FREIGHT	008	16500	114	14606	4 45	4 41	04	15	20	35	5	27.0	4.1
E	RCC PASSENGER	015	24000	114	C4695	4 3C	4 11	19	15	15	30	5	30.0	3.2
F	LCW PRICRITY FREIGHT	012	17750	1C8	04125	4 3C	4 31	01-	10	36	46	5	27.9	7.5
G	LCCAL PASSENGER													
H	WCRK TRNS,WCRK GANGS													
I	SHCRT RUN TRAINS	003	06800	1C0	04517	1 3C	1 43	13-		08	08	2	25.6	4.2
J	EXPERIMENTAL TRAINS													

Exhibit IVb

the possible types of signal blocks are shown in Exhibit I and are described below:

1. 01—intermediate signal block,
2. 02—approach signal block to a siding or to double track,
3. 03—siding signal block,
4. 04—mainline signal block adjacent to a siding signal block,
5. 06—first signal block on double track.

If the train is about to move from one intermediate block (01) to another intermediate block (01), or from an approach block (02) to an intermediate block (01), or from the first block of double track (06) to an intermediate block on double track (01) the simulator checks to see if this next signal block is occupied by another train. If this next signal block is occupied by another train, the signal will be set to prevent the train from entering the next signal block. If the next signal block is unoccupied, the second next signal block ahead is checked. If it is occupied, the signal will be set to restrict the train to enter the signal block at a speed of 30 m.p.h. If the second next signal block ahead is unoccupied, the signal will be clear, and the train can enter the signal block at speed limit.

If the train is about to move from an intermediate block (01) to an approach block (02), the simulator scans the track up to and including the first signal block ahead of the next siding. If another train going in the same direction is detected, the simulator determines whether this other train should be passed, and, if so, at which siding the pass should be executed. Signals and switches are set accordingly. Otherwise, the simulator scans the track up to the third siding ahead to detect if there is another train coming in the opposite direction. If so, the siding at which a meet must take place is determined, as well as which train will go into the siding. In determining at which siding meets will take place, the simulator always attempts to minimize the delay. However, the decision as to which siding will be used could be modified according to:

1. the location and duration of any work stops that either train may have,
2. the amount of delay time that either train has accumulated prior to this meet,
3. the relative priorities of the two trains.

Exhibit II is a hypothetical example of how the siding for a meet is chosen. If the two trains are of equal priority and neither has work stops nor has accumu-

```
                                              EXHIBIT IVc                                                        2
*       C T C    S I M U L A T I C N                                               C A N A C I A N    P A C I F I C

                                          S T A T I S T I C A L    R E F C R T

                A CP RAIL SUBCIVISION                                              PERIOC JAN  1 TO JAN  3 , 1971

  PROPCSAL NO ----- 01               38 TRAINS PER CAY PLUS 2 LOCAL SWITCHER            CFFICE  OF  THE  CHIEF  ENGINEER


                                     III  PERFORMANCE  SUMMARY  BY  SIDINGS
        LCCATICN       TYPE   MILEAGE  LENGTH          DELAY  TIME                      NO         AVERAGE
                                              WK STP   CNFLCT      TOTAL                OF         DELAY PER
                                              HR MIN   HR MIN      HR MIN           CCNFLICTS      CCNFLICT

     EAST TERMINAL     TRMNL     0.00    1.51            1  31       1  31
     SIDING 1          SDP/P     7.17    1.75            3  31       3  31               18            11.7
     SIDING 2          SDP/P    14.52    1.75            1  34       1  34               14             6.7
     SIDING 3          SDP/P    22.47    1.87            1  09       1  09                8             8.6
     SIDING 4          SDP/P    31.18    1.42            1  02       1  02                9             6.9
     SICING 5          SDP/P    37.00    1.70               53          53               13             4.1
     SIDING 6          SDP/P    44.21    1.75      04       4C          44                7             5.7
     SIDING 7          SDP/P    48.86    1.91               36          36                7             5.1
     SIDING 8          SDP/P    55.32    1.70            1  11       1  11               11             6.5
     SIDING 9          SDP/P    62.14    1.85      29       59       1  28               11             5.4
     SIDING 10         SDP/P    69.04    1.75               34          34                6             5.7
     SIDING 11         SDP/P    73.63    1.68            1  32       1  32               12             7.7
     SIDING 12         SDP/P    78.74    1.75               58          58               10             5.8
     SIDING 13         SDP/P    86.82    1.76            2  C3       2  03               11            11.2
     SIDING 14         SDP/P    93.55    1.77   12 15    3  20      15  35                1           200.0
     SIDING 15         SDP/P   103.79    2.91               47          47
                                                -- --   -- --      -- --              ----          -----

    AVERAGES FCR  15 SIDINGS                       51    1  23       2  14               9.2            9.1
```

Exhibit IVc

lated delay in excess of thirty minutes, then the meet would take place at siding *A*. However, if train 1 had accumulated more than thirty minutes delay, then the meet would take place at siding *B*.

If the train is about to move from an intermediate block (01) to the last block of double track (06), the simulator scans each signal block ahead up to the next siding. If a train is detected coming in the opposite direction, the signal at the end of the 06 signal block is set to prevent the train from entering the approach block (02). Otherwise, the train is given clear signals.

If the train is about to move from an approach block (02) into a siding block (03), the signal is set to reduce the train speed to 15 m.p.h. If the train has a meet or a pass at this siding and if the other train is already at the siding, the meet or pass is cancelled.

If the train is about to move from an approach block (02) into a main line block adjacent to a siding block (04) and if the train does not have a meet or pass at this siding, the train is given a clear signal. Otherwise, the signal is set to restrict the train speed to 30 m.p.h. If the other train is already at the siding, the meet or pass is cancelled.

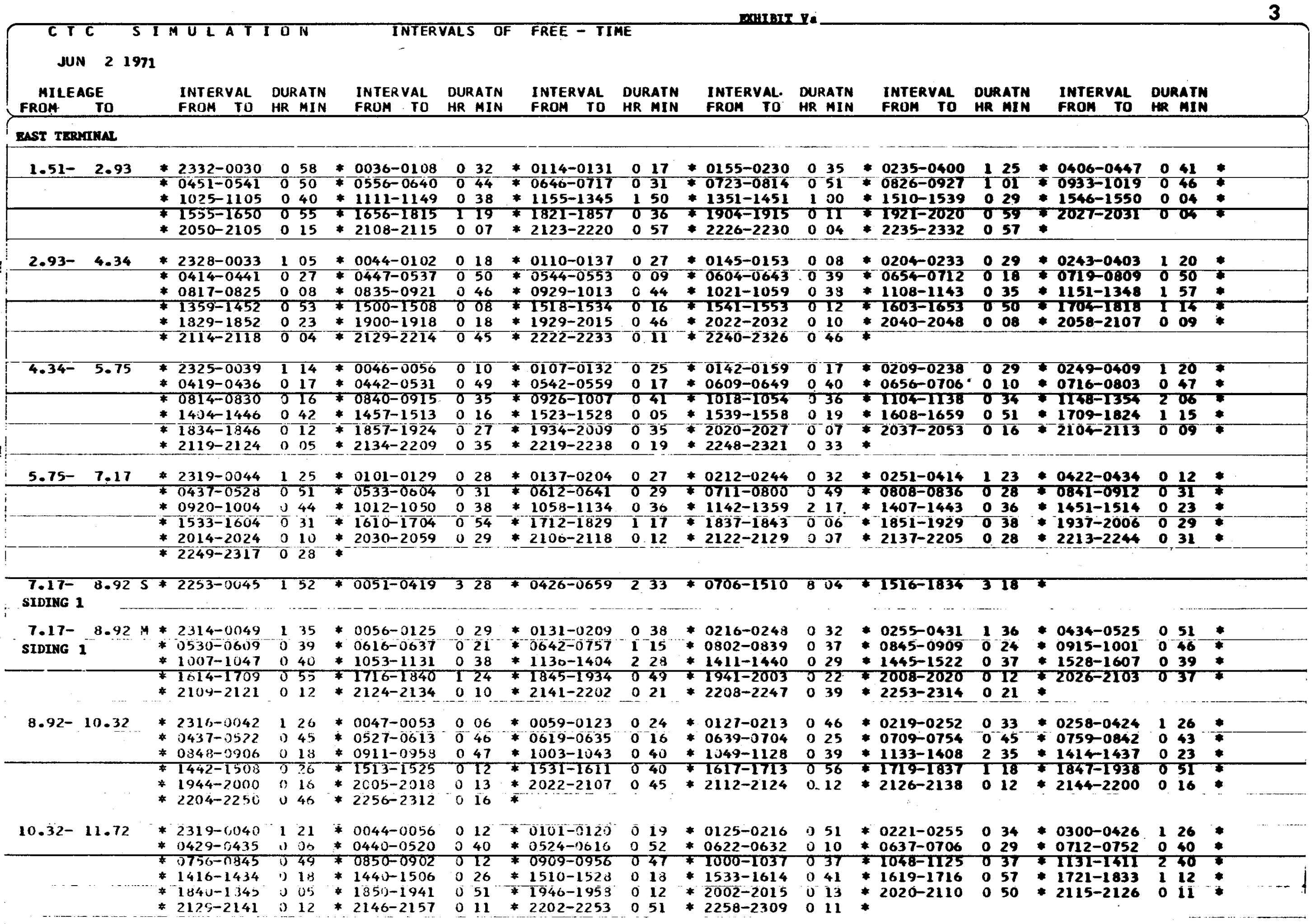
If the train is about to move from an approach block (02) into the first block of double track (06), and has a meet set up in the (06) block, the meet is cancelled.

If the train is about to move from a siding block (03), or from the main line block adjacent to a siding block (04), or from the last block of double track (06) into an approach block (02), the simulator checks whether the train has a meet or pass in this block. If so, and if the other train is not yet at the siding, or on the double track, the signal is set to prevent the train from entering the next block. Otherwise the meet or pass is cancelled. Then the simulator scans the track ahead up to the next siding. If a train is detected coming in the opposite direction, the signal is set to prevent the train from entering the next signal block. Otherwise, the train is given a clear signal.

The simulator has now evaluated the position of the train relative to other trains and has set the signal for the next signal block accordingly. If the signal indication is set to prevent the train from entering the next signal block, one minute is added to the train clock time and the train is returned to the reservoir without being moved into the next signal block. Otherwise, the time for the train to move into the next signal block is calculated and added to the train clock time. The train is then moved into the next signal block and returned to the reservoir. The simulator processes each train in the above manner until all trains have reached their terminating point.

The simulator produces three types of output:

1. a time-distance graph,
2. statistical reports,



```
                                                                            EXHIBIT Va                                                   3
  C T C    S I M U L A T I O N          INTERVALS  OF  FREE - TIME

   JUN  2 1971

  MILEAGE          INTERVAL   DURATN    INTERVAL   DURATN    INTERVAL   DURATN    INTERVAL.  DURATN    INTERVAL   DURATN    INTERVAL   DURATN
 FROM   TO         FROM  TO   HR MIN    FROM  TO   HR MIN    FROM  TO   HR MIN    FROM  TO   HR MIN    FROM  TO   HR MIN    FROM  TO   HR MIN

EAST TERMINAL

  1.51-   2.93   * 2332-0030  0 58  * 0036-0108  0 32  * 0114-0131  0 17  * 0155-0230  0 35  * 0235-0400  1 25  * 0406-0447  0 41  *
                 * 0451-0541  0 50  * 0556-0640  0 44  * 0646-0717  0 31  * 0723-0814  0 51  * 0826-0927  1 01  * 0933-1019  0 46  *
                 * 1025-1105  0 40  * 1111-1149  0 38  * 1155-1345  1 50  * 1351-1451  1 00  * 1510-1539  0 29  * 1546-1550  0 04  *
                 * 1555-1650  0 55  * 1656-1815  1 19  * 1821-1857  0 36  * 1904-1915  0 11  * 1921-2020  0 59  * 2027-2031  0 04  *
                 * 2050-2105  0 15  * 2108-2115  0 07  * 2123-2220  0 57  * 2226-2230  0 04  * 2235-2332  0 57  *

  2.93-   4.34   * 2328-0033  1 05  * 0044-0102  0 18  * 0110-0137  0 27  * 0145-0153  0 08  * 0204-0233  0 29  * 0243-0403  1 20  *
                 * 0414-0441  0 27  * 0447-0537  0 50  * 0544-0553  0 09  * 0604-0643  0 39  * 0654-0712  0 18  * 0719-0809  0 50  *
                 * 0817-0825  0 08  * 0835-0921  0 46  * 0929-1013  0 44  * 1021-1059  0 38  * 1108-1143  0 35  * 1151-1348  1 57  *
                 * 1359-1452  0 53  * 1500-1508  0 08  * 1518-1534  0 16  * 1541-1553  0 12  * 1603-1653  0 50  * 1704-1818  1 14  *
                 * 1829-1852  0 23  * 1900-1918  0 18  * 1929-2015  0 46  * 2022-2032  0 10  * 2040-2048  0 08  * 2058-2107  0 09  *
                 * 2114-2118  0 04  * 2129-2214  0 45  * 2222-2233  0 11  * 2240-2326  0 46  *

  4.34-   5.75   * 2325-0039  1 14  * 0046-0056  0 10  * 0107-0132  0 25  * 0142-0159  0 17  * 0209-0238  0 29  * 0249-0409  1 20  *
                 * 0419-0436  0 17  * 0442-0531  0 49  * 0542-0559  0 17  * 0609-0649  0 40  * 0656-0706  0 10  * 0716-0803  0 47  *
                 * 0814-0830  0 16  * 0840-0915  0 35  * 0926-1007  0 41  * 1018-1054  0 36  * 1104-1138  0 34  * 1148-1354  2 06  *
                 * 1404-1446  0 42  * 1457-1513  0 16  * 1523-1528  0 05  * 1539-1558  0 19  * 1608-1659  0 51  * 1709-1824  1 15  *
                 * 1834-1846  0 12  * 1857-1924  0 27  * 1934-2009  0 35  * 2020-2027  0 07  * 2037-2053  0 16  * 2104-2113  0 09  *
                 * 2119-2124  0 05  * 2134-2209  0 35  * 2219-2238  0 19  * 2248-2321  0 33  *

  5.75-   7.17   * 2319-0044  1 25  * 0101-0129  0 28  * 0137-0204  0 27  * 0212-0244  0 32  * 0251-0414  1 23  * 0422-0434  0 12  *
                 * 0437-0528  0 51  * 0533-0604  0 31  * 0612-0641  0 29  * 0711-0800  0 49  * 0808-0836  0 28  * 0841-0912  0 31  *
                 * 0920-1004  0 44  * 1012-1050  0 38  * 1058-1134  0 36  * 1142-1359  2 17  * 1407-1443  0 36  * 1451-1514  0 23  *
                 * 1533-1604  0 31  * 1610-1704  0 54  * 1712-1829  1 17  * 1837-1843  0 06  * 1851-1929  0 38  * 1937-2006  0 29  *
                 * 2014-2024  0 10  * 2030-2059  0 29  * 2106-2118  0 12  * 2122-2129  0 07  * 2137-2205  0 28  * 2213-2244  0 31  *
                 * 2249-2317  0 28  *

  7.17-   8.92 S * 2253-0045  1 52  * 0051-0419  3 28  * 0426-0659  2 33  * 0706-1510  8 04  * 1516-1834  3 18  *
 SIDING 1

  7.17-   8.92 M * 2314-0049  1 35  * 0056-0125  0 29  * 0131-0209  0 38  * 0216-0248  0 32  * 0255-0431  1 36  * 0434-0525  0 51  *
 SIDING 1        * 0530-0609  0 39  * 0616-0637  0 21  * 0642-0757  1 15  * 0802-0839  0 37  * 0845-0909  0 24  * 0915-1001  0 46  *
                 * 1007-1047  0 40  * 1053-1131  0 38  * 1136-1404  2 28  * 1411-1440  0 29  * 1445-1522  0 37  * 1528-1607  0 39  *
                 * 1614-1709  0 55  * 1716-1840  1 24  * 1845-1934  0 49  * 1941-2003  0 22  * 2008-2020  0 12  * 2026-2103  0 37  *
                 * 2109-2121  0 12  * 2124-2134  0 10  * 2141-2202  0 21  * 2208-2247  0 39  * 2253-2314  0 21  *

  8.92- 10.32    * 2316-0042  1 26  * 0047-0053  0 06  * 0059-0123  0 24  * 0127-0213  0 46  * 0219-0252  0 33  * 0258-0424  1 26  *
                 * 0437-0522  0 45  * 0527-0613  0 46  * 0619-0635  0 16  * 0639-0704  0 25  * 0709-0754  0 45  * 0759-0842  0 43  *
                 * 0848-0906  0 18  * 0911-0958  0 47  * 1003-1043  0 40  * 1049-1128  0 39  * 1133-1408  2 35  * 1414-1437  0 23  *
                 * 1442-1508  0 26  * 1513-1525  0 12  * 1531-1611  0 40  * 1617-1713  0 56  * 1719-1837  1 18  * 1847-1938  0 51  *
                 * 1944-2000  0 16  * 2005-2018  0 13  * 2022-2107  0 45  * 2112-2124  0 12  * 2126-2138  0 12  * 2144-2200  0 16  *
                 * 2204-2250  0 46  * 2256-2312  0 16  *

 10.32- 11.72    * 2319-0040  1 21  * 0044-0056  0 12  * 0101-0120  0 19  * 0125-0216  0 51  * 0221-0255  0 34  * 0300-0426  1 26  *
                 * 0429-0435  0 06  * 0440-0520  0 40  * 0524-0616  0 52  * 0622-0632  0 10  * 0637-0706  0 29  * 0712-0752  0 40  *
                 * 0756-0845  0 49  * 0850-0902  0 12  * 0909-0956  0 47  * 1000-1037  0 37  * 1048-1125  0 37  * 1131-1411  2 40  *
                 * 1416-1434  0 18  * 1440-1506  0 26  * 1510-1528  0 18  * 1533-1614  0 41  * 1619-1716  0 57  * 1721-1833  1 12  *
                 * 1840-1345  0 05  * 1850-1941  0 51  * 1946-1958  0 12  * 2002-2015  0 13  * 2020-2110  0 50  * 2115-2126  0 11  *
                 * 2129-2141  0 12  * 2146-2157  0 11  * 2202-2253  0 51  * 2258-2309  0 11  *
```

Exhibit Va

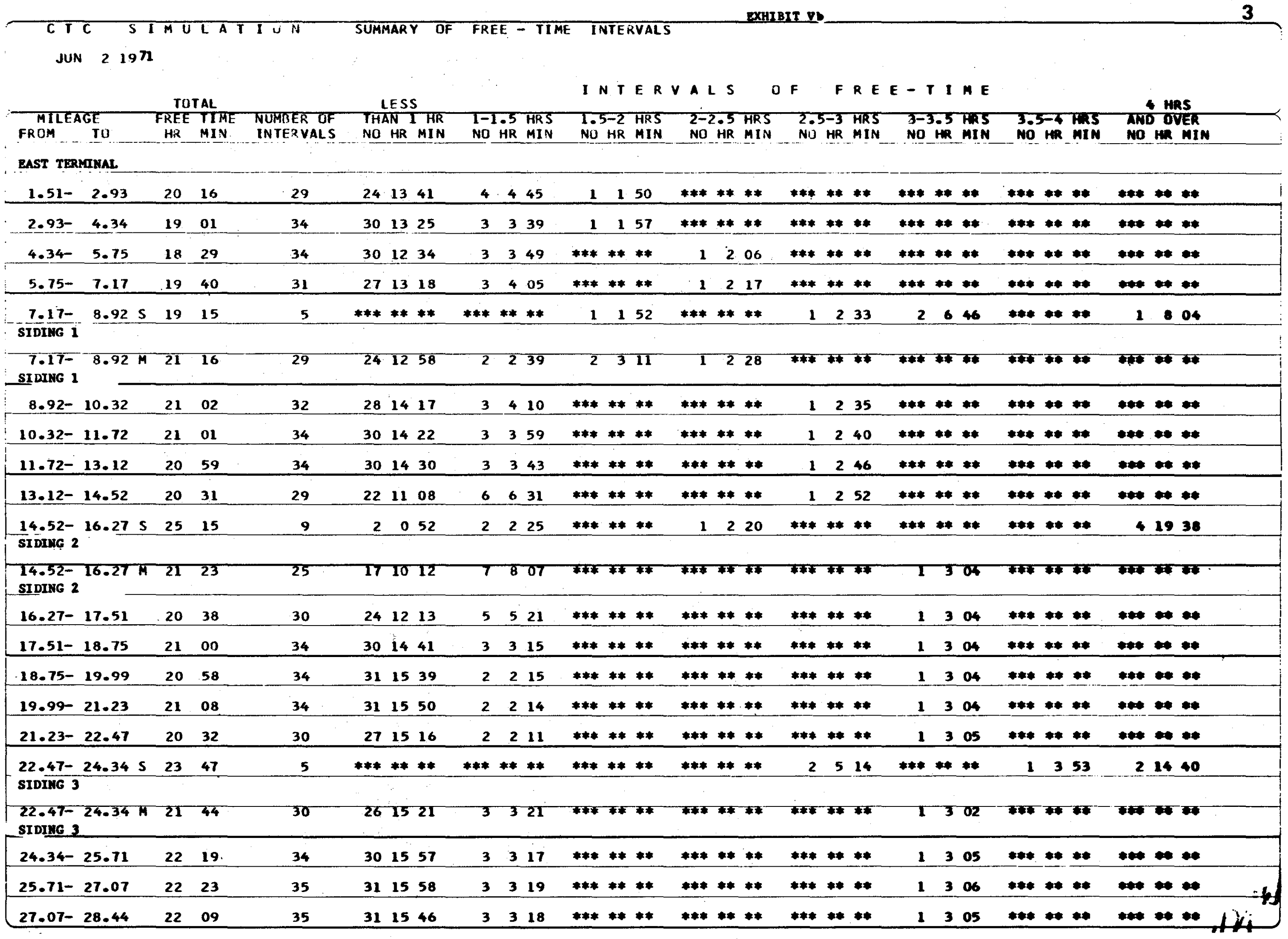
3. reports listing and summarizing the intervals that each signal block is unoccupied.

Exhibits III, IV, and V are samples of these outputs.

On the time-distance graph, time is represented on the horizontal axis, and distance on the vertical axis. The sidings are represented as the horizontal lines on the graph. The trace of the train is indicated as it traverses the territory, showing the location of any work stops, meets and passes that the train had en route. A similar graph is produced by the CTC system, which makes the simulation graph easily understood by operating personnel in the evaluation of the simulation output.

The statistical reports are comprised of three tables. The first (Exhibit IVa) tabulates the performance of each individual train in terms of its elapsed running time, which is compared to the standard or scheduled running time, the amount of delay due to work stops, meets, and passes, the number of conflicts (meets and passes), and the average speed. These results are then averaged to give general performance indicators for the simulation. The second table (Exhibit IVb) gives the same general information as the first table, but averaged for various classes of trains. This report is useful in determining the performance of each class of train. The third table (Exhibit IVc) summarizes the performance of each siding in terms of the delay due to work stops and conflicts, the number of conflicts at each siding, and the average delay per conflict. This report is very useful when evaluating alternative siding configurations.

The third type of output is the report listing and



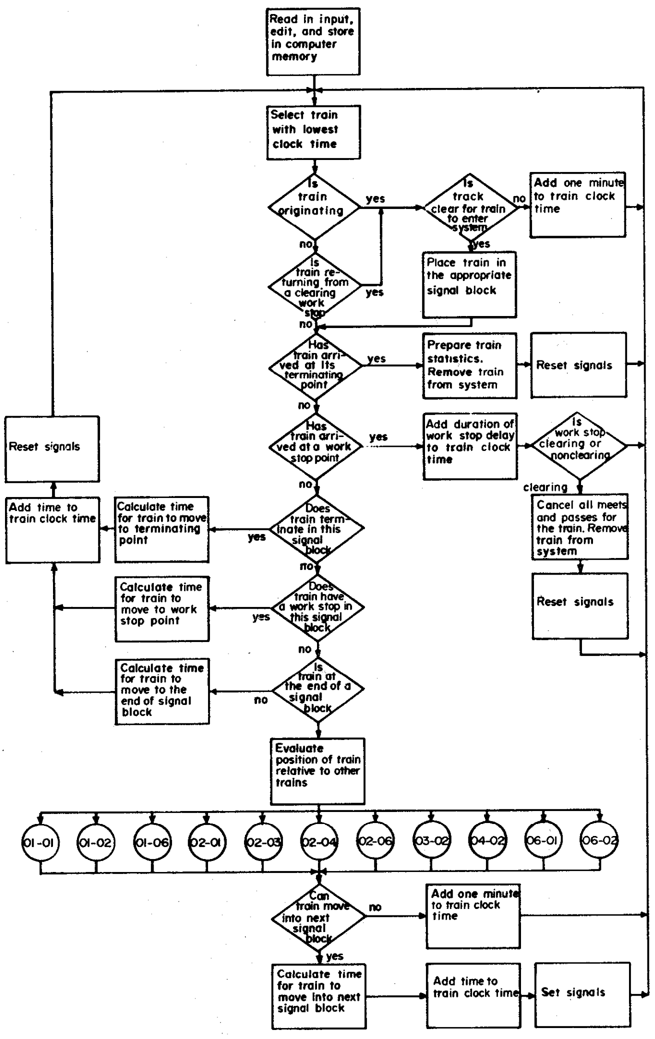
```
                                                                                EXHIBIT Vb                                         3
   C T C    S I M U L A T I O N       SUMMARY  OF  FREE - TIME  INTERVALS

   JUN  2 1971

                                                              I N T E R V A L S    O F    F R E E - T I M E
                       TOTAL                      LESS                                                                                        4 HRS
    MILEAGE         FREE TIME   NUMBER OF    THAN 1 HR     1-1.5 HRS    1.5-2 HRS    2-2.5 HRS    2.5-3 HRS    3-3.5 HRS    3.5-4 HRS    AND OVER
  FROM     TO        HR  MIN    INTERVALS    NO HR MIN     NO HR MIN    NO HR MIN    NO HR MIN    NO HR MIN    NO HR MIN    NO HR MIN    NO HR MIN

  EAST TERMINAL

   1.51-   2.93      20  16         29        24 13 41      4  4 45      1  1 50     *** ** **    *** ** **    *** ** **    *** ** **    *** ** **
   2.93-   4.34      19  01         34        30 13 25      3  3 39      1  1 57     *** ** **    *** ** **    *** ** **    *** ** **    *** ** **
   4.34-   5.75      18  29         34        30 12 34      3  3 49     *** ** **     1  2 06     *** ** **    *** ** **    *** ** **    *** ** **
   5.75-   7.17      19  40         31        27 13 18      3  4 05     *** ** **     1  2 17     *** ** **    *** ** **    *** ** **    *** ** **
   7.17-   8.92 S    19  15          5       *** ** **     *** ** **     1  1 52     *** ** **     1  2 33      2  6 46     *** ** **     1  8 04
  SIDING 1
   7.17-   8.92 M    21  16         29        24 12 58      2  2 39      2  3 11      1  2 28     *** ** **    *** ** **    *** ** **    *** ** **
  SIDING 1
   8.92-  10.32      21  02         32        28 14 17      3  4 10     *** ** **    *** ** **     1  2 35     *** ** **    *** ** **    *** ** **
  10.32-  11.72      21  01         34        30 14 22      3  3 59     *** ** **    *** ** **     1  2 40     *** ** **    *** ** **    *** ** **
  11.72-  13.12      20  59         34        30 14 30      3  3 43     *** ** **    *** ** **     1  2 46     *** ** **    *** ** **    *** ** **
  13.12-  14.52      20  31         29        22 11 08      6  6 31     *** ** **    *** ** **     1  2 52     *** ** **    *** ** **    *** ** **
  14.52-  16.27 S    25  15          9         2  0 52      2  2 25     *** ** **     1  2 20     *** ** **    *** ** **    *** ** **     4 19 38
  SIDING 2
  14.52-  16.27 M    21  23         25        17 10 12      7  8 07     *** ** **    *** ** **    *** ** **     1  3 04     *** ** **    *** ** **
  SIDING 2
  16.27-  17.51      20  38         30        24 12 13      5  5 21     *** ** **    *** ** **    *** ** **     1  3 04     *** ** **    *** ** **
  17.51-  18.75      21  00         34        30 14 41      3  3 15     *** ** **    *** ** **    *** ** **     1  3 04     *** ** **    *** ** **
  18.75-  19.99      20  58         34        31 15 39      2  2 15     *** ** **    *** ** **    *** ** **     1  3 04     *** ** **    *** ** **
  19.99-  21.23      21  08         34        31 15 50      2  2 14     *** ** **    *** ** **    *** ** **     1  3 04     *** ** **    *** ** **
  21.23-  22.47      20  32         30        27 15 16      2  2 11     *** ** **    *** ** **    *** ** **     1  3 05     *** ** **    *** ** **
  22.47-  24.34 S    23  47          5       *** ** **     *** ** **    *** ** **    *** ** **     2  5 14     *** ** **     1  3 53      2 14 40
  SIDING 3
  22.47-  24.34 M    21  44         30        26 15 21      3  3 21     *** ** **    *** ** **    *** ** **     1  3 02     *** ** **    *** ** **
  SIDING 3
  24.34-  25.71      22  19         34        30 15 57      3  3 17     *** ** **    *** ** **    *** ** **     1  3 05     *** ** **    *** ** **
  25.71-  27.07      22  23         35        31 15 58      3  3 19     *** ** **    *** ** **    *** ** **     1  3 06     *** ** **    *** ** **
  27.07-  28.44      22  09         35        31 15 46      3  3 18     *** ** **    *** ** **    *** ** **     1  3 05     *** ** **    *** ** **
```

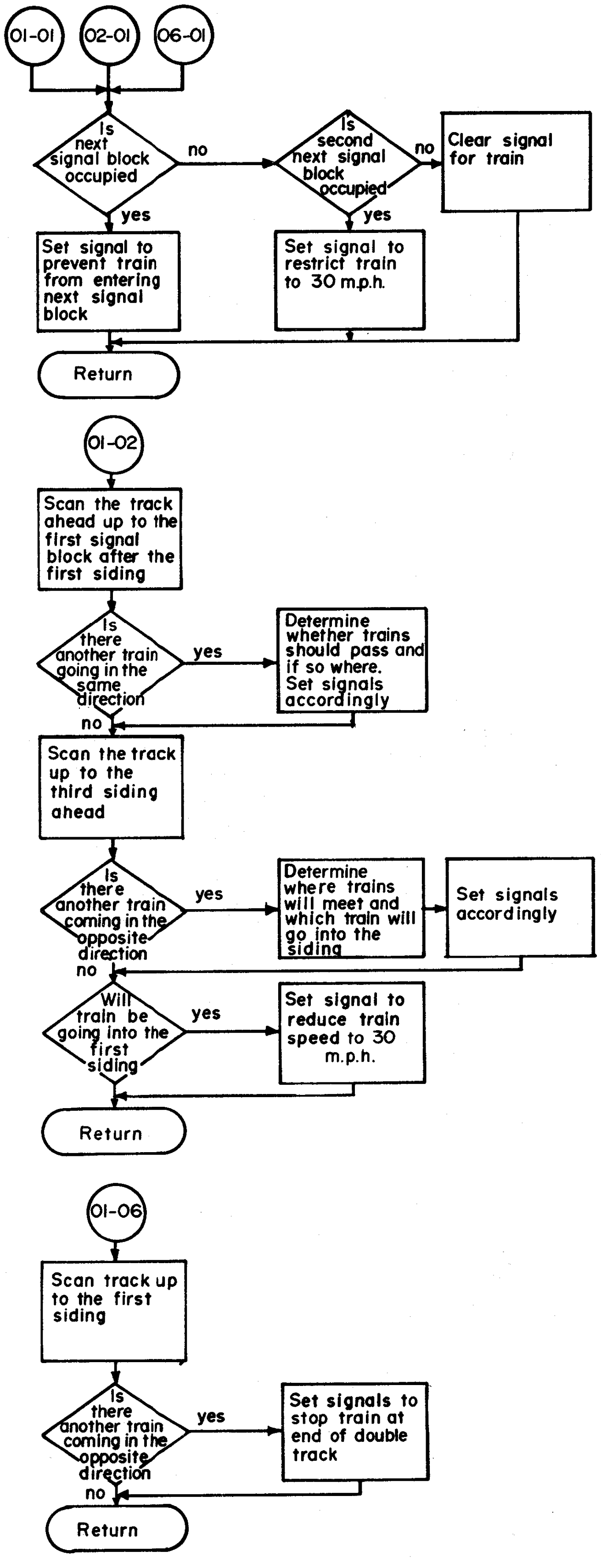
Exhibit Vb

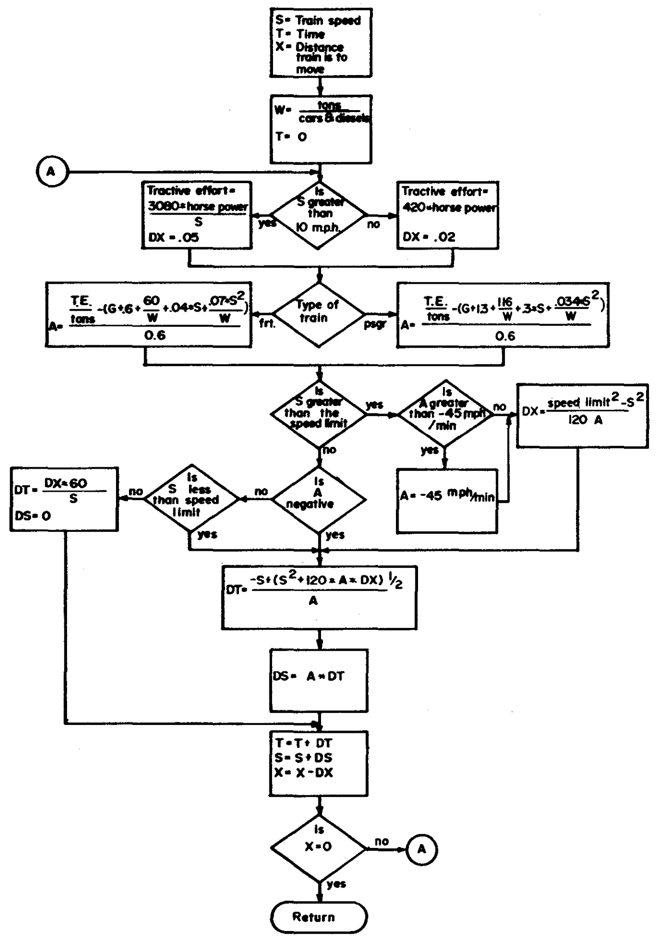
summarizing the intervals of time that each signal block is unoccupied. These reports are used to estimate the amount of track maintenance that could be performed with various train densities and schedules.

Initially, the simulation was used mainly to evaluate various siding configurations for proposed CTC installations. More recently, the simulation was used extensively to determine the effects of large increases in the density of trains through an existing CTC system, and what effects capital improvements would have. The time-distance graph was very useful in helping operating personnel visualize the projected train operations up to ten years in the future.

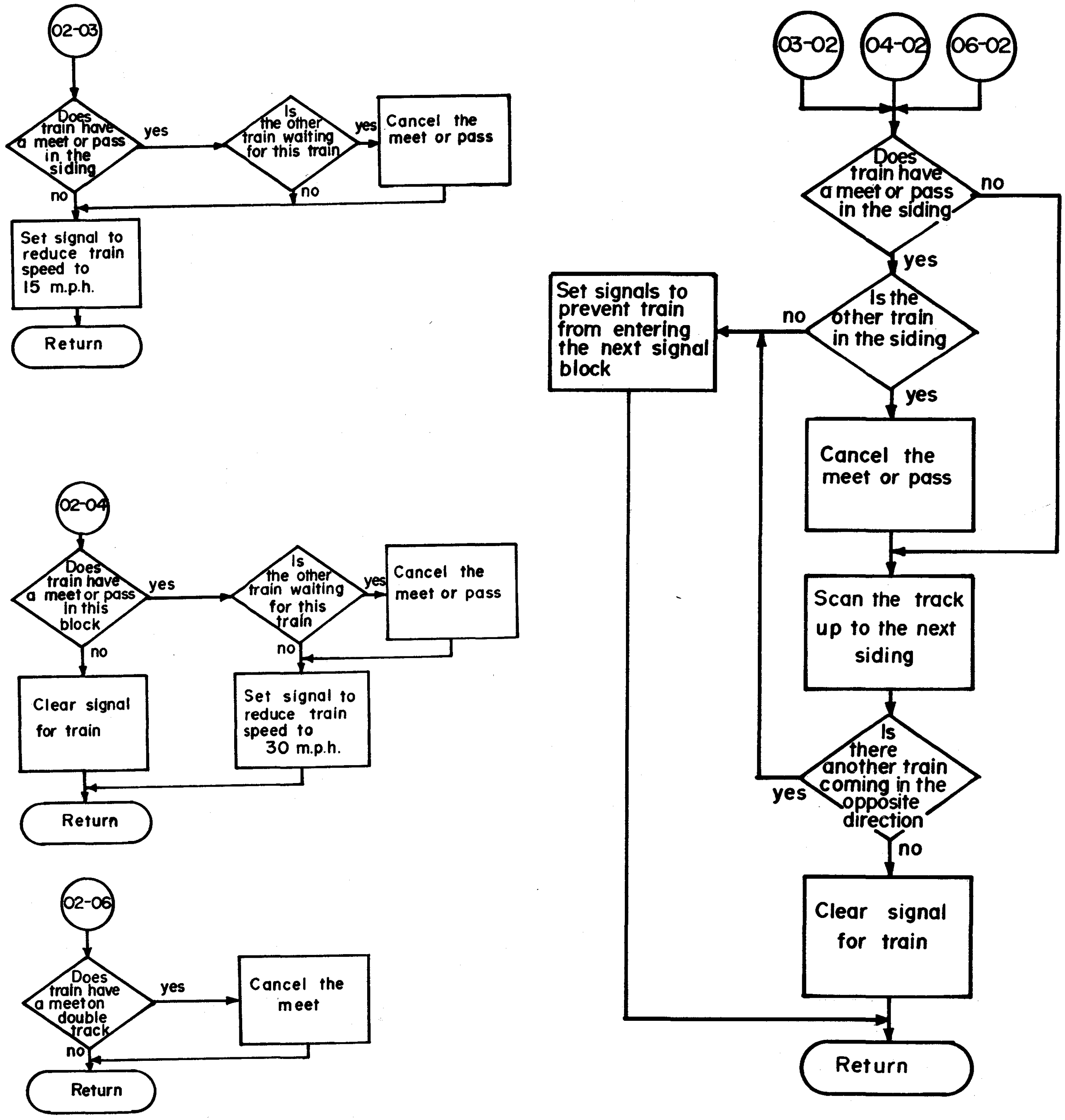
In terms of future requirements for simulation models of this type, computer aided dispatching and dispatcher training simulation models are only two of the possibilities. In both these cases, visual display systems will play a key role.


Read in input, edit, and store in computer memory
Select train with lowest clock time
Is train originating
yes
no
Is track clear for train to enter system
no
yes
Add one minute to train clock time
Place train in the appropriate signal block
Is train returning from a clearing work stop
yes
no
Has train arrived at its terminating point
yes
Prepare train statistics. Remove train from system
Reset signals
no
Has train arrived at a work stop point
yes
Add duration of work stop delay to train clock time
Is work stop clearing or nonclearing
clearing
Cancel all meets and passes for the train. Remove train from system
Reset signals
no
Does train terminate in this signal block
yes
Calculate time for train to move to terminating point
Add time to train clock time
Reset signals
no
Does train have a work stop in this signal block
yes
Calculate time for train to move to work stop point
no
Is train at the end of a signal block
no
Calculate time for train to move to the end of signal block
Evaluate position of train relative to other trains
01-01
01-02
01-06
02-01
02-03
02-04
02-06
03-02
04-02
06-01
06-02
Can train move into next signal block
no
Add one minute to train clock time
yes
Calculate time for train to move into next signal block
Add time to train clock time
Set signals


01-01
02-01
06-01
Is next signal block occupied
no
yes
Is second next signal block occupied
no
yes
Clear signal for train
Set signal to prevent train from entering next signal block
Set signal to restrict train to 30 m.p.h.
Return
01-02
Scan the track ahead up to the first signal block after the first siding
Is there another train going in the same direction
yes
no
Determine whether trains should pass and if so where. Set signals accordingly
Scan the track up to the third siding ahead
Is there another train coming in the opposite direction
yes
no
Determine where trains will meet and which train will go into the siding
Set signals accordingly
Will train be going into the first siding
yes
Set signal to reduce train speed to 30 m.p.h.
Return
01-06
Scan track up to the first siding
Is there another train coming in the opposite direction
yes
no
Set signals to stop train at end of double track
Return


S= Train speed
T= Time
X= Distance train is to move
W= tons / cars & diesels
T= 0
A
Is S greater than 10 m.p.h.
yes
no
Tractive effort= 3080*horse power / S
DX = .05
Tractive effort= 420*horse power
DX = .02
Type of train
frt.
psgr
A= (T.E./tons -(G+.6+60/W+.04*S+.07*S²/W)) / 0.6
A= (T.E./tons -(G+1.3+116/W+.3*S+.0346²/W)) / 0.6
Is S greater than the speed limit
yes
no
Is A greater than -45 mph/min
no
yes
DX= (speed limit² - S²) / 120 A
A= -45 mph/min
Is A negative
no
yes
Is S less than speed limit
no
yes
DT= DX*60 / S
DS= 0
DT= (-S+(S²+120*A*DX) 1/2) / A
DS= A*DT
T=T+DT
S= S+DS
X= X-DX
Is X=0
no
yes
A
Return

Appendix II


02-03
Does train have a meet or pass in the siding
yes
no
Is the other train waiting for this train
yes
no
Cancel the meet or pass
Set signal to reduce train speed to 15 m.p.h.
Return
02-04
Does train have a meet or pass in this block
yes
no
Is the other train waiting for this train
yes
no
Cancel the meet or pass
Clear signal for train
Set signal to reduce train speed to 30 m.p.h.
Return
02-06
Does train have a meet on double track
yes
no
Cancel the meet
Return
03-02
04-02
06-02
Does train have a meet or pass in the siding
no
yes
Is the other train in the siding
no
yes
Set signals to prevent train from entering the next signal block
Cancel the meet or pass
Scan the track up to the next siding
Is there another train coming in the opposite direction
yes
no
Clear signal for train
Return

Appendix I—Continued

A general display terminal system

by J. H. BOTTERILL and G. F. HEYNE

IBM General Systems Division
Rochester, Minnesota

INTRODUCTION

Large multiprocessing computer systems using large capacity, high-speed direct-access storage have become very common. Likewise, high speed display terminals are becoming increasingly popular, especially in special applications such as airline reservations systems. Display terminals offer an even greater potential in making available the resources of the large computer system and its data base to the majority of computer users in their office area. A generalized display capability of this type is superior to a terminal system tied to the relatively slow typewriter terminals, cards, or to a data base which is incompatible with non-terminal facilities.

Multiple **T**erminal **M**onitor **T**ask (MTMT) described in this paper is a display terminal system which extends to the individual users in their local areas the advantages of display access to a centralized computing system with its common pool of direct-access storage and high computing power. The terminal user is provided immediate access to his source programs, data, and job-control statements resident on direct-access storage. Records may be viewed, changed, added, or deleted freely and rapidly. In addition, the user may set up and submit background jobs or request commonly needed data management services. This system was designed and written to run under OS/360 MVT (Multiprogramming with a Variable Number of Tasks). A further description of the individual services is given in the Appendix.

This paper presents the design objectives for MTMT and how we attempted to meet them. First, we describe the overall system design and control, and secondly, the approach taken to several of the most important services. These objectives could be used for any display terminal system design.

OBJECTIVES

The way programmers, engineers, and administrative personnel used our computing facilities indicated that all three would greatly benefit from a terminal system. The programmer needed to conveniently modify his programs and execute them so that he could debug them more rapidly. The engineer needed to execute application programs, like Electronic Circuit Analysis Programs (ECAP), to solve his problems without having to learn programming or to commute to the computing center. The administrative personnel needed rapid, easy-to-use data retrieval and update capabilities on a real-time basis. Consequently, we decided to provide data retrieval, data updating, and job submission services along with the ability to add application-oriented support. These services had to be compatible with standard data sets or files for ease of conversion to the system and compatibility with currently provided programs.

The terminal system could not be a dedicated system since it was necessary to satisfy large production requirements on the same computer. We could not tolerate frequent interruptions in the production throughput caused by terminal system failures, and terminal down time would severely reduce the terminal system's value as an immediate data base access facility. Therefore, the terminal system had to be reliable and resilient, and had to possess a recovery capability that would protect the user from loss of data, loss of changes to data, or loss of other previously accomplished work. The system also needed to provide a convenient hardcopy capability so that users could print or punch copies of their data sets for backup or record purposes. All the above objectives had to be met without modification to the Operating System. Otherwise the future of the system would be in jeopardy

Figure 1—IBM 2260 Display Station

at each Operating System release change, plus, the cost of release change modifications would be prohibitive.

Finally, the terminal selected had to be a low cost unit which could handle large quantities of information. To provide the data transfer speed desired—yet not require extensive user training and user familiarity with the system to make it functional—we used the IBM 2260 Display Station (Figure 1). Cables allowed placing the terminals within ⅓ mile of the computer center and avoided the reliability and speed problems of telephone lines. With this display, 960 characters of 12 lines by 80 characters could be displayed quietly and quickly. The 12 lines permitted a display of an area of data to be changed, a group of records describing a machine part, a logical group of instructions (Figure 2), or an option list from which the user could select his next operation (Figure 3). Thus, option displays and self-tutorial information could be used to make all services conveniently available and avoid an extensive command language.

With the objectives set and the type of display terminal selected, we found it necessary to design and implement our own system, since our needs were not met by any available system. The following sections describe the system developed to meet these objectives.

INTERNAL PHILOSOPHY

System

Tasking structure

The needed reliability and fail-soft capability was provided by using the multi-tasking facilities of the Operating System. A separate task was created to control each terminal. This controlling task is referred to as the "terminal driver." All services requested from a terminal are attached, using OS/360 multitasking capabilities, as sub-tasks of that terminal's terminal driver.

If a service fails, the terminal driver can detect the failure, notify the user, and allow him to continue his session. Each terminal driver is attached by the system driver, whose job is to monitor all terminals attached to the system. If a failure occurs at the terminal driver task level, the system driver is in the position to detect the abnormal termination and reattach the terminal driver, thus putting the terminal back in operation at the sign-on display.

At initialization, the system driver attaches a terminal driver for each terminal and control is turned over to these terminal drivers as shown in Figure 4. Thereafter, the system driver has only two functions: (1) to reattach a terminal driver if it abnormally terminates, and (2) to respond to system operator requests (for example, specify number of active terminals, halt job submission, or halt a particular terminal).

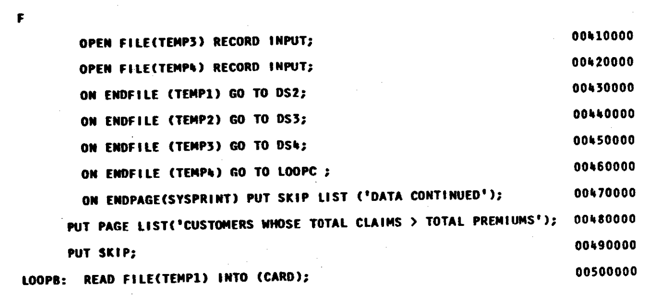

```
T#01 6 8 * *    *  EDIT    MTMT2260. SAMPLE01
 F
        OPEN FILE(TEMP3) RECORD INPUT;                              00410000
        OPEN FILE(TEMP4) RECORD INPUT;                              00420000
        ON ENDFILE (TEMP1) GO TO DS2;                               00430000
        ON ENDFILE (TEMP2) GO TO DS3;                               00440000
        ON ENDFILE (TEMP3) GO TO DS4;                               00450000
        ON ENDFILE (TEMP4) GO TO LOOPC ;                            00460000
        ON ENDPAGE(SYSPRINT) PUT SKIP LIST ('DATA CONTINUED');      00470000
      PUT PAGE LIST('CUSTOMERS WHOSE TOTAL CLAIMS > TOTAL PREMIUMS'); 00480000
      PUT SKIP;                                                     00490000
 LOOPB:  READ FILE(TEMP1) INTO (CARD);                              00500000
```

Figure 2—Sample MTMT edit display

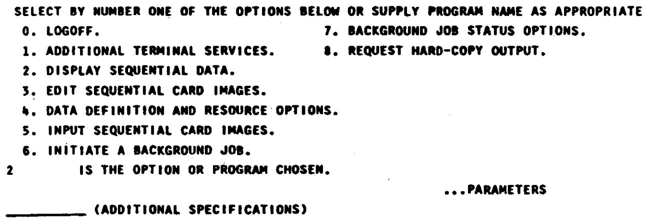

```
T#01             MULTIPLE TERMINAL MONITOR TASK OPTION MENU
 SELECT BY NUMBER ONE OF THE OPTIONS BELOW OR SUPPLY PROGRAM NAME AS APPROPRIATE
  0. LOGOFF.                         7. BACKGROUND JOB STATUS OPTIONS.
  1. ADDITIONAL TERMINAL SERVICES.   8. REQUEST HARD-COPY OUTPUT.
  2. DISPLAY SEQUENTIAL DATA.
  3. EDIT SEQUENTIAL CARD IMAGES.
  4. DATA DEFINITION AND RESOURCE OPTIONS.
  5. INPUT SEQUENTIAL CARD IMAGES.
  6. INITIATE A BACKGROUND JOB.
2        IS THE OPTION OR PROGRAM CHOSEN.
                                                      ...PARAMETERS
__________ (ADDITIONAL SPECIFICATIONS)
```

Figure 3—MTMT option menu

The terminal drivers, meanwhile, completely control the activity on their respective terminals by controlling user sign-on, displaying menus of services, and then attaching service modules. When a service module is attached by the terminal driver as the result of a request from the option menu, the service module is given control of all displays for that terminal until the user requests the service to complete. At this time the terminal driver has the option menu displayed.

Overall system status flags and information are kept in a common area in the system driver. The address of this common area is passed to each terminal driver. Each terminal driver has a common area which contains all the information pertinent to the terminal and the address of the system driver common area. The address of the terminal's common area is passed to all the modules that are attached or linked by the terminal driver. Therefore, MTMT is built around two basic common areas—a system common area and a terminal common area—each of which is accessible to each module in the system.

Interface with operating system

Multiple Terminal Monitor Task is a long-running job which does not require modification of the Operating System code. However, MTMT needs two capabilities not normally granted to background jobs. First, it needs to be able to dynamically change the unit address field in the Task Input/Output Table (TIOT) for a given Data Definition (DD) statement. This allows DD statements to be swapped to point to any data set on any permanently-mounted, direct-access volume. Thus, data sets on any permanently-mounted volume may be viewed or updated.

Secondly, MTMT must be able to issue operator commands to the Operating System such as start reader (S RDR). Both of these additional capabilities are provided by an MTMT Supervisor Call Routine (SVC).

A second SVC (which is optional) will allow the user to run background jobs that communicate back to the terminal from which they are submitted. Each such background job formats its own displays and accepts input conforming to its own standards. This second SVC is not needed unless the batch conversation capability is desired.

Since MTMT does not depend on modification of the Operating System, it is very release independent. The only areas where MTMT is subject to release dependence are in the format of the OS internal tables it reads, and the input parameters for macros and SVCs. Operating System changes in these areas seldom affect

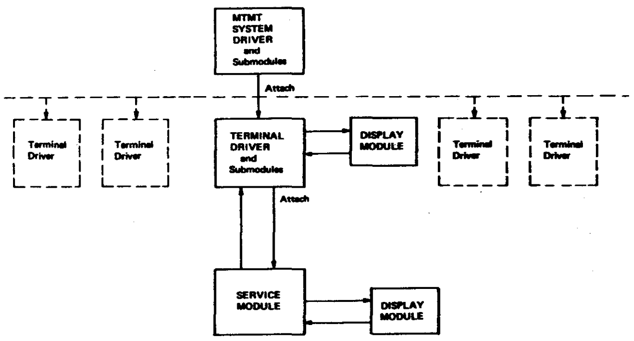


Figure 4—MTMT tasking structure

MTMT because such changes usually pertain to new fields or parameters, rather than to existing ones.

Core use philosophy

MTMT is designed to require only a small region of core. This allows normal batch processing capability on the same machine. These batch processing regions are then used by MTMT to execute compilers and assemblers for the terminal user, as well as other types of OS/360 jobs. By eliminating the need to execute compilers and assemblers in the MTMT region, the terminal system can better manage its region and require a smaller amount of core, but still provide the full services of OS/360. This control is gained by having all modules that run in the region hold to certain core use conventions. In general, modules are kept smaller than 4*K* bytes. Modules are made reenterable so that no more than one copy of a given program is in core at a time. Reenterability requires each load module to obtain a work area (dynamic area) each time it is called. All request dependent pointers, parameter lists, I/O buffers, and tables are kept in this work area and the module itself remains unchanged.

Dynamic areas from 1*K* to 4*K* are acquired from a pool of 4*K* pages in subpool 0 of the MTMT region. This pool of 4*K* pages is reserved during initialization and is managed by paging routines which keep only currently-active pages in core. When an MTMT service needs a page owned by a module which is waiting for a reply from the terminal, the page is written (rolled) to disk or drum and assigned to the requesting module. When the rolled page is again needed, it is rolled back into the same area of core for use by the proper routine. A page will not be rolled unless its area is required by another routine. In general, the modules supporting a terminal need only one 4*K* page. No more than two pages are ever needed for

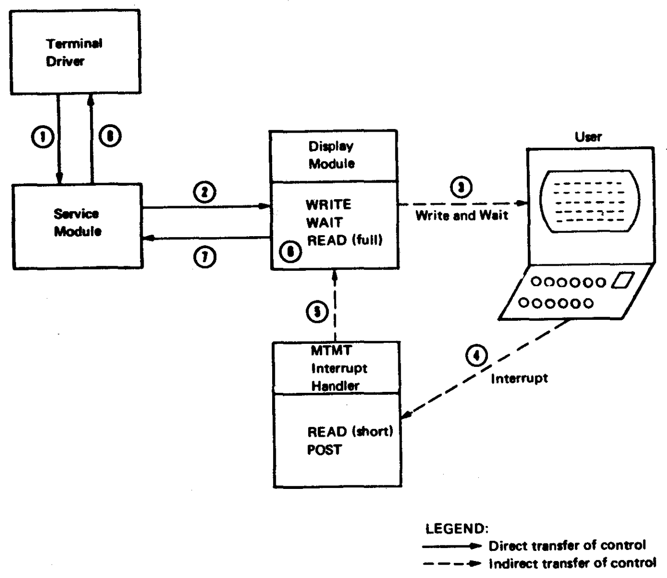


Figure 5—I/O sequence of events

a terminal. Therefore, if the paging algorithm is used, two 4*K* pages will support 4 to 5 active terminals. To provide good response time, approximately one more 4*K* page is needed for each additional two active terminals.

If an installation can justify the extra core, the paging algorithm can be nullified, thus keeping the dynamic areas in core and improving response time. If this is done, between one and two 4*K* blocks of core are needed for each terminal that may be active at a given time.

Requests for dynamic areas of less than 1*K* are handled by standard OS GETMAIN-FREEMAIN macros. Core fragmentation is minimized by having many modules request the same amount of core for their dynamic areas. If a dynamic area of greater than 72 bytes is needed, the request is given in multiples of 128 bytes from subpool 20. If a dynamic area of 72 bytes or less is needed, 72 bytes are requested from subpool 0. This increases the likelihood that the requested amount of core will have been freed by another module.

Interrupt handling and graphic I/O

To standardize the display I/O handling and better facilitate maintenance and modification, the responsibility for graphic I/O is handled by a common I/O module rather than by each MTMT module that requires a display. Furthermore, one I/O routine was created to handle each supported device. This keeps core requirements for a given installation at a minimum and simplifies the addition of support for new devices. (To facilitate discussion, this group of I/O modules will be referred to as "DISP.")

The graphic I/O is handled by standard OS support. The Graphic Access Method (GAM) is used for all local graphic I/O operations, while Basic Telecommunication Access Method (BTAM) is used for remote terminals.

A Data Control Block (DCB) forms the link between DISP and the actual terminal. To eliminate delays caused by sharing a DCB, one DCB is provided for each local terminal. This complements the 1-sec screen write time for local terminals. A local terminal's DCB is stored in its common area. For remote terminals, it is necessary to have one DCB per line. These DCB's are stored in an area GETMAINed specifically for that purpose.

Due to the multiple usage of the DCB, remote terminal users causing interrupts at approximately the same time will notice a definite wait before their interrupts are processed. With a 2400-BAUD line, it takes approximately 4 sec to completely rewrite a 960-character display. This means that 3 or 4 terminals should be the maximum on a line to prevent unusually long waits while processing interrupts.

The terminal user directs the conversation with MTMT. The terminal task writes a display, becomes inactive, and remains so until the user gives an interrupt. The interrupt is detected and routed to the terminal interrupt handler by the GAM basic attention handler. If the user gives additional interrupts before a reply to his initial request is received, the additional interrupts will be ignored. The request is serviced and a display is returned. The user once again has control. No new activity is initiated until the user requests it.

The following list, which refers to Figure 5, gives the sequence of events for local terminal I/O activity when a service module is invoked.

1. The service module is invoked by the terminal driver.
2. The service module links to DISP with a display request.
3. DISP writes the 960-character display and then waits for user interaction.
4. The user makes the appropriate modifications to the display and gives an interrupt, causing the MTMT interrupt handler to be invoked.
5. The MTMT interrupt handler issues a read from the Start symbol (▶) to the End of Message symbol (—) or up to 80 characters. DISP is then posted complete.
6. DISP determines if the data read by the MTMT interrupt handler is sufficient or if the full

screen of 960 characters is needed. If a full display reply is needed, the 960 characters are read back into the storage area provided by the service module.

7. Return is to the service module. Operations 2 through 7 act like a loop until the user has completed the service.
8. The service module returns to the terminal driver.

The Start symbol (▶) is used on the screen before control information to minimize the number of full screen reads. The response time for a short read is significantly faster than for a full screen read. This is especially true for remote terminals.

Handling interrupts for the remote terminals is different from that used by local terminals as the interrupts are synchronous and are handled in DISP. Each remote terminal has a polling list located in its common area. While the remote terminal is inactive, it will be polled every 30 sec for the detection of an interrupt. Once a user has signed on to a terminal, his terminal is polled every 1.5 sec when DISP expects an interrupt from the user. Polling is controlled by DISP instead of by an automatic BTAM polling list so that the remote terminal is polled only when necessary. This decreases overhead and improves response time.

After an interrupt is detected, a short read takes place in DISP instead of in the interrupt handling routine. A buffer area of 960 bytes plus control characters is needed for the short read in BTAM. A short read ends with the End of Message (—) character. This could mean a read of all 960 characters on the screen. After the read, control is passed synchronously to the interrupt handling routine to determine whether the terminal user has requested a special action.

The main difference between local and remote terminal interrupt handling is that the local terminals rely on an asynchronous basic attention handler which invokes a user interrupt handling routine. For remote interrupt handling, DISP polls the terminal in a specified interval and gives synchronous control to the interrupt handling routine.

Services

This section gives a general description of how we met our initial objectives in several of our services. This includes preventing the loss of user data or changes to his data, submitting jobs to the Operating System, and communicating with background programs by using an SVC for inter-region communication.

Data editing

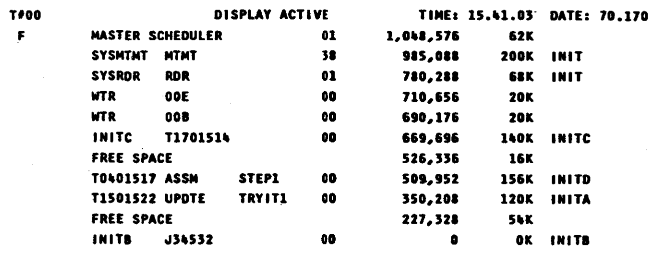
In data editing, the user must be protected from losing his data or any of the changes made to that data during his terminal session. This was accomplished as follows: when a user requests the Edit service for data set modification or key entry, the user's data set is copied into an MTMT system file reserved for that terminal on a high-speed direct-access device. This is referred to as the intermediate file because any changes requested by the user are made to this copy of his data set, rather than to the original data set. The original data set is not used for the data update process. After the user has completed his changes and wants the updated copy to replace the original copy, the original data set is opened and the updated copy is written over the original. At any time during the Edit session the intermediate file for that terminal contains the most recent form of the data set being updated and the original data set remains intact. If a system failure occurs, nothing is lost. When the system is again active, the user can request the updated intermediate file copy for further updating and/or replacement into his original data set.

Remote job entry

Multiple Terminal Monitor Task provides a complete remote job entry capability. This allows the user to submit to the system any type of job where the data for that job is on-line. Jobs may be submitted by specifying either a system cataloged procedure or a private data set containing job control statements. If a cataloged procedure is used, overrides or additional job control information may be specified to modify the standard procedure to meet the individual's needs. The same result can be achieved by using a private data set since the data set can be modified prior to the submission of the remote job.

Either of the above methods allows the user to specify the destination of his output. Under normal batch processing, the output goes to the printer, but by modifying the job control statements the user can request that the output be written into a private data set. This allows the user to view the output at the display terminal immediately after the run has completed. If a user wants a hardcopy of the results, the data set can be printed by selecting the print-punch option.

Whenever submission of a remote job is requested, a temporary data set is allocated for the job control statements. When a cataloged procedure is requested, the appropriate JOB card, the statement requesting the cataloged procedure, and any overrides are written



```
T#00               DISPLAY ACTIVE              TIME: 15.41.03  DATE: 70.170
 F      MASTER SCHEDULER              01     1,048,576     62K
        SYSMTMT  MTMT                 38       985,088    200K  INIT
        SYSRDR   RDR                  01       780,288     68K  INIT
        WTR      00E                  00       710,656     20K
        WTR      00B                  00       690,176     20K
        INITC    T1701514             00       669,696    140K  INITC
        FREE SPACE                             526,336     16K
        T0401517 ASSM      STEP1      00       509,952    156K  INITD
        T1501522 UPDTE     TRYIT1     00       350,208    120K  INITA
        FREE SPACE                             227,328     54K
        INITB    J34532               00             0      0K  INITB
```

Figure 6—Display active of jobs in the system

into the temporary data set. A reader is used to transfer the job in the temporary data set into the system job queue. Once the job is on the queue, the temporary data set is scratched. The reader is started by using the standard operator start reader (S RDR) command to the Operating System through the use of the MTMT SVC. The same procedure of allocating a data set, starting a reader, and scratching the data set is used when a user requests a hardcopy (print or punch) of a specified data set.

Displays are provided for all the system job queues and for the active jobs in the system (Figure 6). This allows the user to trace the progress of his job. If the job control statements are in error, the user is given the opportunity to scan the statements, find his error, and resubmit the corrected remote job without having to wait for his job to be returned to find the problem.

If terminal job submission is heavy or the input queues become full, the system readers started by MTMT may cause a problem. For such an installation, an in-core reader is provided which remains in core and services both the card reader for batch job input and an MTMT intermediate queue. If this in-core reader is used, remote jobs are placed directly on the intermediate queue and the reader transfers the jobs to the system input queue from the card reader and the MTMT queue according to a ratio dynamically supplied by the operator.

Background job conversation

After the initial version of MTMT was operating, we realized that a better method was needed for debugging succeeding versions of the terminal system modules. Stand-alone time was prohibitive both in cost and in terms of available time. A method was needed to execute the test modules during prime time without affecting the operational version of the terminal system or requiring additional hardware. If test modules were executed in the terminal region, the operational modules or their storage areas would be susceptible to being overwritten. This danger did not exist if the test modules were run in a background region because of OS protection keys. The use of a background region also eliminated the possibility of MTMT region fragmentation caused by test programs.

An SVC was developed to use the enqueue/dequeue (ENQ/DEQ) facilities of OS/360 to allow inter-region communication. The SVC provides the ability to move a block of data from one region to another and posts an Event Control Block (ECB) in the receiving region to indicate that a message was sent.

At MTMT initialization an enqueue is issued for each terminal. The enqueue list contains the ECB address and the address of the buffer set up to receive the message. The terminal number is used as part of the enqueue name to distinguish between terminals. Through a simple call to a communication module, the background program sends a message to the terminal from which it was submitted. The communication module passes a list to the SVC containing the enqueue name corresponding to the receiving terminal's enqueue, the address of the message, and the message length. The SVC, in turn, scans the outstanding queue control blocks to find a matching name. Upon finding a match, it extracts the receive area address and the ECB address from the queue control block. It then transmits the data across the region boundary and posts the receiver's ECB. The SVC does not transmit messages longer than the length specified by the receiver.

DISP checks this ECB before each display is generated. If the ECB has been posted, an asterisk is placed in the upper left-hand corner of the display to notify the user that his program is ready to communicate. He can then select the conversation option to receive the display generated by the background job. The user enters the information requested by the display and gives an interrupt.

The display is sent back via the inter-region communication SVC to the waiting background job. MTMT resets its ENQ and ECB for that terminal and the conversation can continue until the background job is terminated.

This conversation facility has not only been used for testing the MTMT releases since the original version, but has proved extremely useful for users. However, it was necessary to limit the duration of these conversational sessions since the background job controlling the conversation remains in core throughout the conversation, and thus can affect the turnaround time of other background jobs. Application programs have been written by users to communicate with the terminal to meet their individual needs.

PERFORMANCE

A terminal user becomes frustrated if the response to his request takes more than a few seconds. Therefore, response time is one of the prime considerations in designing any display terminal system in a multiprocessing environment. Our definition of response time is the elapsed time between the time when the MTMT interrupt handler receives control and the time when the next display is sent to the terminal.

To find out exactly how long users had to wait for responses, we monitored all requests for three weeks. These response time statistics, shown graphically in Figure 7, were gathered under the following conditions:

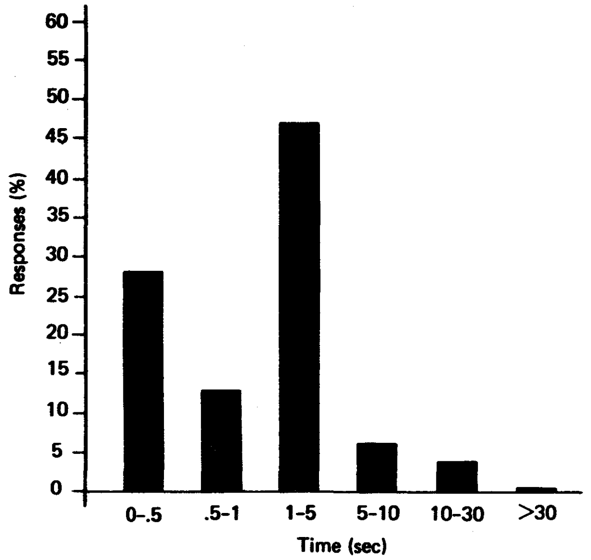
- The computer was an IBM System 360/Model 65 under OS/MVT with a million bytes of core.
- The active jobs during this period were the Master Scheduler, MTMT, 2 writers, a reader, and 4 initiators processing batch jobs.
- Only the peak load period (7:30 am to 4:30 pm) was used.
- The paging algorithm was used which rolls in and out to disk the dynamic areas of the load modules.

MTMT was run in 200*K* of core with an average of 15 of the 27 local IBM 2260 terminals active at any given time. Note that the MTMT processing time for an interrupt is the same for both local and remote terminals. If remote terminals were used, only the teleprocessing I/O time would increase. The bar graph in Figure 7 shows the number of responses during 132 hours of prime shift operation. The average response time for all terminals at any one time with the above interaction of jobs was 2.46 sec. Of the total responses, approximately 90 percent were less than 5 sec.

The number of terminals active at any one time affects the user response time. The following listing illustrates this point:

Number of terminals active	1-4	5-9	10-14	15-19	20-24
Average response (seconds)	1.11	1.39	1.96	2.55	3.02

Measurements taken on our system showed that MTMT used 2.0 percent of the time the processor was active, and 1.5 percent of the time the multiplexor channel was busy transferring data or instructions. This amounts to 1.3 percent of the total processor capacity, and 0.05 percent of the total multiplexor capacity. These figures substantiate our experience and indicate that MTMT does not degrade the performance of the entire system to any noticeable extent.



	0-.5	.5-1	1-5	5-10	10-30	>30
Total Responses	89,156	38,379	147,802	19,947	12,607	1,440
Percentage	28.82	12.41	47.78	6.45	4.08	0.46

Figure 7—Response time

CONCLUSIONS

The initial objectives for developing a general purpose display system have been exceeded in the implementation of MTMT. Through the use of the multitasking facilities of OS MVT, each display terminal operates independently and reliably. The services of OS/360 are provided in a convenient form in the local work areas.

Because standard data sets are used and accessed, MTMT is completely compatible with other Operating System facilities. Data bases and programs can be easily updated through MTMT. Programs can then be submitted to the OS through the complete remote job entry capability. We provided the use of standard compilers and assemblers without requiring a large terminal-dedicated region. This allows running normal production programs concurrent with supporting the terminals. By using an SVC to allow the dynamic specification of the volume to be accessed, we avoided modifying the Operating System. This bought Operating System release independence and ease of installation and maintenance. The convenience and practicality of the terminal system is reflected by the fact that over 60 percent of the jobs at our installation are setup and initiated from MTMT terminals. All the programming for the IBM System/3 Operating System was performed using the services of MTMT. In addition, engineers found MTMT useful for running their circuit analysis, part design, and computational programs.

The option menu concept, along with the standardized procedures throughout the services, have made the system easy to learn. The modularity of services facilitates adding special applications which use the submodules of MTMT. The success of MTMT, in the three years of operation, has proven that displays are not only the way of the future, but also the way of the present.

ACKNOWLEDGMENTS

The authors express their appreciation to W O Evans, R J Hedger, and D H McNeil, IBM GSD Rochester Development Laboratory, for their help in designing and implementing MTMT; and to R F Godfrey, IBM Poughkeepsie, for the inter-region communication SVC.

REFERENCE

1 J H BOTTERILL W O EVANS G F HEYNE D H McNEIL
Multiple Terminal Monitor Task (MTMT)
IBM Type III Program 360D.05.1.013 for the IBM 2260 Display Station Rochester Minnesota March 10 1969
172 pp

APPENDIX—MTMT SERVICES

MTMT supplies the services of OS/360 MVT to users at local and remote IBM 2260 Display Stations. Up to thirty-two terminals may be active simultaneously while normal batch job processing runs concurrently. The MTMT system options may be summarized as follows:

1. Display sequential data—A data set containing EBCDIC records of fixed, variable, or undefined type may be viewed 10 records at a time. Scanning for desired text and multiple paging, both forward and backward, are options available to the user. These data sets may contain job output or a terminal-maintained data base.
2. Edit sequential card images—A data set containing EBCDIC records having fixed format, 80-byte logical records (blocked or unblocked) may be viewed 10 records at a time. Card images in this data set may be changed or deleted and new card images may be added. The user can scan for a sequence of characters and page forward and backward through his data set.
3. Remote job entry—The user may submit a background job from the terminal by specifying a user defined or standard cataloged procedure. A hardcopy (print or punch) option for any sequential or partitioned data set on-line is available.
4. Key entry into OS card image data set—The Edit intermediate file is blanked and card images can be entered directly. The options of Edit are available to the user as he enters his data or program.
5. Data management services
   Allocation of data sets—Direct-access data sets may be allocated, deleted, cataloged, or uncataloged. The unused space on a direct-access volume may be determined.
   Move or copy OS data sets—This option provides the ability to move or copy sequential data sets or members of partitioned data sets on direct-access storage to either sequential or partitioned data sets.
   Data sets status and attributes—The attributes and allocation characteristics of a sequential or partitioned data set on direct-access storage may be obtained.
6. Console operator services
   Display active—The job name, subtask count, core location in decimal, region size, and initiator in control of job are shown for each job in the system. Free areas are also displayed.
   Display queues—All input and output queues in the system can be displayed. The job name, priority, and position in queue are shown for each job.
   Display job control statements—For each job in the queue, the job control and system statements can be scanned.
   Enter system commands—Jobs can be cancelled, moved to another queue, or have their priorities changed by the system operator.
   Terminal status—This is a display of all terminals on-line and the service in use at each terminal.
7. Background job conversation—A background job submitted from the terminal may exchange display loads of data with the user at the 2260 through inter-region communication.
8. Execution of well-behaved user written programs in the MTMT region—Subsystems using MTMT modules have been written to perform special types of data retrieval and updating. The subsystems run under the control of MTMT; there-

fore, the user has all of the MTMT services available to him.

MTMT requires the current release of OS/360 MVT on a System/360 Model 40 or larger. MTMT core requirements depend upon the number of terminals active and the modules permanently resident in the Link Pack Area. A general guideline for the core requirements of an MTMT region capable of supporting local 2260 display stations is:

2 terminals— 64*K*
8 terminals—110*K*
16 terminals—160*K*
24 terminals—200*K*
32 terminals—240*K*

AIDS—Advanced Interactive Display System

by T. R. STACK and S. T. WALKER

National Security Agency
Fort George G. Meade, Maryland

INTRODUCTION

Faced with the problem of developing a multiterminal interactive graphics display system, analysis of past experience[1] led to three specific problem areas which must be addressed in order to build a workable system. Not necessarily in order of importance or complexity these areas are (1) the difficulty in specifying interaction between the user and the computer, (2) the complexity of handling large quantities of graphic data and its interrelationship with the display hardware, and (3) the need to get away from assembly (or assembly type) languages so that one need not be a senior systems programmer in order to write a graphics program.

In past systems, the primary concern has been the handling of display images. The specification of interaction between the user and the computer has appeared of secondary importance if it was addressed at all. Unfortunately, a human being sitting in front of an array of keys, buttons, knobs, pens (both light and tablets) along with sets of lights, noise makers and the display itself cannot be programmed quite as cleanly as a card reader or magnetic tape handler. Experience has shown that attempts to control all the interactive devices available on most display systems leads to several months of "call-back debugging" to handle those unforeseen situations where "that key was supposed to be disabled."

The problem of handling graphical data has led many people to the conclusion that some form of data structure is required to facilitate efficient utilization of the display.[2] Closely related to this problem is the development of a true compiler-based language to support graphics efforts. To get valid applications for graphics systems, people who understand the application must write the programs. As long as systems programmers must be employed to write application programs, graphics will remain in the experimental, almost useful state. Current languages, either assembly or pseudo-compiler, are either so complex or unnatural that applications programmers are driven away.

A detailed analysis of these problems evolved into a design for a multiterminal interactive graphics display system supported by a compiler level language. This design brings together a number of concepts which have been used before and several new proposals. It is not intended to be the answer to everyone's problem, but from our experience it is a significant advance over what is generally available.

The system environment consists of an SEL 840MP processing unit with 64K of 24 bit core memory, movable head disk, multiple magnetic tape units, a card reader, a line printer and three modified SEL 816A graphics consoles. Interaction devices provided at each console are light pen, three shaft encoders, a bank of function switches with lamps independently programmable and an ASCII compatible keyboard. In addition to the user controlled interaction devices the system provides the application programmer with two special interaction aids. A clock pulse occurring at one second intervals is available to any interactive program. The application programmer should treat this interaction aid as an asynchronous "interrupt" since his program response is subject to system loading at execution time. The precise time of day is available should exact time be necessary to the application. The second special interaction aid is a program-to-program communication package. In essence one program may transmit a message to another, provided the latter has been enabled to accept the message. We feel this notify interaction capability is extremely powerful in a command and control environment.

To the system, jobs are classified either batch or interactive. Further, interactive jobs may be either graphics or non-graphics in nature. Batch jobs are the type most commonly supported by standard operating systems. An interactive job is one generated by the AIDS compiler and must contain at least one interaction specification. A non-graphics interactive job is one which employs either the clock or notify capability. A number of non-graphics interactive jobs concurrent in a system provide considerable enrichment. Allocation of memory and files as well as execution priority are based on job classification.

Before describing the fundamental concepts employed by AIDS, the distinction between users, programmers, and systems development types must be pointed out. A user is the ultimate consumer of a graphics system and is expected to know the application being run on the display but understand nothing of the workings of the display itself. A programmer is a person well versed in the application and possessing reasonable knowledge about the graphics system and language. He should know only the compiler language (to require him to learn assembly language would reduce his effectiveness in his application specialty) and cannot be expected to understand or perform "tricks" with the display. A systems type is one who has very limited knowledge of the applications being developed but is thoroughly familiar with the details of the display system.

It is the user's responsibility to utilize the display system effectively. The programmer must develop programs which allow the user to concentrate on his application and to develop confidence that the machine is helping rather than opposing him. Ideally, after sitting at the console and logging on, the user should forget that he is directing a computer and be led in a natural way through his application with all his concentration and effort being directed at that application. The systems developer must devise a reliable operating system offering in "a reasonable manner" all of the capabilities of the hardware being used. To the extent that he succeeds, the programmer and user will be able to perform their jobs better.

This short divergence probably has validity in many areas, but when considered with the present state of graphics development it has particular truth. The complexities of graphic systems have caused the systems developers to concentrate on devising reliable, capable operating systems which unfortunately don't offer these capabilities in what programmers consider "a reasonable manner." Thus, most programmers are reluctant (at best) to undertake graphics programs and those that do, are so enmeshed in the system that they lose their close contact with the application they're programming. Ultimately the user suffers from lack of good application programs and develops the idea that graphics is just a cute toy.

AIDS LANGUAGE AND SYSTEM DESCRIPTION

The AIDS system deals with each of the three problem areas mentioned above. The Interactive Operating System provides a simple means of organizing and specifying user interaction at the terminal. The Graphic Structure Commands allow orderly development of complex display structures. The AIDS Pre-Compiler removes the programmer from assembly language problems while providing valuable bookkeeping and error detection functions.

The interactive specification problem

The major contribution of the AIDS development is a simplified means of specifying user-computer interaction. With the WHEN Interactive Specification Statement, the programmer details precisely what interactive devices should be enabled, when, and what is to be done if one of these is activated. The system[3] was devised from analysis of the way a programmer designs the proposed user interaction with the computer. Consider a user at a graphics console. Before any pictures appear on the screen or lights flash etc., the program passes through an initialization phase. A picture is presented, certain of the interactive devices are enabled for the user's choice, and the program then pauses waiting for the user to decide what to do. Selecting one of the enabled devices causes a burst of computer activity, perhaps changing a picture, perhaps enabling a different set of devices, but eventually resulting in another pause where the user must make a selection. Looking at the interactive program as a whole, the process appears as a series of interrelated pauses and bursts of activity which can be described in a state table type notation. In the Appendix, Figure 4, states are represented by circles and conditions active in a state are represented by lines proceeding from one state to another. The conditions are written in quotes along with the responses to each condition (see example, Figure 4). This state table development is the process which a programmer, consciously or not, goes through in designing his application program. The extent to which the transitions between states are made in a natural way determines the effectiveness of the program to the user.

In the past, the specification and processing of all interactive devices, was handled entirely by the programmer, allowing uncertainty to develop as to which devices are active. AIDS uses the WHEN statement as an extension of the state table to specify interaction. The operating system then performs all enabling and disabling functions. For each condition in each state, a WHEN statement is written detailing the state, condition (interactive device), and the responses (what to do if that device is activated by the user). The program which results consists of a main program handling initialization of the necessary pictures, tables, etc., followed by a series of statements derived from the state table and detailing what the programmer wishes to do.

Given this unusual program form, the AIDS pre-compiler maps each WHEN statement into a form usable by the operating system. In this way the system developer has provided a buffer between the complexities of his system and the desired simplicity which the programmer demands. The programmer has an easily analyzed and well organized specification of his thoughts which can readily be revised or expanded and the user has a far better chance of developing confidence in the equipment since there are fewer chances that it will fail to do what he commands.

Description of WHEN statements

The following is a summary of interactive control statements associated with the AIDS Operating System.

WHEN IN STATE n, IF condition a,
THEN . . . response . . .

This is the fundamental statement which is written directly from the programmer's state table. The pre-compiler offers considerable flexibility in that the programmer can make his statements as wordy or concise as he wishes, offering a self-documenting, easily readable program or a tight, easily coded form for quick preparation of test routines. The simplest form of the statement is:

WHEN n, condition a, . . . response . . .

Where . . . "n" is a positive integer corresponding to the arbitrary number assigned to each state in the state table. If a condition is active in a number of states, a list of the states can be given in place of the single state number. If a condition is active in all states, (an emergency panic button, for example) it can be declared to be in state "0" and will be active at all times.

. . . a condition may be any of the interactive devices available on the terminal being used, or the clock or notify as mentioned earlier. Here again a long form of the condition is available for the finished documented program, but a short, easy to use form will also be recognized.

. . . a response can consist of any single AIDS or Fortran statement or any sequence of statements excluding another WHEN statement. Since any statement can be included in a response translating the programmer's state table into AIDS code is quite simple. Anything that can be done in the main program can also be done in a response.

ENABLE STATE n

This statement causes a particular state to be activated (i.e., the conditions specified in WHEN statements containing this state will now be searched for by the operating system). No further action is required by the programmer to activate interactive devices. Whenever he wished to enable a new set of devices, he enables another state.

WAIT

This statement needs background explanation. All of the routines which we developed for this system are written as reentrant code so that one copy will suffice for all terminals. However, the Fortran library is not reentrant and the code produced by the compiler has the same flaw. A serious problem develops if one allows the main program to be interrupted. If the response calls a library routine which the main program was in the process of executing, the main program's return will be destroyed along with any temporary storage locations. The WAIT statement was instituted to relieve this problem.

The ENABLE statement activates the conditions associated with a state but the main program will not be interrupted by satisfaction of a condition until a WAIT statement has been encountered. The effect of the WAIT command is to indicate that unless the user satisfies an enabled condition, the program needs no more CPU time. At the beginning of each time slice normally allocated to this program a check of the interactive devices is made, if no conditions were met, the time is allocated to another program. The program is reactivated as soon as an enabled condition is satisfied.

ENDWAIT

This statement is available in case the programmer would like to reactivate the main program to do more than its initialization role. An ENDWAIT statement executed in a response will cause the main program to restart following the WAIT statement at which it is currently stopped. In our experience this feature is a valuable one for developing parallel processes.

CHECK

The combination of WAIT-ENDWAIT doesn't quite complete the picture. WAIT causes the program to relinquish control and needs an ENDWAIT to restart it. There are times in a long calculation process where the programmer would like to see if any conditions have

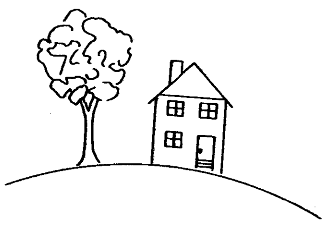


Figure 1

been satisfied without stopping the main program. CHECK provides this capability. Upon executing a CHECK the system will determine if any responses remain to be satisfied. If any exist they will be executed and then the main program continued; if not, the main program is continued immediately.

Graphical data management problem

In deciding to provide a means of graphical structure manipulation for AIDS, several considerations were involved. First, the programmer had to be relieved of the tedious problems of constructing display instruction files, and yet any capability which he had before must be available in the new system. Next, something more than the display subroutine capability offered by many hardware systems and echoed without improvement by software systems must be provided. Some reasonable means of building and organizing files in a logical and concise manner was needed. Third, a capability to link non-graphical data to the display structure must be available so that, for example, by selecting a circuit element on the screen with a light pen the programmer could easily determine the notation, component value, and other non-graphical data for that element.

The basic form of the graphical data structure which we chose is a variation on the GRIN Graphical Structure as described by Christensen.[4] The elements of the structure described there satisfy most of the requirements mentioned above and offer the most natural, easy to use yet powerful structure which we had encountered. The basic elements which will be described here are directly related to the GRIN system and many of the operations which our system provides are similar to GRIN commands; however, we made no attempt to duplicate the elaborate dual processor (GE635 and PDP-9) program execution, or memory management schemes which are part of the GRIN design. In addition, a major difference between systems is the executable data structure which we implemented as opposed to the interpretive structure developed for GRIN. Their design has a PDP-9 dedicated to each display which they calculate is idle much of the time and therefore should be used if possible to speed display execution. The PDP-9 is used to interpret everything except the actual display code. In our design the main CPU is time sliced between three displays and several "background" jobs and isn't available for file interpretation forcing us to devise an executable data structure.

Description of AIDS graphical data structure

The basic elements of the graphical data structure are the SET, INSTANCE, IMAGE, and LABEL blocks. An IMAGE is a collection of points, lines, and/or characters which is considered to be the most basic form of display entity. It is the only element which contains displayable code. An INSTANCE defines the occurrence of an IMAGE. Each time an image is displayed on the screen it is specified and positioned by means of an INSTANCE. A SET is a collection of INSTANCES whose occurrence can again be defined by another INSTANCE. An elaborate "tree-like" structure of these basic elements can be developed describing many applications in an organized manner and allowing quick retrieval of information at any level of the structure.

A trivial example of the structure and a convenient way of diagramming it (as specified by Christensen) is given here. Consider the picture of a house in Figure 1. The typical non-structured display file for this picture might describe each element in a sequence of instructions or subroutines containing instructions. Figure 2 shows a possible AIDS data structure representation of this picture which is not significantly improved over the subroutine description method. However, Figure 3 illustrates a much more complicated representation of the picture organized by the programmer according to his particular need to retrieve specific information at the various data structure levels. Note on the diagrams the representation of IMAGES as rectangles containing a drawing of what the IMAGE will produce on the screen, INSTANCES as lines connecting SETS with IMAGES or other SETS (an INSTANCE defines the occurrence of a SET or IMAGE) and SETS as circles which collect together INSTANCES at various levels. The illustrated

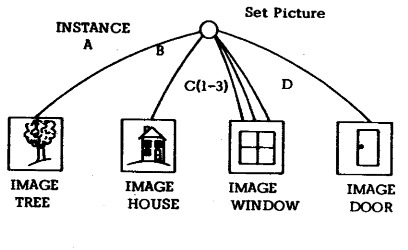


Figure 2

structure shown in Figure 3 is exaggerated but for many complex applications this type of multi-level structure is vital.

The LABEL block is used to store non-graphical data associated with any graphical element. It could be attached to each of the INSTANCES defining occurrences of the IMAGE WINDOW in Figure 3, stating the sash dimensions of each window. The organization of data within a LABEL block is entirely up to the programmer, the system provides a means of entering and retrieving data and of associating the block with any graphical element. As a follow-on to the AIDS system, a list processing capability could be implemented to improve the handling of these blocks. Examples of AIDS Structure Manipulation Statements are given in the next section.

Compiler level language requirement

The pre-compiler idea has been used on a number of systems and we feel it is a valuable tool in providing the flexibility which an interactive graphics programming system requires. An alternative is to write a complete compiler and unless significant resources are available this approach should be undertaken with considerable caution. Fortran is a good computational language but is not well suited to either interactive specification or graphical manipulation, therefore we decided to maintain the algebraic qualities of Fortran and let the pre-compiler handle all interactive and graphic statements. Examples of the AIDS statements as given here and in the Appendix exhibit little similarity to Fortran and in particular are designed to be readable by a non-programmer.

We have already discussed the Interactive Specification Statement. The pre-compiler extracts the necessary information from these commands and passes it to the operating system. The Fortran compiler was not

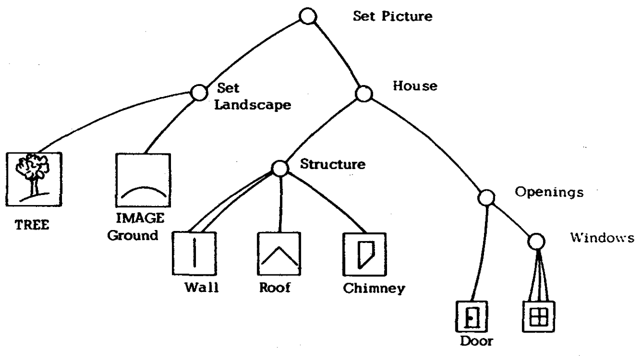


Figure 3

modified to handle interactive capabilities. The following examples show how the AIDS graphic structure manipulation statements are constructed. First the SET, INSTANCE, and IMAGE associations are specified by:

```
SET ALPHA .CONTAINS. INSTANCE BETA
INSTANCE BETA .DEFINES. IMAGE GAMMA
```

where ALPHA, BETA, and GAMMA are graphic elements declared at the beginning of the program. The descriptors SET, INSTANCE, and IMAGE are optional; the verbs .CONTAINS. and .DEFINES. determine the nature of the association and the pre-compiler further checks to be sure the elements on each side of the verbs are of the proper type. This is a valuable service which the pre-compiler furnishes since a type error undetected here can cause multiple errors later. A picture can be specified as a series of individual statements or as a concatenation in one single statement. Thus the structure in Figure 2 can be specified as:

```
SET PICTURE .CONTAINS. INSTANCE A
*           .DEFINES. IMAGE TREE
*           .AND.    INSTANCE   B
*           .DEFINES. IMAGE
*            HOUSE
*           .AND.   C(1)  .DEFINES.
*            WINDOW
*            . . .
```

Building graphical data within an IMAGE is done with the INSERT command. The following are typical examples:

```
INSERT INTO IMAGE PICTURE: A LINE
  FROM 100, 200, TO X, Y
INSERT INTO PICTURE: TEXT *THIS IS
  AN EXAMPLE*
INSERT: FORMAT 100 AT IX, IY/(ARRAY
  (I), I = 1, 25)
```

Other capabilities include:

Detaching any element from another
Clearing or copying an IMAGE
Positioning or determining the position of an INSTANCE
Entering and Fetching Data from a LABEL Block and associating it with any graphic element
Showing (causing to be displayed) any SET and all structure below that SET
Destroying any element not currently needed to conserve core

Many of the functions of the AIDS pre-compiler could have been bypassed by allowing simple calls to be added to the Fortran input deck. However, the interactive specification tables, the structure generation statements, the compile time diagnostics, and the ease of compiler maintenance make the pre-compiler concept attractive.

CONCLUSIONS

The principal programmer complaint at the present time is the slow response time. Some of this can be attributed to the generalization of the system functions and to the currently untuned status of the overall system. Another complaint concerns the lack of editing facilities at the image level. Sufficient "handles" have been designed into the system to address this problem but as yet no effort has been made to specify the functions. A possible weakness concerns the fact that the system provides no queuing of interrupts with associated user control of priorities.[5]

On the positive side, a number of programmers with limited experience have been writing reasonably complex interactive graphics applications in a matter of days rather than the weeks previously required.

REFERENCES

1 N A BALL H Q FOSTER W H LONG I E SUTHERLAND R L WIGINGTON
A shared memory computer display system
IEEE Transactions on Electronic Computers Vol EC15 No 5 October 1966

2 I E SUTHERLAND
Sketchpad: A man-machine graphical communications system
Proceedings of the 1963 Spring Joint Computer Conference

3 S T WALKER
A study of a graphic display computer—Time sharing link
Masters Thesis Electrical Engineering Department University of Maryland College Park June 1968

4 C CHRISTENSEN E N PINSON
Multi-function graphics for a large computer system
Proceedings of Fall Joint Computer Conference 1967

5 R G LOOMIS
A design study on graphics support in a fortran environment
Proceedings of Third Annual SHARE Design Automation Workshop New Orleans La May 1966

6 W M NEWMAN
A system for interactive graphical programming
Proceedings of the 1968 Spring Joint Computer Conference

7 W M NEWMAN
A high-level programming system for a remote time-shared graphics terminal
Pertinent Concepts in Computer Graphics Univ of Illinois Press 1969

8 R L WIGINGTON
Graphics and speech computer input and output for communications with humans
Computer Graphics Utility/Production/Art Thompson Book Company

APPENDIX

As an example of an AIDS program, we have selected a problem that everyone is familiar with: (1) draw an object on the screen, (2) position it wherever desired, and (3) be able to delete it from the screen. Figure 4 illustrates the State Diagram of the action which this program is to perform. Table I lists the Interaction Requirements which are derived from the State Diagram. With the help of these figures, the WHEN Interactive statements needed for this program are easily written. Figure 5 illustrates the display element structure which is developed by the program.

Initially all display data elements which are going to be used in the program must be declared. There will be one SET, four individual INSTANCES and one array of 20 INSTANCES, and a similar number of IMAGES. In addition certain Fortran variables are declared Type INTEGER. The first relational statement creates an occurrence (INSTANCE CROSS) of the IMAGE TRACK. The next two INSERT commands create a

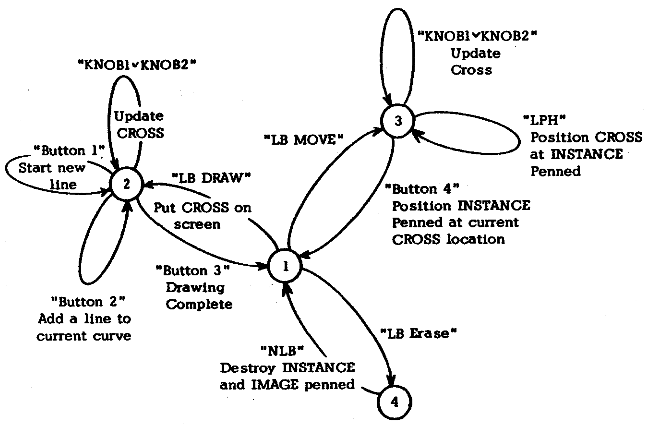


Figure 4

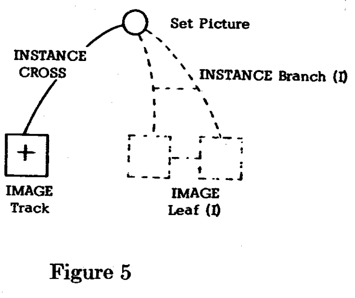


Figure 5

TABLE I—Interaction Requirements for the Example

STATE	*CONDITION*	*RESPONSE*
1	DRAW Light Button	Add an INSTANCE and IMAGE to the display. Position the Tracking Cross at 500,500 on the screen and enable State 2.
1	MOVE Light Button	Enable Light Pen for selection of object to move and enable State 3.
1	ERASE Light Button	Enable the Light Pen for selection of object to be erased.* Enable State 4.
Draw Function		
2	Button 1 L	Get starting point of curve to be drawn from KNOB 1 and KNOB 2; OLDX=KNOB1; OLDY=KNOB2.
2	KNOB1 or KNOB2	Position Tracking Cross according to current counts of KNOB1 and KNOB2; CURX=KNOB1; CURY=KNOB2.
2	Button 2 L	Add a line segment to the current IMAGE from OLDX, OLDY to CURX, CURY; OLDX=CURX, OLDY=CURY.
2	Button 3 L	Curve drawing is complete. Remove Tracking Cross from screen. Set up for next IMAGE. Enable State 1.
Move Function		
3	KNOB 1 or KNOB 2	Same as KNOB1 or KNOB2 in State 2.
3	LPWLB (Light Pen Hit with Light Buttons)	Determine position of object penned. Position tracking cross at this location. Set KNOB1, KNOB2 to same X and Y values.
3	Button 4 L	Place object to be moved at current X, Y position of cursor. Remove cursor from screen and enable State 1.
Erase Function		
4	NLB	Erase object selected.
Ø	Button 24 L	End wait mode. Job is complete.

* Note: User may not erase the Light Buttons 'MOVE,' 'DRAW' or 'ERASE,' however, movement of these Light Buttons is allowed.

cross in IMAGE TRACK, the occurrence of which is positioned at 500,500 by the POSITION command. The next six commands similarly create occurrences of the words DRAW, MOVE and ERASE. The Fortran variable I, used to control allocation of images created by the DRAW function, is set to 1. The next command causes SET PICTURE to be displayed. You will note that nothing will appear on the screen since nothing is yet attached to PICTURE and SETS contain no display 'ink.' Enable State 1 causes the system to activate those interactive devices defined in State 1. All other devices are inactive and will not burden the system with wasted interrupt processing. The WAIT command notifies the system that the main program no longer needs the CPU. As soon as a condition is met for the current state the program will be given CPU time and execution will begin at the first command of the response corresponding to the condition met.

There are three active conditions in State 1: LB DRAW, LB MOVE, LB ERASE. LB is an AIDS mnemonic for Light Button. LB DRAW notifies the system that INSTANCE 'DRAW' is a light button and that whenever State 1 is enabled, the system should display this Light Button. Thus when State 1 is enabled the words DRAW, MOVE, and ERASE will appear in the upper right corner of the screen. Further selection of one of these Light Buttons will cause the corresponding response to be executed.

Selecting Light Button 'DRAW' causes INSTANCE CROSS to appear on the screen and INSTANCE BRANCH (I) to be attached to PICTURE. Only the cross will be seen since no ink has been inserted into LEAF (I). Next State 2 is enabled, the KNOB counts for KNOB 1 and KNOB 2 are initialized to 500,500 and the response is complete. Selecting Light Button 'MOVE' turns on the Light Pen for Set Picture and enables State 3.

In States 2 and 3 rotation of either KNOB 1 or KNOB 2 will update CURX and CURY with the counts at KNOB 1 and KNOB 2 respectively.

Conditions of the form "if button n L" cause the system to turn on the lamp associated with button n

TABLE II—DRAW-MOVE-ERASE Program

```
        PROGRAM DRAW
C       THIS IS AN AIDS PROGRAM WHICH ALLOWS ONE TO DRAW FIGURES,
C       MOVE THEM, AND DELETE THEM FROM THE SCREEN
C       DECLARATION STATEMENTS
        SET PICTURE
        INSTANCE CROSS, DRAW, MOVE, ERASE, BRANCH (20)
        IMAGE TRACK, DRAWL, MOVEL, ERASEL, LEAF (20)
        INTEGER OLDX, OLDY, CURX, CURY
C       INITIALIZATION
        INSTANCE CROSS. DEFINES. IMAGE TRACK
        INSERT INTO IMAGE TRACK: A LINE FROM -10,0, TO +10,0
        INSERT: LINE 0,-10,0,+10
        POSITION INSTANCE CROSS, AT 500,500
        INSERT DRAWL: TEXT *DRAW*, 900,900
        INSERT MOVEL: TEXT /MOVE/, 900,850
        INSERT ERASEL: TEXT XERASEX, 900,800
        INSTANCE DRAW. DEFINES. IMAGE DRAWL
        MOVE. DEFINES. MOVEL
        ERASE. DEFINES. ERASEL
        I=1
        SHOW SET PICTURE
        ENABLE STATE 1
C       ALL CONDITIONS IN STATE 1 ARE NOW ENABLED
        WAIT
        STOP
C       INTERACTION STATEMENTS
C       DRAW FUNCTION
        WHEN IN STATE 1, IF LB DRAW, THEN
        SET PICTURE. CONTAINS. INSTANCE CROSS
                . AND . INSTANCE BRANCH (I)
                . DEFINES . LEAF (I)
        ENABLE STATE 2
        PUT 500,500 IN KNBCNT
        ENDRESPONSE
C       UPDATE TRACKING CROSS
        WHEN IN STATES 2,3, IF KNOB1/KNOB 2, THEN
        GET KNBCNT INTO CURX, CURY
        POSITION INSTANCE CROSS AT CURX, CURY
        ENDRESPONSE
C       SETUP STARTING POINT OF LINE
        WHEN IN STATE 2, IF BUTTON 1 L, THEN
        GET KNBCNT INTO OLDX, OLDY;
C       ADD A LINE SEGMENT FROM LAST ENDPOINT
        WHEN IN STATE 2, IF BUTTON 2 L, THEN
        INSERT INTO IMAGE LEAF (I): A LINE FROM OLDX, OLDY, TO CURX, CURY
        OLDX=CURX
        OLDY=CURY;
C       CURVE COMPLETE, GO BACK TO INITIAL STATE
        WHEN IN STATE 2, IF BUTTON 3 L, THEN
        DETACH CROSS
        I = I+1
        ENABLE STATE 1;
```

TABLE II—(Continued)

```
C       MOVE FUNCTION
        WHEN IN STATE 1, IF LB MOVE, THEN
        SETUP SET PICTURE: PENON
        ENABLE STATE 3;
C       LIGHT PEN ENABLED, NOW SELECT OBJECT TO MOVE
        WHEN 3, IF LPH,
        GET INSPEN INTO DUMMY
        GET INSPOS OF INSTANCE DUMMY INTO X, Y
        PUT X, Y INTO KNBCNT
        POSITION CROSS, AT X, Y
        SET PICTURE . CONTAINS. INSTANCE CROSS;
C       CROSS MOVED TO DESIRED LOCATION, NOW MOVE OBJECT
        WHEN 3, IF BUTTON 4 L,
        POSITION DUMMY AT CURX, CURY
        ENABLE STATE 1
        DETACH CROSS;
C       ERASE FUNCTION
        WHEN 1, LB ERASE, SETUP SET PICTURE: PENON
        ENABLE STATE 4;
C       CHOSE OBJECT TO BE DELETED
        WHEN 4, NLB,
        GET INSPEN INTO DUMMY
        DESTROY DUMMY
        GET IMGPEN INTO DUMMY
        DESTROY DUMMY
        ENABLE STATE 1;
C       PANIC BUTTON
        WHEN IN STATE Ø, IF BUTTON 24 L, THEN ENDWAIT;
        END
```

when that state is enabled as well as specify an interaction requirement.

"DETACH CROSS" removes the cursor from the screen.

Condition LPH in State 3 specifies that Light Buttons should be treated as normal display entities. That is, the Light Buttons as well as the constructed display objects may be moved with the move function.

Condition NLB in State 4 specified that the light pen is enabled but Light Buttons are not to be displayed so they will not inadvertently be erased.

Destroy DUMMY frees up the memory occupied by object DUMMY.

Finally the State Ø response specifies that the main program is to continue. The main program will execute the Fortran STOP function. A condition in State Ø is active in all states and so button 24 defines our panic mode exit.

CRT display system for industrial process

by T. KONISHI and N. HAMADA

Hitachi Research Laboratory of Hitachi, Ltd.
Hitachi, Ibaraki, Japan

and

I. YASUDA

Ohmika Works of Hitachi, Ltd.
Hitachi, Ibaraki, Japan

INTRODUCTION

Recently, in such industrial fields as steel mills, power stations, chemical plants, the use of computer control system is promoted more and more for improving productivity. For the efficient use of the computer control system various information generated in the form of characters, graphs, etc., must be accurately communicated with higher response between man and machine.

As for the data input to the computer, punched cards or punched tapes prepared by a puncher are transferred to the memory in the computer through a card reader or tape reader. In this case however, it is usually necessary to verify mispunches by another puncher or to detect rejects, and it is always difficult to change the information on the punched card or tape. Thus, this method expends additional man-hours.

Various output information from the control computer such as process condition and operation program are usually displayed on the panel which incorporates lamps or numeric display tubes. These methods, unfortunately, allow little flexibility for changing the displayed format or contents, and with a scale-up of the system, it proves difficult to display in a limited area.

For these reasons the development of such a new display system has become essential that can simplify input/output information needed for controlling industrial plants. There are already many kinds of CRT (Cathode-Ray Tube) displays[1-3] developed as peripherals for supporting business computers, that can satisfy such requirements to some extent. These displays, however, are not always suitable to the industrial application with limited ambient temperature or reliability.

In some of these CRT displays a special deflection method or a special CRT[4-6] are used, but these, consequently, mean the cost-up by the deflection circuits, and complexity of the control circuit, or maintenance. Further, although they might be used in an individual purpose, they are not always suitable when incorporated in various wide computer control application systems.

In consideration of survey results of users' requirement and also the shortcomings of conventional CRT as experienced when employed in actual industrial processes, we have developed a unique industrial process CRT display system by using standard television equipment. This compact display system can be produced at low cost, promising higher flexibility, easy maintainability, and higher reliability.

This device offers an easy man-to-machine communication in computer control systems requiring efficient construction over a wide field of applications.[7] Such applications include the indication of each function of a mill line in steelmaking, the display of the operating state of many tracks in a railway marshalling yard, the display of circuit breaker operation at a power system substation, and so on.

This report concerns itself with the construction, performance, operational principle, special features, test result of the model, etc., relating to this new CRT display system.

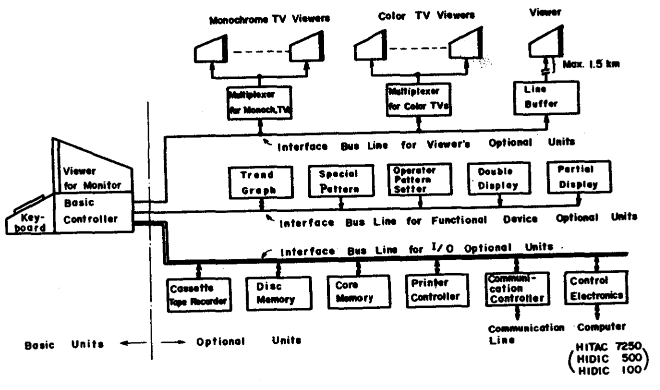


Figure 1—CRT display system

CONSTRUCTION AND PERFORMANCE OF THE CRT DISPLAY SYSTEM

A block diagram of the CRT display system is shown in Figure 1. It consists of a common basic unit, and of the optional units that can be selected according to the application.

Basic unit

A basic unit consists of a keyboard, basic control circuits, and a viewer using monochrome or color CRT.

The keyboard includes character keys composed of alphanumeric and special letter, cursor keys which control cursor position, special keys used to write simple diagrams by combining a special pattern, color keys used to select a color from among seven, and control keys used to select such modes as "transmit," "receive," "write," or "print." For example, when an operator wishes to display characters on the viewer, he depresses the mode control key to "write" and after setting the cursor on the viewer by the cursor key at the desired initial position from which the display should begin, he selects a desired color by color key, then he may begin depressing the character keys.

The basic control unit consists of a display control circuit, a character control circuit, an interface circuit etc., and it treats several signals for displaying on the viewer as explained later. As the viewer, a commercially produced monochrome or color TV set is employed.

Optional units

Several optional units are available for displaying characters other than those on the standard keyboard, patterns or marks, or for displaying the same contents on plural displays and for communicating with the computer or file memories and others.

(a) Optional units for displaying trend graphs or special patterns. The trend graph unit is used to display physical quantity that changes with time, just like a pen recorder.

The special pattern unit is used for the display of histograms, work programs, skeltons of power distribution systems, etc.

(b) Interface units for the viewer.

When the same contents are displayed on several sets of viewers, a multiplexer specified for monochrome or color TV will be used. When the viewer is located remotely, and within the maximum distance of 2 km, a line buffer unit will be used.

(c) Interface units for input/output optional units.

Interface units are used when the previously described basic unit or optional units are connected to other input/output optional parts.

The CPU (central processing unit) interface, used when the display device is connected with a control computer (e.q. the HITAC–7250, HIDIC–500, HIDIC–100 or HIDIC–50, cassette controller), operates as a peripheral device. A cassette tape recorder, disk memory, or core memory is used as a file memory. As shown in Table I, the cassette tape recorder has the maximum value of memory capacity and the core memory has the maximum value of operation speed. An optimum selection among them will be made according to the application fields.

The communication controller is used when the display device is connected with a control computer through a communications network such as a telephone line. As an example of a communications controller, a DC communication system of 2,400 Baud has been

TABLE I—Characteristics of Memories

Devices	Use Condition	Memory Capacity	Speed	Price Ratio
Cassette Tape Recorder	Off line	200 Frame/1 Tape	1-10 min/1 Frame	1 ~ 0.1
Disc Memory	On line	50 Frame/1 Disc.	20 ms/1 Frame	1
Core Memory	On line	10 Frame/4 k word	1 ms/1 Frame	1

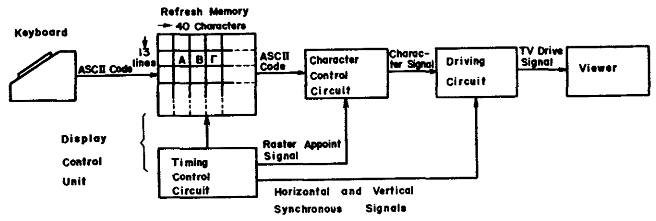


Figure 2—Block diagram of display principle

developed and is in actual use as a management information system at Hitachi Works.

The printer control unit is used when the hard copy of the characters displayed on the viewer is required.

OPERATIONAL PRINCIPLE

Since a commercially produced standard TV receiver is used as the viewer, the operational principle of the display is almost the same as those of USA and Japanese Standard Television systems. Thus, a character is displayed as a set of bright (5 × 7) dots, on a CRT surface by giving a brightness control to the raster scanning electron beam having a horizontal synchronous frequency of 15.75 kHz and a vertical synchronous frequency of 60 Hz.

Display of characters and special patterns

Figure 2 shows a block diagram of the display principle. When the alphanumeric letter *A*, *B* and a special pattern Γ key is pushed down in that order, the

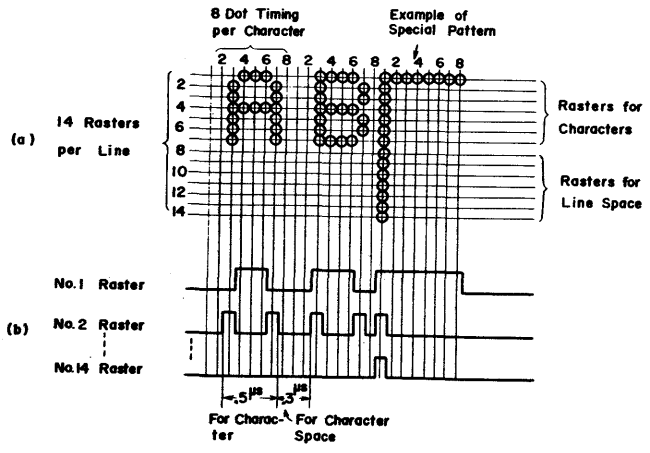


Figure 3—Example of characters and special pattern display, and their brightness control signals

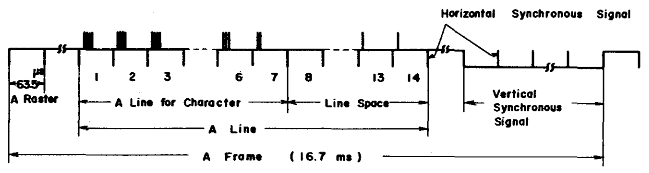


Figure 4—Example of TV drive signals

keyboard will generate 8-bit code signals (the alphanumeric code is ASCII and the special pattern code is prepared exclusively for this device) which correspond with each character accordingly. Those codes are stored in a memory in the basic controller; therefore memory capacity is made to equal the maximum number of characters displayed on the viewer, and a storing position can be appointed by an underline (a cursor) displayed on the CRT viewer. The change of the cursor position is easy and arbitrary by cursor control keys.

If a character key is pushed, and the character is displayed above the cursor, the cursor moves automatically to the next position simultaneously. The memory has a capacity able to store the corresponding number of codes to that of the displayed characters on a viewer field (e.g., storing capacity of 40 characters ×13 lines = 520 characters).

When the electron beam of the viewer (TV) is scanned by the timing control circuit signal, the refresh memory sends out the stored code corresponding to the character control circuit.

By these transmitted codes and the signals showing the position of the scanning line given from the timing control circuit, the character control circuit generates a brightness control signal necessary for forming the characters or patterns.

As shown in Figure 3(a), the unit size of one character and a special pattern is determined by 8 dots timing in the horizontal direction and 14 scanning lines in the vertical direction. Thus, the unit size of a character is

TABLE II—Special Features of Color Code Storage Method

Hard-ware	Memory Construction	Same as monochrome TV
	Color Control Circuit	It is necessary to use an 8-bit color decoder and an encoder for color R, G and B.
	Interface to CPU	Same CE as monochrome TV
	Interface to Keyboard	Same as monochrome TV
Soft-ware	Character construction	For color code a character space occurs every color change
	Data Making	Data making is facilitated by using one byte per character.

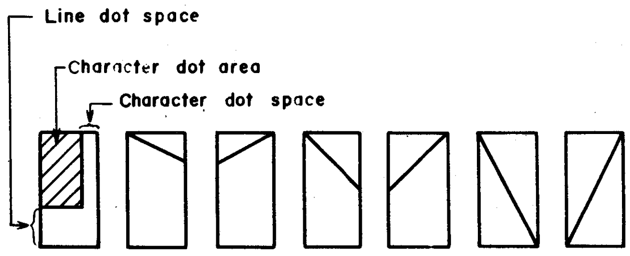


Figure 5—Special patterns for graph display

displayed in a method whereby brightness is controlled in a fixed position of an 8 × 14-dots matrix. The unit size of a special pattern has a usable limit in all 8 × 14 dots of a matrix, but a character has a usable range of 5 × 7 dots of a matrix. For this reason, when A, B and Γ are displayed, the control signal for each raster becomes as shown in Figure 3(b).

The brightness control signal, and the horizontal and vertical synchronous signals are mixed to form the video signal of a standard TV system at the driving unit, and the viewer's CRT is driven by this video signal. Figure 4 is an example of such a driving signal.

Color display

For displaying colored characters, the specified color code must be stored in the memory. In this case, a color code storage method in which no extension of the memory format is necessary is adopted.

In this storage method, the remaining codes as spared from assigning a character or a special pattern is allocated for each color of, for example, red, green, blue and so on totaling 7 colors, and color identification of character is made from a color code using one character space at the head of the character codes. In this way, the memory stores the color code and the character code in series. This color code storing method offers the special features as shown in Table II.

Trend graph display

The deflection method of a CRT electron beam is divided into the two main classes of random positioning and raster scanning. With raster scanning, as a commercial standard TV set can be used, the cost of its CRT control circuit is much cheaper compared with the case of random positioning. But it was very difficult to display a continuous curve, and its programming was very troublesome. However we have succeeded in solving these problems and developed a new graphic display method by raster scanning as described subsequently.

(a) Principle of trend graph display.

The displaying of graphs by raster scanning will be divided into two categories. That is a curve approximation method and a piecewise linear approximation method. In the former, the error of approximation can be made as small as the viewer's resolution.

However, it is not advantageous from the viewpoint of cost as it needs a large memory capacity with the increase of the number of line elements. Therefore a piecewise linear approximation is selected.

The piecewise linear approximation method is further classified into a dotted pattern or a brightness line method. In the dotted pattern method a curve is displayed by combining several special patterns as shown in Figure 5. The principle of operation of this method is identical to that in the special pattern as explained in the character displaying. Although this method is superior in histogram display etc., it requires, as its drawback, a large number of patterns for displaying a curve.

In the brightness line methods, when displaying a curve designated as (a) in Figure 6 by a broken line

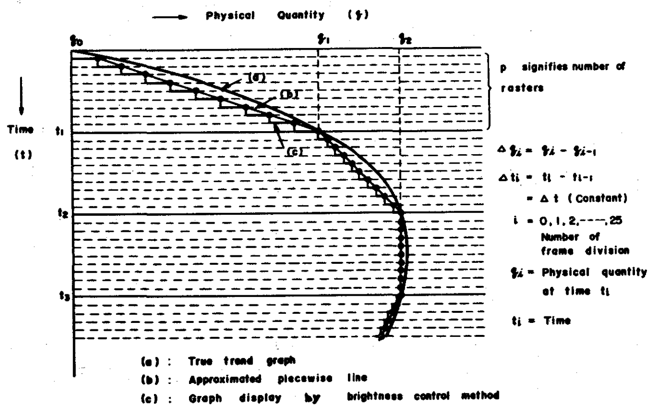


Figure 6—Graph display method

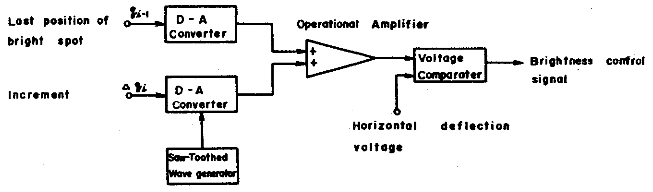

Last position of bright spot
y_{i-1}
D-A Converter
Increment
Δy_i
D-A Converter
Saw-Toothed Wave generator
Operational Amplifier
+
+
Voltage Comparater
Horizontal deflection voltage
Brightness control signal

Figure 7—Analog method of trend graph display

approximation, either method by analog or digital display can be selected. As shown in Figure 7, an analog displaying method of a trend graph is constructed by utilizing a digital-to-analog converter, multiplier, and voltage comparater.

Its operational principle is that voltage of a saw-toothed wave generator corresponding to a value of $\Delta qi/\Delta ti$ is added to vertical deflection voltage corresponding to $qi - 1$ (for an explanation of the symbols, see Figure 6) by the operational amplifier, and its output and horizontal deflection voltage proportionate to the present position of the electron beam are compared in the voltage comparater, and in coinciding timing, a brightness control signal is given to the CRT. Through these operations, the points marked with block dots on the line (b) in Figure 6 are brightened.

The digital method is practically the same in basic principle as the previously described analog method, except that operations of addition and comparison are effected by the digital value. That is, by adding an analog value corresponding to $\Delta qi/p$ to the final value of the last position, the bright spot is moved. By this method, the graph is displayed as shown in Figure 6(c).

(b) Comparison of display methods.

As a criterion of comparison, it is assumed that the memory capacity for the graph display area is as large as that of the character display area (40 characters $\times$ 13 lines), and the number of patterns is 32. Especially, in the case of the brightness line method, it is assumed that a division unit of time axis has 7 rasters ($p = 7$), the resolution of spot position in the horizontal direction is half dot timing ($n = 640$) for a character dot, and a maximum value of Δqo is 255 dots (8 bits).

A comparison between these graph approximation methods obtained under these assumptions is shown in Table III. As the result, it was found that the digital method was most advantageous for its lower cost, and it was adopted. This method, however, has some limit in displaying the curve (i.e., an origin must be in the upper, left corner; the vertical direction must be selected in time axis and the total number of piecewise lines must be under 26). However, a unit length of each axis can be selected or changed by the software.

Thus far, operational principles of the character displaying, the color displaying and the trend graph displaying were described; however, further descriptions in these regards will be omitted.

TABLE III—Comparison of Several Graph Approximation Methods

Graph approximation method		Memory capacity (bit)	Control circuits	Price ratio	Functions	No. of divisions	
						Vertical	Horizontal
Pattern generation method		$\frac{m}{p} \times \frac{n}{q} \times r_p$ (MOS) 4,160	Read/Write circuit Pattern decision Pattern generation	1	Designable in graph form	13	320
Brightness line control method	Analog method	$2(\frac{m}{p} \times r_a)$ (TTL) 416	Read/Write circuit D-A converter voltage adder and comparator Sow-toothed wave generator	0.5	Trend graph having origin only on upper and left corner	26	640
	Digital method	$\frac{m}{p} \times r_a$ (TTL) 208	Read/Write circuit Adder Decision of horizontal position	0.25	Same as above	26	640

Note, m : Number of rasters/Displayed frames
n : Number of vertical divisions/Displayed frames
p : Number of rasters/Unit divisions
q : Position resolution/Unit length of a raster
r_p : Number of bits determined by a pattern
r_a : Number of bits of A-D converter
r_d : Number of dots determining brightness control increment

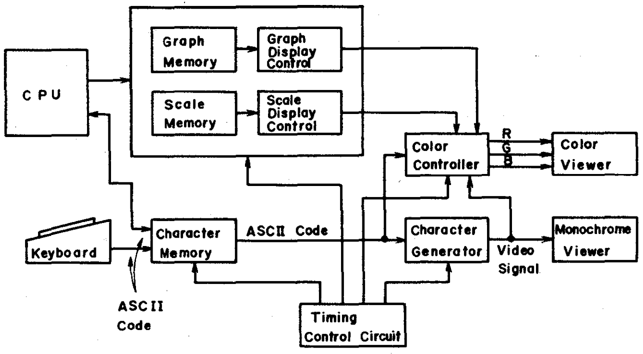


Figure 8—Block diagram of character and graph display of model group

OUTLINE OF MODEL SET

The block diagram of a model device is shown in Figure 8. This device is able to display characters, trend graphs, and their scales on the viewer of a monochrome or color TV receiver. In this chapter the control circuit of the trend graph displays will be given further explanation. As illustrated by using Figure 6 in the previous chapter, when a curve is approximated by broken lines, the vertical axis (time axis) is equally divided as a unit of p-rasters, and unit time $t_{i+1} - t_i$ or a section is assigned to a piecewise line. In a section, a horizontal component of a piecewise line (for example q_oq_i) is equally divided by the number of p, so that a line element of constant length of $(q_1 - q_0)/p$ is displayed while each raster is moving along a line (b) in Figure 6 in a former section. In the model device, the number of raster p is selected as $p = 7$ in order to limit the memory capacity. A memory capacity to accommodate the same word number as a divided number of time axis is necessary and in this case the memory capacity is 26 words. The data format of a word is constructed as shown in Figure 9, and the length of the line component

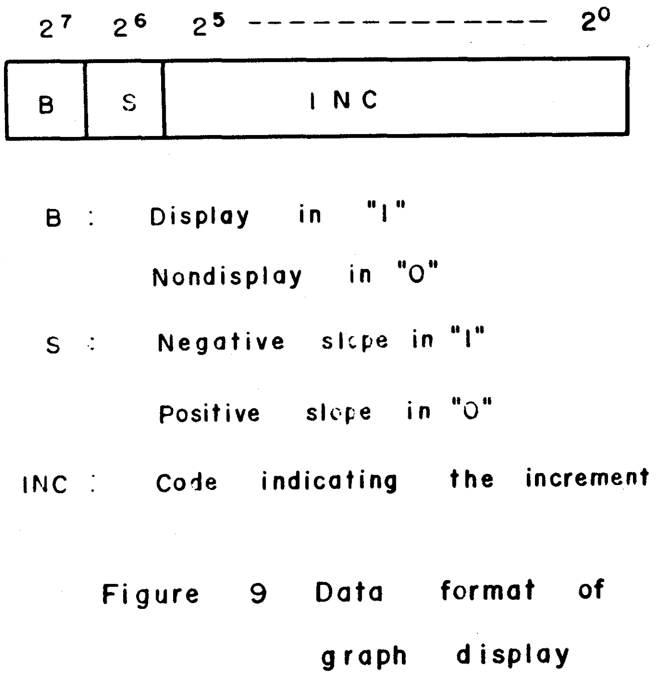


Figure 9—Data format of graph display

is indicated by the INC part. This memory is termed as "graph memory."

Circuit construction of the trend graph display is shown in Figure 10. As clear from the figure the hardwares consist of several circuits such as (1) a graph memory having a capacity of 26 words whereby the length of divided line elements in every piecewise line is stored; (2) a start and stop register that sets a start and stop point of the brightened line on the raster; (3) an arithmetic gate that calculates traveling distance of the brightened spot for every raster; (4) an increment counter, for coding the position of the brightened spot with digitals; (5) a coinciding circuit that effects coincidence between data of the start and the stop resistor and of the increment counter; (6) a brightness

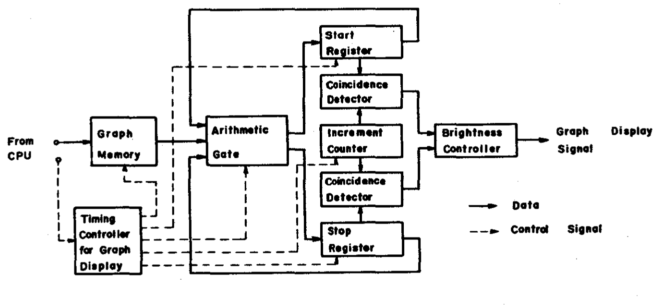


Figure 10—Circuit construction of graph display

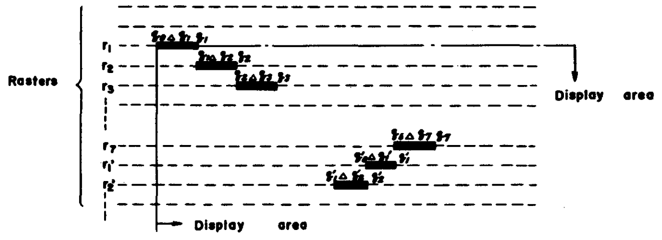


Figure 11—Example of group display

controller that generates the brightness control signal from the coincidence detecters; and (7) a timing controller that generates the timing and control signals.

Nextly, the function of the trend graph display circuit will be explained. In the example of a trend graph display shown in Figure 11, data with a length of line increment $\Delta q = \Delta qi$, $\Delta q' = \Delta q'i$ ($i = 1, 2, \cdots, 7$) and so on are fed to the graph memory. By the control of the timing circuit, the contents of the memory is read out, and calculations of $qi = qi - 1 + \Delta q$, $q_{i+1} = q_i + \Delta q$ and so on are carried out in the arithmetic gate, and their results are stored in the start and the stop register respectively. As soon as a brightened spot of raster r_1 enters the graph display area, the increment counter starts counting, and when the contents of the counter coincides with the start register (qi), the brightness control signal is generated from the brightness controller. This signal output is kept to generate until the contents of the counter coincide with the stop register ($qi + \Delta q$). Just before the scanning spot moves to the next raster, r_2, contents of the start register are replaced by q_1, and contents of the stop register are replaced by $q_1 + \Delta q$ by transfer control between both registers (as shown in the upper column of Table IV).

When the slope of broken lines is in the negative sign with the time axis as a base, or in the case of reverse slope as $\Delta q'i$ as shown in Figure 11, information will be transferred between the two registers as shown in the lower side in Table IV.

Since the model device uses 6 bits for an appointment of the length of increment lines, the slope of the

TABLE IV—Transfer Control Between Registers According to Slope of Increment

Slope of Increment	Contents of Start Register	Contents of Stop Register
+	(Stop Register)	(Stop Register) + (Increment)
-	(Start Register) - (Increment)	(Start Register)

piecewise line can be selected for an angle of 0 or ±63 against the vertical axis.

TESTED RESULTS

Some examples of character display are shown in Figure 12(A), and the trend graph display is shown in Figure 12(B). The test model performed reliably at an ambient temperature from 0 to 50°C with the source voltage of 100V, +10 percent, −15 percent.

SPECIAL FEATURES OF THE DISPLAY DEVICE

The newly developed display system offers the following special features:

(1) Higher reliability and simpler maintenance

Because the composing elements are carefully selected, this device can be used under severe conditions as familiarly encountered in usual electronic devices for industrial use, and this can operate normally even at ambient temperatures from 0 to 50°C and with voltage variance of 100V, +10 percent, −15 percent.

ICs and LSIs are used in the logic elements, in the character generator and in the main memory, and an all-solid-state transistor TV receiver is used as the viewer.

(2) Higher flexibility

The basic units such as keyboard, control unit, display unit, source unit, etc., of the device are designed as blocks and are incorporated into each standard unit of specific performance. These are easily expandable to meet various application requirements such as changes or additions of the characters, colors, trend graph performances, and such as enlargements of the display units and the printer units or such as an installation of long-distance transmission system between the control unit and the viewer unit, and so on.

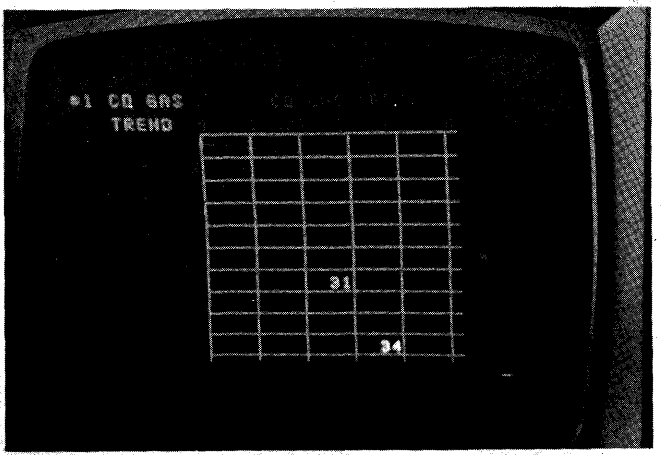
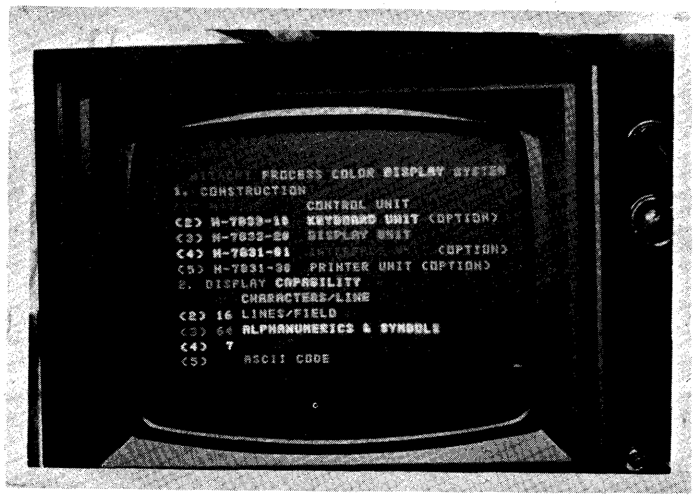


Figure 12 (B)—Trend graph display



```
[illegible] PROCESS COLOR DISPLAY SYSTEM
1. CONSTRUCTION
[illegible]           CONTROL UNIT
(2) H-7833-10  KEYBOARD UNIT (OPTION)
(3) H-7833-20  DISPLAY UNIT
(4) H-7831-01  [illegible]    (OPTION)
(5) H-7831-30  PRINTER UNIT (OPTION)
2. DISPLAY CAPABILITY
         CHARACTERS/LINE
(2) 16 LINES/FIELD
(3) 64 ALPHANUMERICS & SYMBOLS
(4)  7
(5)    ASCII CODE
```

Figure 12 (A)—Character display

(3) Lower cost

As the result of the development of a new control method, a commercially produced low-priced TV set can be used for the viewer. The use of IC or LSI has enabled the reduction of the component parts, as well as the easy manufacture and inspection since the control circuits were assembled into each unit of its specific performance. By optimum selection of options from many available here, a display system with a high performance per cost can be achieved.

EXAMPLES OF APPLICATION

As shown in Table V, this display system is enjoying many practical applications, and in the future, we anticipate its use in every industrial field.

CONCLUSION

This report has described a newly developed industrial process display system, and explains the construction

TABLE V—Examples of Industrial Applications

Application field	Objective (and system construction)	Performance	Optional unit
Steel-making	Management information system of wide flange mill (H-8400x2+H-500CRTctr+CRTs)	1. Monitering of process 2. Display of operation procedure 3. Inquiry system 4. Data setting or adjustment	(1) Trend graph (2) Continuous color (3) Multiplexer for color (4) Blink (5) Line buffer (6) Control electronics
Electric power system	Economic load dispatch and monitoring of automatic dispatch system (Telemeter+Data exchange controller +H-7250<+CRTctr+CRT(ELD) +CRTctr+CRT(Monitoring))	ELD use 1. Table display of predictive load 2. Data setting or adjustment Monitoring use 1. Display of skeleton diagram 2. Flicker of condition change	(2), (3), (4), (6), (7) Double display (8) Special pattern
Electric power system	Miniaturization of monitor panel for concentrated control of substations (Substation--Supervisory controller +H-100+CRTctr+CRT)	1. Constant monitoring of substation system 2. Flicker of condition change of circuit breakers	(2), (3), (4), (6), (7), (8)
Electric power system	Supervisory control of terminal substation (Supervisory controller+CRTctr (with file memory)+ CRT)	1. Monitoring of distributed line 2. Display of trouble	(3), (6), (8)
Chemical process	I/O use of DDC system (Process+H-100+CRTctr+CRT)	1. Data display 2. Diagram (flow chart, block diagram, graph) display	(3), (4), (6), (9) Printer (10) Alarm
Management	Man/machine interface of management information system (H-8500+Data multiplexing controller+CRTctr+CRT)	1. Information retrieval 2. Control of work progress 3. Management of personnel inventory	(8), (11) Communication controller

and performance of the system as well as the operational principle and special features of character display, color display, and trend graph display.

From the test results of the prototype model, it has been proved that this device can operate normally as expected and demonstrate its every unique performance.

Presently, this type of display system is produced in quantity and is employed in various industrial fields.

REFERENCES

1 *The computer display review*
Adams Associates Vol 1 August 1966

2 H S CORBIN
A survey of CRT display consoles
Control Engineering Vol 12 No 12 pp 77-83 December 1965

3 D J THEIS L C HOBBS
Low-cost remote CRT terminals
Datamation Vol 14 No 6 pp 22-29 June 1968

4 S H BOYD
Digital-to-visible character generator
Electrotechnology Vol 75 No 1 pp 77-84 January 1965

5 H L MORGAN
An inexpensive character generator
Electronic Design Vol 15 No 17 pp 242-244 August 1967

6 F W KIME A H SMITH
Data display system works in microseconds
Electronics Vol 36 No 48 pp 26-29 November 1963

7 R L ARONSON
CRT terminals make versatile control computer interface
Control Engineering Vol 17 No 4 pp 66-69 April 1970

Computer generated closed circuit TV displays with remote terminal control

by STANLEY WINKLER* and GEORGE W. PRICE

Executive Office of the President, Office of Emergency Preparedness
Washington, D.C.

INTRODUCTION

In a large interactive computer system, most users with remote terminals will have a primary interest in their own specific queries. These same users may also have a common interest in or requirement for general information which is being concurrently processed. While they can individually query the computer from their own terminals, this procedure can be time consuming and inefficient, particularly if the time of availability of the general information is not known in advance. The suggestion to explore the use of closed circuit TV was a natural one since a closed circuit TV system was available.

The use of closed circuit TV offered a number of advantages. A TV display is easy to use, and each monitor can be viewed by a group rather than by only a single individual. The inherent familiarity with TV made user acceptance almost automatic. There were cost advantages not only because the system was available but also because TV monitors and their maintenance are relatively inexpensive.

There were two disadvantages which initially caused some concern. The first was the read-only nature of closed circuit TV. This was overcome pragmatically by allowing TV viewers to interact with the system by telephoning any questions to the Display Control Operator who could obtain and input the replies to these queries for subsequent display and also by co-locating the TV displays with a remote interactive terminal wherever possible. The second disadvantage was the possible exclusion of remote users who did not have access to the closed circuit TV. This generated the requirement that general information, such as that displayed on the TV monitors, should be able to be directed to any, or all, remote terminals.

* Now with the Systems Development Division, IBM Corporation, Gaithersburg, Maryland.

The system described in this paper was developed within the constraints imposed by the equipment available for the terminals and by the characteristics of the available computer. The design was intended to satisfy an operational need, but the resulting system contains considerable generality and flexibility. In the second section, we state the design philosophy underlying the development; in the third section, the system is described; in the fourth section, the software, a display sub-program, callable from any main program, is discussed; in the fifth section, the performance of the system is briefly outlined; and in the last section, brief mention is made of possible improvements and future developments.

Subsequent to the completion of the work described in this paper, our attention was called to the work of Bond, et al., at the Carnegie-Mellon University.[1] They describe an interactive graphic monitor program for use in a batch processing computer system with remote entry. Their system, although considerably different from our system, is nonetheless of general interest.

DESIGN PHILOSOPHY

The system was developed over a four month period to meet a specific operational need. This schedule dictated that only currently available equipment would be used, since hardware development or modification was not feasible in the time available. We wanted to display, on an available closed circuit TV system, a combination of pre-stored data, results of computations and data inserted from a remote terminal.

The system had to be simple to operate and convenient to use. A permanent record of each display was required, as well as the ability to review each display prior to placing it on the closed circuit TV. The length of time during which a display remained visual was to be under the control of the Display Control Operator who was to have the option of aborting any display, either prior to its appearance on the TV circuit or at any time after it was displayed.

It was important that the desired information be displayed in a timely fashion and that the design of the system should not limit the number of closed circuit TV monitors. Desirable features of the system included the capability to replace only a portion of a display as well as the capability to display a fixed pattern or table with changeable data. Some form of emphasis at the option of the Display Control Operator was considered useful.

It was desired to write the handler for the displays in such a manner that any user program written in FORTRAN, COBOL or any other higher-level language, could supply results of computations, data and information for the display. Finally, it seemed desirable to write the program in a modular fashion in order to facilitate the introduction of additional features or improvements suggested by experience in actual operation.

SYSTEM DESCRIPTION

The system was developed for implementation on the Office of Emergency Preparedness' UNIVAC 1108 digital computer. The computer has four banks (262,000 words) of main high-speed core storage. The operating system used on the 1108 is EXEC 8 which has the capability of operating in real time and demand modes as well as in remote batch and local batch modes. The OEP computer system currently handles up to 15 or 20 low-speed remote terminals (e.g., teletype) in an interactive demand mode as well as a number of high-speed terminals (operating at 2400 or 4800 baud). The low-speed remote terminals are connected to the computer by voice telephone links interfaced with acoustically coupled modems and the high-speed terminals use standard data sets.

The teletype was selected as the display control terminal because it combined the advantages of low cost and ready availability, and met the requirement of providing a hard copy of each display shown on the closed circuit TV monitors. The teletype was connected to the computer in the usual way through a telephone line. Although the operator at the Display Control teletype would manage the information to be displayed, the main program, which generates the displayable material, could be initiated from any remote terminal, low speed or high speed.

The video signal necessary to drive the closed circuit TV system is tapped off the Computer Communication Inc. CC30 cathode ray tube display terminal. This terminal consists of a Sony TV set, a keyboard, a buffer memory, and associated power supplies. The CCI terminal is directly coupled to the computer through a 1600 baud line. The signal is transmitted to the TV studio where the video distribution to the closed circuit TV monitors is made.

The actual distance from the CCI terminal to the TV studio was about 75 feet, but transmission over distances of several hundred feet is feasible using RG 59/U coaxial cable. Four lines connect the CCI terminal to the studio. The first line carries the video signal, already mentioned; the second and third lines are for the vertical and horizontal drive pulses; and the fourth line, which is optional, is a standard twisted pair of wires for voice communication. With this voice circuit, the operator at the Display Control terminal can provide "live" voice narration or pre-recorded audio tape messages for any display. A diagram of the system is shown in Figure 1.

The fact that the CCI terminal contained a TV receiver as its display element, made it relatively simple to obtain the useful video signals required. The display consisted entirely of alphanumeric information. The quality of the display was very satisfactory and the picture obtained on the closed circuit TV monitors was also of very good quality. Figure 2a is a photograph of the CCI terminal equipment including a display on the cathode-ray tube, and Figure 2b is a photograph of the same display on the closed circuit TV monitor. The picture of the TV monitor screen was taken using a 4 second shutter speed. The normal TV set jitter is clearly visible, but has no effect on the legibility of the display.

Although the keyboard of the CCI terminal could have been used to introduce data into the closed circuit TV display, it was not actually used in our system since

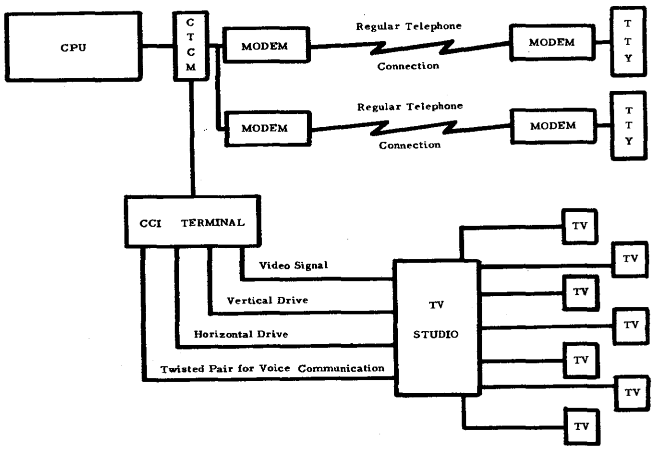


Figure 1—The display system

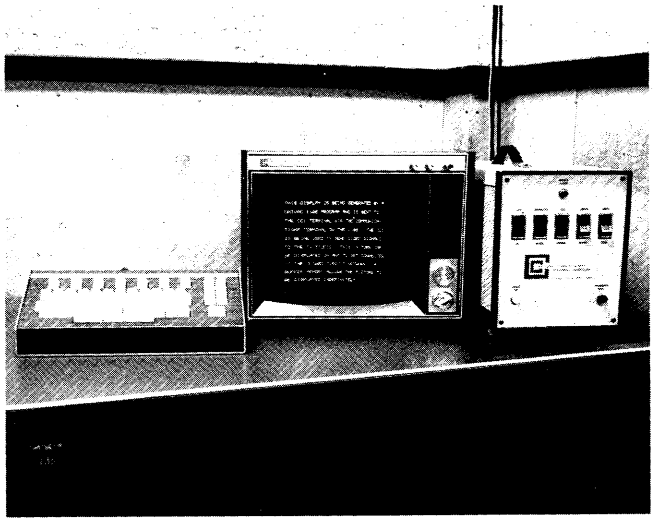
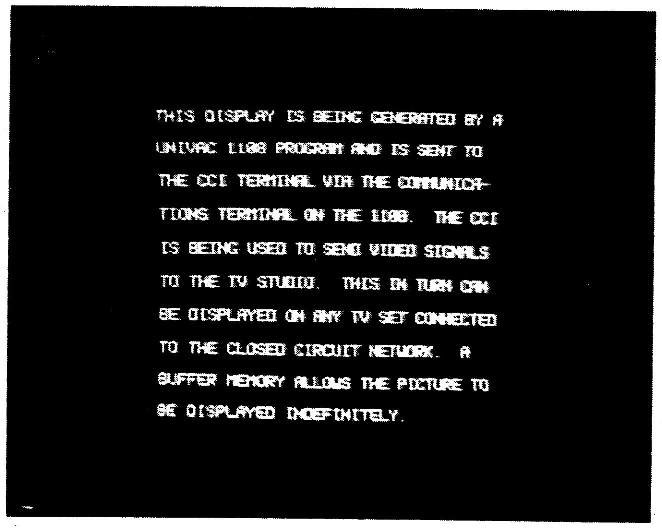


Figure 2(a)—CCI terminal equipment with computer generated display on TV set

it would have bypassed the Display Control teletype and not have furnished the required hard copy. The last two lines on the face of the cathode-ray tube of the CCI terminal were not used for the computer generated display in order to allow for variances in the picture size on the TV monitors. Probably one line would be sufficient protection, and the other line could be used for special one-line messages.

The only significant difficulty encountered in developing this system was in achieving hardware compatibility. Some experimentation and adjustment was required to



```
THIS DISPLAY IS BEING GENERATED BY A
UNIVAC 1108 PROGRAM AND IS SENT TO
THE CCI TERMINAL VIA THE COMMUNICA-
TIONS TERMINAL ON THE 1108. THE CCI
IS BEING USED TO SEND VIDEO SIGNALS
TO THE TV STUDIO. THIS IN TURN CAN
BE DISPLAYED ON ANY TV SET CONNECTED
TO THE CLOSED CIRCUIT NETWORK. A
BUFFER MEMORY ALLOWS THE PICTURE TO
BE DISPLAYED INDEFINITELY.
```

Figure 2(b)—Photo of computer generated display as seen on closed circuit TV monitor

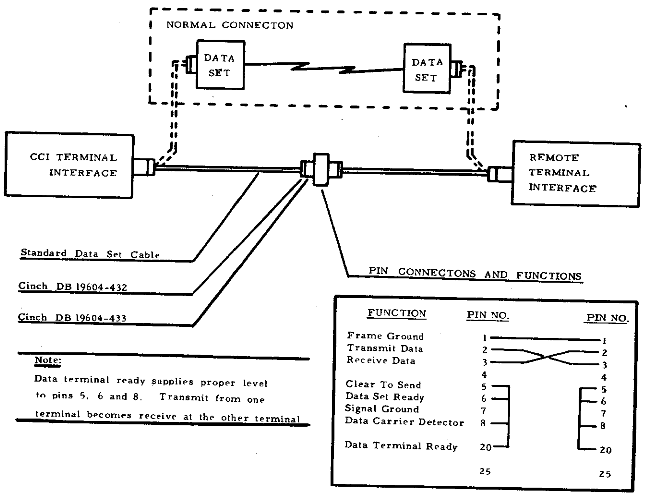


Figure 3—Circuit to bypass data set

obtain proper synchronization of signals between the CCI terminal and the TV studio. The direct coupling of the CCI terminal to the 1108 computer required bypassing the normal data set interface. This was accomplished by designing a data set bypass circuit. The circuit diagram is shown in Figure 3.

A minor, but important, problem was the format of the data to be displayed. Any of the programs written for the 1108 computer can be used to provide data for display on the CCI terminal. Most of these programs are written to furnish an edited output of 120 to 132 characters per line. The line length is 72 characters on a teletype and 40 characters on the CCI terminal. For a suitable display on the TV monitors available to us, the data had to be reformatted to fit on 20 lines of 36 characters each. The number of lines and the line length will vary somewhat among closed circuit TV systems.

THE DISPLAY SUB-PROGRAM

The sub-program is written in assembler language and contains approximately 500 lines of code. The system requires that a main program, before it may call the display sub-program, initialize the data or information to be displayed. This output data is stored in a buffer containing a maximum of 720 characters, formatted in 20 lines of 36 characters each, and must be in FIELDATA computer code (octal). Since the CCI terminal requires ASCII code, the sub-program contains a FIELDATA to ASCII conversion table. The main program is any user program modified by adding an option to permit bypassing the old output instructions and substituting for them the display sub-program output instructions. Except for formatting, these output

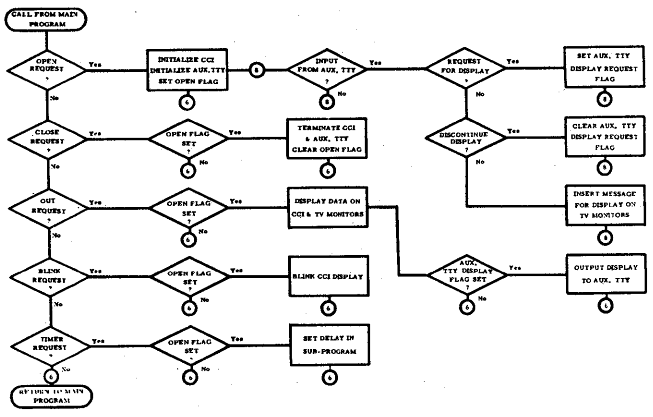

CALL FROM MAIN PROGRAM
OPEN REQUEST ?
Yes
No
INITIALIZE CCI INITIALIZE AUX, TTY SET OPEN FLAG
INPUT FROM AUX, TTY ?
REQUEST FOR DISPLAY ?
SET AUX, TTY DISPLAY REQUEST FLAG
CLOSE REQUEST ?
OPEN FLAG SET ?
TERMINATE CCI & AUX, TTY CLEAR OPEN FLAG
DISCONTINUE DISPLAY ?
CLEAR AUX, TTY DISPLAY REQUEST FLAG
OUT REQUEST ?
DISPLAY DATA ON CCI & TV MONITORS
INSERT MESSAGE FOR DISPLAY ON TV MONITORS
BLINK REQUEST ?
BLINK CCI DISPLAY
AUX, TTY DISPLAY FLAG SET ?
OUTPUT DISPLAY TO AUX, TTY
TIMER REQUEST ?
SET DELAY IN SUB-PROGRAM
RETURN TO MAIN PROGRAM
6
8

Figure 4—Flow diagram of display sub-program

instructions are essentially the same for all main programs. A flow diagram of the display sub-program is shown in Figure 4.

In the sub-program there are a number of external calls or functions which enable the operator to control the system and exercise the various options available within the system. The first of these calls is OPEN, which performs the initialization of the CCI terminal display and the initialization of the auxiliary teletype (aux TTY), if this feature is included. The OPEN call also establishes the entry of the system into the real time mode in the computer. The call, CLOSE, takes the system out of the real time mode and terminates the CCI terminal as well as the auxiliary teletype. Thus the CCI terminal and hence the closed-circuit TV displays operate in the real time mode. The control teletype operates in the customary demand mode.

The OUT, or output, call is the one which determines whether the information on the control teletype will be transmitted to the closed circuit TV monitors or not. If the output is requested (approved) by the Display Control Operator, then the information is displayed on the CCI terminal display, sent to the auxiliary teletype, if required, and the appropriate video signals are transmitted to the TV master control for distribution to the monitors.

The call, BLINK, provides the means to cause each new display to turn on and off for a fixed number of blinks when the display first appears. This method is used to call attention to the fact that a new set of data or information is being displayed. Blinking can also be omitted at the discretion of the operator. The TIMER call permits the automatic sequencing of a series of displays with a predetermined time delay in milliseconds between each individual display. It is also possible to retain each display on the TV screens until the next display is available, or, after a pre-set time, to remove the display and allow the screens to remain dark until the next display.

Another function, currently available, enables the operator to add information to an existing display or to remove a portion of the data on a display. This feature was developed to permit the display of tabulated data with the capability of updating data while retaining the tabular format.

The simple and modular character of the sub-program allows easy modification and straightforward addition of desired features. The approach adopted here was to retain an essential simplicity and to add only such features as experience in operational use clearly showed to be desirable.

PERFORMANCE

The system operates interactively permitting data or information displays to be prepared and selected for viewing on closed circuit TV as requested and is under the full control of an operator. A main program which either contains or generates the data to be displayed must be initiated and accessible to either a teletype (control TTY) or a remote batch terminal. The main program retains complete control over the operation of the system and performs all data handling and computations. Any data verification or modification must occur within the main program. The display sub-program only activates the display on command. In the operational system we developed, control is exercised with a display control teletype. This teletype could be located anywhere, and is acoustically coupled to the computer by a regular telephone. The data and control flow within the system is shown schematically in Figure 5.

The operation of the system is straightforward. A request for data from the main program is made at the display control teletype. The main program then enters the requested data, which must be a single screen or page of data, into the output buffer. At the same time, the data is transmitted to the control teletype where the data can be reviewed. At the control teletype, the operator can make a decision either to display or reject the data sent for review. A "go" decision is made by typing G and a "no-go" decision by typing X. A "go" instruction from the teletype transfers control of the data to the display sub-program which then clears out the previous display and initiates the real time mode display of the data on the CCI terminal.

Simultaneously, the video signal from the Sony TV set in the CCI terminal is transmitted to the TV studio and from there distribution is made to the TV monitors in the closed circuit TV system. Television broadcast of the data could be accomplished in a similar way.

In addition to accepting or rejecting, for display, the data from the main program, the operator at the control teletype can insert an addition to a display or originate an entire message for display. If auxiliary teletypes are permitted access to the system, the request (an OPEN call) from the control teletype (for data) also activates the auxiliary teletypes. The display sub-program then continues to poll the auxiliary teletypes for an input or request and acts on any inputs or requests received. The polling continues until a CLOSE call terminates the connection to the auxiliary teletypes. The auxiliary teletypes may request transmission of the current and subsequent displays, or request the discontinuance of the displays. They may also transmit messages for display.

The system described here was tested operationally over a 12 hour period in a dynamic situation during which information was continuously received and data values were rapidly changing. Ten TV monitors and four auxiliary teletypes were used. A fixed schedule for the display of general information was established, in this case, the first ten minutes of each hour. Each of the ten groups (one for each monitor) had a five minute time slice during which their special data requests were displayed. Interruptions to display messages were made at the discretion of the Display Control Operator. Each group had direct telephone access to either the Display Control Operator or an auxiliary teletype and could easily submit a request for data or information update. The test successfully demonstrated the capabilities of the system, satisfying the requirements of the user groups.

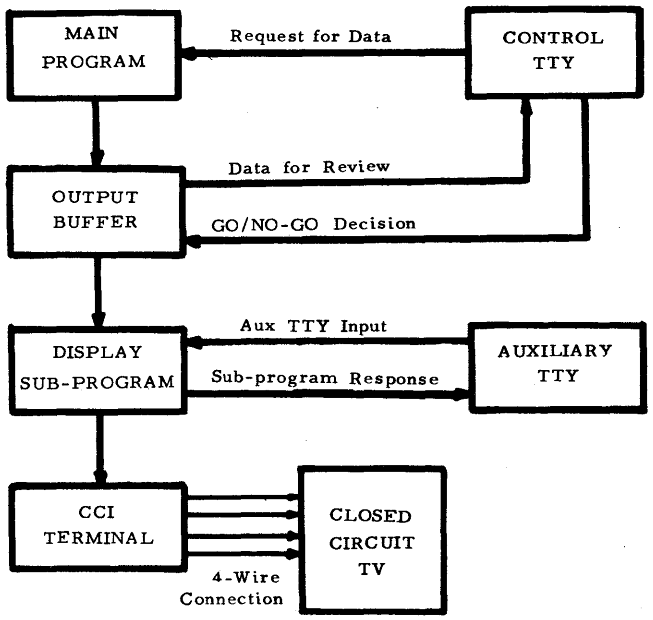


Figure 5—How the system works

FUTURE DEVELOPMENTS AND IMPROVEMENTS

A number of improvements to the system have been considered and, should the need arise, would be incorporated into the system at a future date. Among the more significant new developments are the ability to include graphic material in addition to the alphanumeric, the use of overlays which can be selectively introduced and removed, and the introduction of special messages from any telephone. It also seems worthwhile to have the capability to queue a series of displays and then be able to select any of them in an arbitrary order.

The addition of a graphic capability would permit the introduction of maps and the presentation of output data in the form of curves, and would remove much of the present limitations on the type of information to be displayed. It would also exploit the ability of a TV monitor to present actual pictures together with data and text. It seems easy enough to introduce pictures or even live action using a standard closed circuit TV camera. However, formatting the data and text to appear properly with the picture has not yet been worked out.

ACKNOWLEDGMENTS

We want to express our appreciation to all of our colleagues who assisted and encouraged this effort. In particular we want to mention Richard Vaughan whose help was essential in developing the connection to the main program and to James Johnson for his help with the equipment.

REFERENCE

1 A H BOND J RIGHTNOUR L S COLES
An interactive graphical display monitor in a batch-processing environment with remote entry
Comm ACM Vol 12 No 11 pp 595-603 607 1969

The theory and practice of bipartisan constitutional computer-aided redistricting

by STUART S. NAGEL*

Yale Law School
New Haven, Connecticut

THE THEORY

The theory behind bipartisan constitutional computer-aided redistricting essentially consists of three normative criteria which any systematic legislative redistricting scheme probably ought to seek to achieve. First, the redistricting system ought to be feasible such that it keeps computer time and other costs down to a minimum. Second, the system ought to provide legislative districts that will be approximately equal in population per representative so as to satisfy the Supreme Court's equality criterion, and the districts should be so shaped as to satisfy other legal requirements that relate to contiguity. Third, the system should consider the impact of the resulting districting on incumbents and on party balance so as to minimize the unhappiness which redistricting might otherwise produce for political leaders and for diverse political viewpoints.

These three criteria are summarized in the title of this paper. The adjective "computer-aided" refers to computer feasibility. It also emphasizes that the computer aids in redistricting like an elaborate desk calculator and does not do the redistricting itself. The adjective "constitutional" refers to the federal and state legal requirements with regard to equality and contiguity. The adjective "bipartisan" refers to promoting mutual party interests rather than ignoring the partisan impact that all redistricting inevitably has.[1]

Computers can usefully supplement traditional hand methods of redistricting. This is so because the computer when adequately instructed has great (1) accuracy, (2) speed, (3) versatility to satisfy many criteria simultaneously including legal and political criteria, (4) ability to break deadlocks by facilitating political compromises, (5) ability to minimize disruption to incumbents, (6) inexpensiveness relative to the quality of the results, and (7) flexibility to allow for local variations and special considerations. The key questions in this paper relate to what criteria should the computer instructions seek to satisfy and to what extent have various redistricting programs addressed themselves to those criteria.

Satisfying computer feasibility

A logical approach to computer-aided redistricting if one had unlimited computer time and funds available might be to (1) establish an overall criterion of goodness; (2) have the computer generate every possible combination of precincts or census tracts into a given number of districts in the area to be redistricted; and then (3) apply the optimizing criterion to each of the districting patterns in order to determine which districting pattern maximizes the criterion.

Assuming agreement could be obtained among lawyers and politicians on a composite optimizing criterion, such an approach would lack computer feasibility. With any realistic number of precincts or census tracts larger than 40, the number of different districting patterns into which they could be made quickly becomes astronomical and infeasible to handle.[2]

A simple alternative to trying all possible combinations is to (1) start with the prevailing districting pattern; (2) move each precinct or census tract from the district that it is in into every other district; (3) each time a move is made check to see if the districting has been improved in light of the optimizing criterion; and (4) each time an improvement is made use that districting pattern as the one to be improved upon until no further improvements can be made.[3]

This method can avoid making a high percentage of moves that would be made in the all-combinations method by inserting into the computer program certain

* On leave from University of Illinois

prerequisites that must be met before a move can even be checked against the optimizing criterion. For example, no move of a precinct from its present district to another district will be made if the move will cause either the district *to* which the precinct is moved or the district *from* which the precinct is moved to become non-contiguous, such that one could not go from any point in the district to any other point in the district without leaving the district. Likewise, no precinct will be moved from its present district if it is the only precinct or unit within the present district thereby destroying the district and decreasing the number of districts. In addition, the district from which a precinct is moved must be different from the district to which the precinct is moved.

The above system of moving each precinct from its present district to every other district in order to obtain successive improvements can also be supplemented by simultaneously trading a precinct from one district for a precinct from another district. Every pair of precincts gets an opportunity to be involved in such a trade provided the above-mentioned prerequisites with regard to contiguity, multiple-precinct districts, and diverse districts are met. When a trade is attempted, the resulting redistricting combination is checked against the legal and political optimizing criterion to see if an improvement has been made just as in the single precinct moving approach.

Alternatives to the moving-and-trading approach other than the all-combinations approach include such techniques as the pie-slices approach of Myron Hale,[4] the diminishing-halves approach of Edward Forrest,[5] and the transportation algorithm of Weaver and Hess.[6] A comparative analysis by Michael Strumwasser emphasizing the computer feasibility aspects of these alternative approaches concluded: "The generalized swapping algorithms (moving-and-trading), following closely the human approach to such a problem, offer a better solution than either geometric allocation (the pie-slices and diminishing-halves approach) or mathematical programming (the transportation algorithm). While the latter two approaches are aesthetically satisfying, the simplifying assumptions are violated in practice."[7]

One additional aspect of computer feasibility relates to the use of optical input and output. Edward Forrest has advocated the use of optical scanners to read maps as input into the computer,[8] but this clearly seems to be less economically feasible than relying on the Census Bureau tapes which provide (for each census tract or enumeration district) information on population, longitude, latitude, and other miscellaneous information. Additional clerical work, however, is needed (1) to show what precincts touch each other precinct and if desired (2) to convert census tract boundaries and information into political precincts. Forrest also recommends maps as output, but the cost would be far higher than an output which says District 1 consists of Precincts A, B, and C, and District 2 consists of Precincts D and E, and so on. From that verbal information, one can easily draw district lines on a precinct map showing what precincts are joined together in the same district.[9]

Satisfying the legal requirements

The legal requirements which any computer-aided redistricting scheme should satisfy consist of equal population per district and generally contiguity within each of the districts.

Equality of population can be measured in a variety of ways. The crudest way, although sometimes quite dramatic, is to present the ratio between the most populous single-member district and the least populous single-member district in the area being redistricted. This simple approach obviously ignores all the information available about the population of the non-extreme districts. At the most complex end of a continuum of equality measures would be such esoteric figures as the squared geometric mean[10] or the inverse coefficient of variation.[11] These complex measures have no legal standing in that they have never been cited as appropriate for measuring equality in a published court decision or a statute.[12]

The most favorably cited measure of equality in the literature is to (1) divide the total population of the state or area to be districted by the number of districts or seats in the legislature in order to determine the ideal population per district or per representative;[13] and (2) determine by what percentage the population of each actual district deviates from this ideal population.

If the percentage deviation from ideal for any district is more than a few percentage points, then the districting probably represents a violation of the equal protection clause of the Constitution and the democratic notion of one man, one vote. Thus in the most recent Supreme Court case dealing with the equality standard, Missouri's congressional districting was declared unconstitutional even though no district deviated from the ideal by more than 3.13 percent. Justice Brennan delivering the opinion of the Court stated that the "standard requires that the State make a good-faith effort to achieve precise mathematical equality."[14]

It should be noted that no matter how low the average deviation is if there is even one district that has a substantial percentage deviation from the ideal, the whole districting will probably be held unconstitutional. This must be recognized in writing the optimizing criterion even though mathematicians find it more aesthetic to minimize or maximize averages.[15]

Contiguity of districts is the second legal requirement. It is usually stated as a requirement in state constitutions or state statutes, or prevails as a matter of custom with minor exceptions, although it is not a U.S. Supreme Court requirement. A district is contiguous if one can go from any point in the district to any other point without leaving the district.[16] Contiguity is sought for many purposes including the purpose of (1) simplifying redistricting by eliminating many alternative combinations of precincts, (2) enabling legislators to have easier access to their constituents, (3) decreasing partisan gerrymandering, (4) encouraging people of similar interests to be together in the same district, and (5) making districting patterns more understandable and more aesthetically appealing.

A district can be contiguous and not compact, and likewise it can be compact and not contiguous. Compactness can either mean being geographically like a circle or a square,[17] or it can mean having its people clustered close together regardless of the shape of the perimeter of the district.[18] Compactness is not a legal requirement. In fact the Supreme Court recently said "A State's preference for pleasingly shaped districts can hardly justify population variances."[19] Likewise the Court disparaged the value of population compactness as well as geographical compactness by saying "to accept population variances, large or small, in order to create districts with specific interest orientations is antithetical to the basic premise of the constitutional command to provide equal representation for equal numbers of people."[20]

In spite of the importance the Supreme Court has given to equality and the non-importance it has given to compactness and in spite of the importance the states have given to contiguity, a number of redistricting programs heavily emphasize compactness at the expense of equality and do not at all guarantee contiguity.[21]

Satisfying the political requirements

It can be demonstrated that if the precincts, census tracts, or other building blocks out of which districts are made are small enough, then virtually perfect equality can be provided and still allow room for taking political interests into consideration. Given this leeway, it seems reasonable to expect the politicians to want computer-aided redistricting to (1) minimize disruption to incumbents, and to (2) facilitate political compromises.

Disruption to incumbents can be legally minimized by the following techniques: (1) the prevailing districting plan can be used as a starting point rather than starting from an undistricted map of the state; (2) districts that already come sufficiently close to the ideal population can be removed from the redistricting; (3) one can select building blocks from which districts are built with the knowledge that these units will not be broken into smaller pieces; (4) no move will be consummated if the new districting is merely equal in value to the previous one rather than an improvement as measured by the optimizing criterion; (5) redistricting can be done to satisfy the equality requirement and then district lines can be drawn so as to make the number of districts dominated by the Democrats or Republicans as equal as possible to the number before the redistricting.

Political compromises can be facilitated in various ways if the computer redistricting scheme inputs information on the number of Democrats and Republicans in each precinct or building block. For example, the computer can quickly show the Democrats and Republicans what is the maximum number of districts which they could each dominate given the Court's equality requirements and the partisan information. From these outermost positions, both sides can work in toward a compromise. Once the equality requirement has been met, the computer simply shifts from an equality optimizing criterion to a Democratic or a Republican optimizing criterion in order to reveal those outermost positions.[22]

Some compromises might require that districts in certain sections of the state be drawn to favor the Republicans up to a specified point and that districts in other sections be drawn to favor the Democrats up to a point while providing court-required equality. The computer can aid in this kind of politically-oriented redistricting, but only if it has been programmed to provide for such a political option.

An option can also be exercised within the computer program to make the percent of districts dominated by the Democrats (or Republicans) as close as possible to the percent of Democrats (or Republicans) in the state. Doing so provides a kind of proportional representation without the complicated voting procedures which are usually associated with proportional representation.[23]

Finally, the computer also facilitates political compromises by quickly providing information on the partisan composition of the districts in various tentative redistricting plans.[23a]

In the most recent relevant Supreme Court decision with regard to political considerations, the Court said, "Problems created by partisan politics cannot justify an apportionment which does not otherwise pass constitutional muster."[24] The implication is that if the reapportionment otherwise passes constitutional muster, then problems created by partisan politics and legislative interplay can be legitimately considered.[25]

In some states or areas, an additional political requirement for redistricting might relate to minimizing the negative reaction of minority ethnic groups like blacks or Spanish-speaking Americans. Just as the computer can attempt to provide proportional representation to the Democrats and Republicans, it can also attempt to provide proportional representation to minority ethnic groups by seeking to have the percentage of districts which they dominate equal to their percentage of the population within the state.

William Below and Michael Strumwasser have prepared computer programs that do seek to minimize disruption to incumbents and facilitate political compromises.[26] Other programs, however, have lacked any attempt to consider their partisan effects although some have been labeled non-partisan. Labeling a redistricting program non-partisan does not make it non-partisan if the results change the partisan balance of power as they are likely to do. The non-partisan label has merely meant that the computer program so labeled ignores the partisan effects of its work, and thus cannot facilitate political compromises or minimize political disruption.[27] Robert Dixon in his comparative analysis of alternative computer approaches particularly emphasizes the importance of political sophistication and understanding the political impact of computer-aided redistricting.[28]

THE PRACTICE

A computer program that was written in 1964 to satisfy the requirements of computer feasibility, constitutionality, and bipartisanship has thus far had some limited applications which might be worth reporting. Further applications are anticipated after the 1971 state legislatures convene and decide on the general procedures they intend to follow in redistricting the 50 states for congressional and state legislative purposes.

The first application of the bipartisan constitutional program consisted of experimental runs made to convert 90 downstate Illinois counties from 21 districts down to 18 districts.[29] Using counties rather than precincts or census tracts as the building blocks out of which to make districts greatly limited the flexibility to maneuver. Under current constitutional standards units smaller than counties would be a court-ordered requirement.[30] Nevertheless the redistricting was able to convert the original 21 districts, in which 8 violated the Illinois constitutional requirement of no more than 20 percent deviation, into 18 contiguous districts in which none violated the Illinois constitutional requirement. This conversion took only 81 seconds of computer running time.

After meeting the Illinois constitutional requirement, the program generated various politically-oriented districting patterns. They ranged from a pattern in which the Democrats obtained a majority in only 22 percent of the 18 districts, up to a pattern in which the Democrats obtained a majority in 39 percent of the districts. This ability to provide alternative political patterns could have facilitated the Republicans making some concessions in the downstate area in return for related concessions by the Democrats in the Chicago area. Instead both parties moved so slowly trying to develop political compromises that the constitutional deadline passed and an at-large election had to be held to choose the state legislature.

The next application of the bipartisan constitutional program was by William Below working for the California Assembly Committee on Elections and Apportionment in 1965.[31] According to his report, "The program was applied to Assembly districts in Los Angeles, Orange, San Francisco, and Santa Clara counties. In San Francisco, the use of the program served only to verify that a particular set of goals was not obtainable. In each of the other counties, plans were produced which the committee staff considered good enough to submit to committee members and the affected incumbents. Three out of the thirty-one districts in Los Angeles (those which underwent the greatest change), were included in the assembly bill almost exactly as the program produced them. The plans for Orange and Santa Clara Counties were slightly changed on the advice of the incumbents."[32]

Below also reports that "Members of the committee staff with no data processing experience became proficient at specifying the initial plans, weights, and desired proportions necessary to use the program."[33] Below's version of the program added increased flexibility by (1) allowing different political goals for each district rather than just having an overall political goal for the area to be redistricted, (2) interspersing moving and trading rather than doing all the trading after completing all the moving, (3) developing techniques for translating census areas into political areas, (4) simplifying the information inputed to preserve contiguity, and by (5) translating the program into the Fortran programming language.

The third application of the bipartisan constitutional program was by C-E-I-R, Inc., for the Illinois Republican Party in 1965. Norman Larsen, who handled the application for C-E-I-R, reported that the politicians were more interested in being quickly and accurately provided with useful information on the characteristics of the districts in a variety of tentative redistricting plans than they were in having the computer produce an optimum output.[34] The Republican Party was the minority party in the Illinois legislature at that time, and its districting patterns were less influential on the final result than the Democratic districting patterns. Nevertheless the Republican Party leaders did buy $31,000 of computer redistricting consulting services, and they appeared to be satisfied with what they obtained.

Like Below in California, Larsen in Illinois made various changes in the program to take into consideration the fact that thousands of townships and other units were used to create the districts rather than a mere 90 counties as in the original example. The contiguity checks in particular were streamlined. Continuous intermediate output was generated to allow a monitoring of convergence toward an optimum. Time saving conditions were also introduced to eliminate various kinds of moves that were not likely to lead to an improvement.

The experience received in applying the program in California for a bipartisan state legislative committee and the experience in Illinois for the minority political party showed that the bipartisan constitutional computer-aided redistricting approach is feasible from the three viewpoints of computer technology, law, and political realism. The approach would clearly be less meaningful if it failed to satisfy fully these three essential criteria. It is anticipated that other versions of this basic program and approach will be developed and applied to the 1971 redistricting which is about to get under way across the country.

REFERENCES

1 A computer program that seeks to achieve all three of these criteria and their sub-criteria simultaneously is described in:
NAGEL
Simplified bipartisan computer redistricting
17 Stanford Law Review 863 1965

2 This was the finding of Garfinkel and Nemhauser as described in the masters thesis of:
M STRUMWASSER
A quantitative analysis of political redistricting
UCLA School of Business Administration 1970

3 The essence of this precinct moving system was first developed by:
H KAISER
An objective method for establishing legislative districts
10 Midwest Journal of Political Science 200 1966

4 M HALE
Representation and reapportionment
Dept of Political Science Ohio State University 1965
See also:
M HALE
Computer methods of districting
In Reapportioning Legislatures H Hamilton ed
Charles Merill 1966

5 E FORREST
Apportionment by computer
4 American Behavioral Scientist 23 1964

6 J B WEAVER S W HESS
A procedure for nonpartisan districting: Development of computer techniques
73 Yale Law Journal 288 1963
A revised version of their approach is described in:
CROND INC
REDIST: Program description and user manual
National Municipal League 1967

7 M STRUMWASSER
Op cit Ref 2 at p 19
See Also:
RISE INC
Proposal for the reapportionment of the California Assembly
March 1970
Available from 417 South Hill Street Los Angeles California
Because the Hess and Weaver approach begins from many starting points rather than from the existing districting pattern, their approach can produce many equally equal plans which further decreases the feasibility of their approach.

8 E FORREST
Electronic reapportionment mapping
Data Processing Magazine July 1965

9 Providing a map as output does not seem worth the extra cost over providing words as output. Along related lines, however, providing a system whereby one can move population units through a computerized typewriter and receive immediate feedback may be quite useful to politicians who want to do their own districting, but who want the computer to provide information on the alternatives they suggest rather than have the computer suggest alter natives. See:
C STEVENS
On the screen: Computer aided districting
Conflicts Among Possible Criteria for Rational Districting 40-49 National Municipal League 1969

10 H F KAISER
A measure of the population quality of legislative apportionment
62 American Political Science Review p 208 1968

11 G SCHUBERT C PRESS
Measuring malapportionment
58 American Political Science Review p 302 1964

12 G BAKER
Implementing one man, one vote: Population equality and other evolving standards in lower courts
Conflicts Among Possible Criteria for Rational Districting p 24-39 32 National Municipal League 1969

13 Where there are multi-member districts involving different numbers of representatives in a state, the courts talk in terms of population per representative rather than population per district. Someday, however, the courts may recognize that one voter who is a member of the majority interest group in a district with two representatives and 2000 people has more political power than one voter in a dis-

trict with one representative and 1000 people, since the first voter can determine who two representatives will be.
J BANZHAF III
Multi-member electoral districts—Do they violate the "One man, One vote" principle
75 Yale Law Journal p 1309 1966

14 KIRKPATRICK v PREISLER
394 U S 526 530 1969
Although Kirkpatrick dealt with congressional districting, its standards would probably equally apply to state legislative districting. In fact the equal protection clause which applies to state districts, more specifically requires equality than Article I of the Constitution which applies to congressional districts.

15 The programs developed by Bill Below and Henry Kaiser only optimize averages rather than force outliers under a maximum cut-off.
W BELOW
The computer as an aid to legislative reapportionment
An ALI-ABA course of Study on Computers in Redistricting
American Law Institute 1965
H KAISER
Op Cit Ref 3

16 For two intra-district units to touch or be contiguous they must share part of a common line no matter how small, not merely a common point. Legal boundaries of land areas adjacent to bodies of water normally extend over a portion of the water at least for the purpose of court and police jurisdiction if not for the purpose of ownership, and these extended boundaries should be used in determining contiguity not the shoreline.

17 E C REOCK
Measuring compactness as a requirement of legislative apportionment
5 Midwest Journal of Political Science 70 1961

18 WEAVER HESS
Op Cit Ref 6

19 KIRKPATRICK v PREISLER
394 U S 526 536 1969

20 Ibid p 533

21 WEAVER HESS
Op Cit Ref 6
FORREST
Op Cit Ref 5
The Kaiser program also lacks a guarantee of contiguity:
KAISER
Op Cit Ref 3

22 For a discussion of the computer programming that is involved in both the equality optimizing criterion and the various political optimizing criteria, see:
NAGEL
Op Cit Ref 1

23 James Weaver impliedly defines "non-partisan" as providing proportional representation by stating, "A procedure blind to politics should provide a random opportunity for changes in the party in power, hopefully approximating the partisan ratio in the area."
J WEAVER
Fair and Equal districts: A how-to-do-it manual on computer use
p 3 National Municipal League 1970
The Weaver-Hess program, however, makes no attempt to provide proportional representation (i.e., to approximate the partisan ratio) other than by blind hope.

23a An additional option can easily be added to the program whereby the computer seeks to maximize the number of districts in which neither the Democrats nor the Republicians have more than 53 percent of the two-party vote. This would please political scientists who feel competitive districts make for more responsible representatives, but might displease politicians who prefer greater margins of safety.

24 KIRKPATRICK v PREISLER
394 US 526 533 1969

25 Thus far, the Supreme Court has refused to declare political line drawing unconstitutional where equality was provided, WMCA v Lomenzo, 382 U.S. 4 (1965). Someday, however, the Supreme Court might say that the equal protection clause is prima facie violated if the districting plan gives the minority party a substantially lower percentage of the districts than the percentage of minority party members in the state. This would be the case, for example, if the minority party constitutes 40 percent of the people in the state, but the lines are drawn so that the minority party dominates only 15 percent of the districts.

26 W BELOW
Op Cit Ref 1
M STRUMWASSER
Op Cit Ref 2
Both of these programs are based on the Nagel program, Op Cit Ref 1

27 This is true of the programs of Weaver-Hess, Op Cit Ref 6; Forrest, Op Cit Ref 5; Hale, Op Cit Ref 4; and Kaiser, Op Cit Ref 3. It is also true of the more obscure programs of
C HARRIS
A scientific method of districting
9 Behavioral Science p 219 1964
J THORESON J LIITTSCHWAGER
Computers in behavioral science: Legislative districting by computer simulation
12 Behavioral Science p 237 1967

28 R DIXON
Democratic representation: Reapportionment in law and politics
pp 527-35 Oxford University Press 1968

29 NAGEL
Op Cit Ref 1

30 KIRKPATRICK v PREISLER
The court said "we do not find legally acceptable the argument that variances are justified if they necessarily result from a State's attempt to avoid fragmenting political subdivisions by drawing congressional district lines along existing county, municipal, or other political subdivision boundaries." 394 U.S. 526,533, 1969.

31 BELOW
Op Cit Ref 1
Information on the California application also comes from 1965 correspondence between William Below and this writer

32 Ibid p 7

33 Ibid

34 In a report from Norman Larsen to Jack Moshman of C-E-I-R, Inc. dated Nov 1 1965

"Second generation" computer vote count systems—Assuming a professional responsibility

by COLBY H. SPRINGER and MICHAEL R. ALKUS

Systems Research Inc.
Los Angeles, California

In recent years, the costs of holding an election have increased substantially. Not only are more voters being added to the registration rolls, but rising numbers of candidates and issues have bloated ballot sizes in city, county, and state elections. Not surprisingly, the most promising solution to the growing vote tabulation problem has seemed to lie in the application of automatic data processing technology. Few would disagree that computers should be able to perform the vote counting task more economically and efficiently than any of the other systems which have been designed for that purpose. Because vote counting appeared to be a relatively simple tabulation process, it seemed to be an ideal example of the kind of job computers do best: counting quickly, cheaply, and accurately.

In terms of the number of elections successfully completed, computer vote counting systems have turned in a satisfactory record. Both major punched card voting equipment manufacturers have "run" hundreds of elections. The exceptions, however, have been notable. Fresno, California, has experienced consistently late counts; Los Angeles suffered nationwide publicity with its own tardy returns; and Detroit endured two late counts after a history of successful elections with lever-type voting machines. It has been the latter cases—the only two large-scale applications of computer vote counting systems—which have resulted in widespread questioning of the concept of computerized elections by the public.

There appear to be three reasons for the failures. First, punched card ballots are themselves susceptible to damage from handling. Second, centralized elections of any kind—particularly those involving the introduction of a new technology—require precise planning and good management. Third, the manufacturers' computer software systems have contained major design weaknesses. Although any of these problem areas might result in delays in any election, the duration of such a delay is magnified in a large election because of the sheer volume of the processing task.

Several election delays have resulted from punched card damage. In the Flint, Michigan, 1970 general election, cards soaked in heavy rains failed to feed—even after being baked in the high school oven. Similarly,

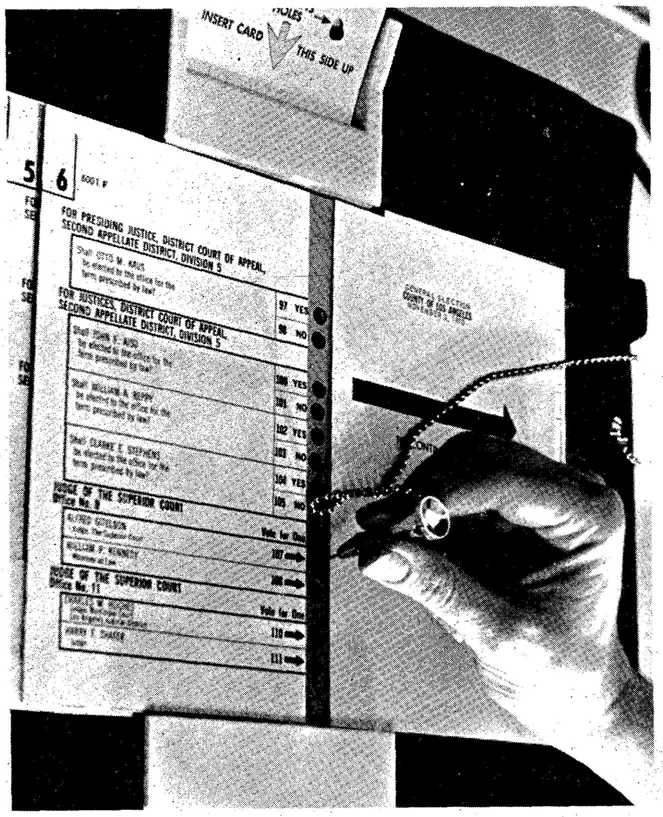


Figure 1—A typical punched card voting machine in use. The stylus is used to punch through a mask and template to the ballot inserted below

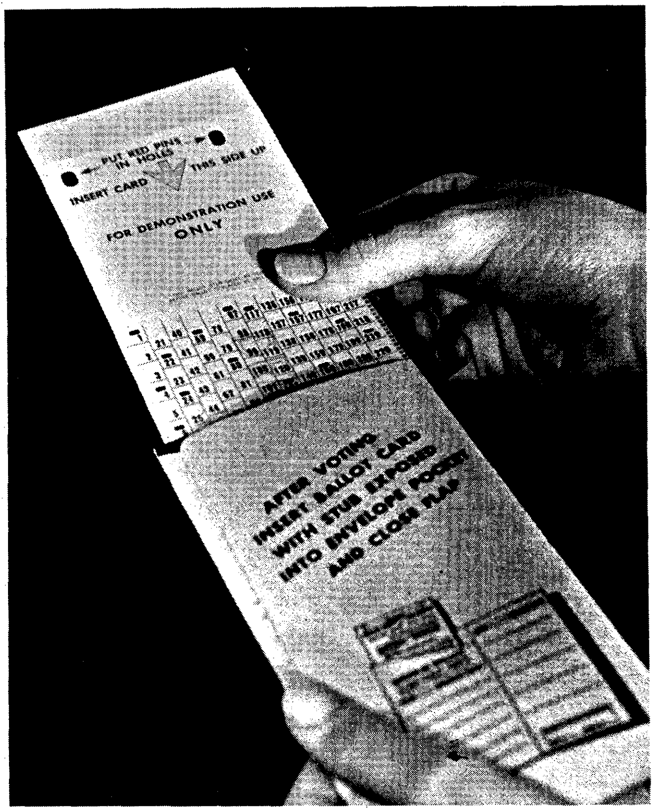

PUT RED PINS IN HOLES
INSERT CARD THIS SIDE UP
FOR DEMONSTRATION USE ONLY
AFTER VOTING INSERT BALLOT CARD WITH STUB EXPOSED INTO ENVELOPE POCKET AND CLOSE FLAP

Figure 2—The punched card is inserted in a ballot envelope. The attached stub is then removed and given the voter as a receipt

punched card ballots in Detroit, after being stored in an uncontrolled environment for several rainy days before the election, failed to feed because of edge softness. The Los Angeles County audit team suggested consideration of alternate media, but concluded that the economy of punched cards makes them the only medium currently practical.[1]

Reports from both Los Angeles County and the City of Detroit 1970 primary elections pointed to specific planning and management weaknesses as one cause of election failures. These experiences indicated that problems in transportation, ballot protection during transit, inspection, and the performance of inexperienced personnel were related to inadequate preparation for the massive demands posed by the sheer logistics of these operations. Further, management can be hampered by election laws written for paper ballots. Although most punched card voting machine manufacturers include some management services as part of their support package, legal requirements demand that election officials remain in control of election night operations. This fact alone can result in situations where decisions must be made by election officials who do not have a complete understanding of the underlying data processing system. In one case—Detroit—contractor personnel were precluded by legal interpretation from being present at the counting sites. On the other hand, counties with strong data processing management, a history of successful data processing applications, and close coordination between data processing and election officials have had good elections. One of many good examples is California's Santa Clara County.

In the Los Angeles example, a massive planning and management effort (mounted after long delays had been experienced in the June, 1970 primary) produced a general election count without significant delays.[2] Previous planning oversights were avoided by transferring major responsibilities to the County Administrator's Office supplemented by a significant amount of outside consulting talent. According to one report, the extra effort raised the cost of the November election to more than $2.00 per ballot.

But another basic contributor to the major election system failures has been the computer software design itself. This paper focuses on such system weaknesses and suggests major design changes. It identifies the need and suggests improvements for design and programming standards, auditing procedures, and support documentation.

CURRENT DESIGN

It was the issue of security that first brought attention to the weaknesses of computer vote count systems. Our original research efforts focused on the possibility of deliberate program modification as a potential security threat.* This preliminary research, including the construction and analysis of a model vote count system, uncovered several potentially disastrous weaknesses. The November, 1969 Intellectron report[3] identified the following:

- The manufacturer-supplied operating system was vulnerable to modification and would permit changes without requiring access to the user-developed vote tabulation programs.
- Detection of a vote bias routine—one which would change the ballot image—would be difficult during production without destroying the election results. Furthermore, such a routine could be written

* This and subsequent investigations were directed toward the IBM Votomatic System as implemented in Los Angeles County. All commercially marketed punched card voting systems currently in use are forms of Votomatic. Although IBM has not released a list of the firms now marketing these systems, seven have been identified.

without altering linkage editor totals or core-image length.

- A valid "logic and accuracy test" would require either a sophisticated computer program or prohibitive amounts of computer time—perhaps several times as much as that required for the actual vote count itself.
- Many techniques of computer vote fraud require the access of only one person and, at most, an operator and a programmer.
- None of these techniques would be detected by a casual observer, even if he had an extensive background in data processing.

These findings indicated that the possibilities for system failures—not only due to fraud, but to unintentional error as well—had been seriously underestimated by their designers.[4] The subsequent elections proved the prediction to be correct.[5,6,7]

The analysis of these problems in the Votomatic system may, however, prove valuable in constructing "second generation" vote count systems. While such a detailed analysis of current systems has not been made, it is now possible to identify some of the factors which should be included in future systems.

At least part of the problem derives from a basic misconception on the part of some of the users and systems developers themselves. Although the job may *seem* to be relatively uncomplicated, in fact, the challenges offered to a conscientious designer are anything but trivial. First, the accuracy attained by such a system must be absolute and unquestionable. If the slightest doubt as to the reliability of the results is tolerated, one of the prime goals of the system—maintenance of public confidence in the sanctity of the ballot—is forfeit.

Second, the system must operate under the most exacting timing requirements. In the short period between the determination of candidates and issues and election night, ballots must be designed, printed, and distributed; workers must be recruited, trained, organized; and program modifications must be completed, tested, and certified. The system must deliver results promptly and reliably on election night—and it must do so the first time.

Third, the system must be extraordinarily secure. Computerization means centralization: where counting was once conducted by precinct workers scattered throughout many locations, a computerized system necessitates consolidation of election night activities, and a corresponding threat to security. Centralization of counting processes reduces the obstacles to a would-be vote embezzler—where previously he would have had to bribe or coerce large numbers of election workers, under the new systems he need only alter the central counting mechanism. For this reason, that mechanism must contain adequate multiple safeguards.

Last, because of the high stakes involved in election tabulation, the entire vote counting process must be reconstructible. Although extensive audit procedures are not normally provided in non-financial systems, audit trails are needed here on several grounds. One clear reason is that the threat of deliberate tampering is reduced when audit procedures increase the likelihood of such activity being discovered. Another is that accidental error is less likely to escape unnoticed, and the program fault which allowed it to go uncorrected, when control totals are automatically compared during the course of tabulation. Perhaps most importantly, thorough audit procedures can answer challenges to the validity of results, and thus act to maintain public credibility.

Clearly, the system which fulfills all of these requirements performs far more than a simple tabulation task. And yet, in spite of the obvious need for the safeguards and redundancies required by any high-security system, few of those now used in vote counting contain even one of the performance characteristics described above. The system outlined in this paper is designed to meet the special demands which the past several years have shown to be made of such systems. The technology is now present to implement the system described for small- and medium-sized elections; and it is reasonable to believe that a rigorous research effort could yield one capable of handling very large elections, as well.

DESIGN CONCEPTS

A vote tabulation system must be designed to satisfy three criteria: assurance of ballot security; performance of tabulation functions at minimum cost; and prompt issuance of election results.

Of these criteria, security is clearly the most critical. An electronic vote tabulation system must provide special safeguards against both accidental and intentional alteration of the results. These safeguards can be provided by internal audit procedures, by full operational testing of all peripheral (as well as the main counting) programs, and by external inspectors and observers. A system can be designed to take advantage of the high speeds of third generation computers to provide more thorough reports, more complete auditing than is currently available, and to allow independent inspection of any of its procedures or results.

A system properly designed to reduce the cost of ballot preparation and vote tabulation can achieve economies unavailable through any other tabulation method. A well-planned system can reduce the costs of tabulation until they become a small fraction of the overall expense of conducting an election, and without compromising security. Similarly, the introduction of computers to the ballot preparation stage can offer significant savings and increase the reliability of pre-election activities.

SYSTEM CHARACTERISTICS

A vote tabulation system includes the computer tabulation and auditing programs and documentation, and a planning guide. A specific "second generation" system described below would offer a series of computer programs for the preparation, tabulation, and audit of an election. It is significantly more extensive than current systems, since ballot design and auditing aids are included—although both features have been recommended to improve security and reduce potential errors, neither has yet been incorporated in an operational system. The introduction of computer techniques early in the process would offer significant economies and a higher degree of security than is possible when computers are used only in the tabulation process.*

A specific system design best illustrates the features requisite to this type of computer program. The design suggested below is the result of a preliminary research project conducted to provide such an illustration. The programs have not been tested.

Ballot preparation

Recent experiences have shown that the preparation of ballots is a crucial step which, if performed improperly, can prevent the rest of the system from functioning as intended. Further, the communication of ballot formats to the tabulation program has often been difficult, creating errors in the actual vote count. These difficulties can be avoided by using the computer to prepare the ballot assemblies, sample ballots and other reports, and having the ballot preparation program itself communicate format to the tabulation program.

In the proposed system, the ballot assembly program receives as input appropriate precinct data, the political units in whose borders each precinct lies, the candidates, the offices for which the candidates are running, and the rules by which candidates' names are rotated on the ballot. The program then produces the design of the actual pages to be placed in the voting machine as well as the ballot assemblies for all precincts.

The same program can produce the sample ballots to be mailed to voters, and proof sheets for comparison with ballot assemblies distributed to the precinct.[8] Other reports generated by the program include ballot layout summaries for candidates, parties, election officials, and impartial observers. This innovation is included to assure candidates that the precinct components of their vote have been correctly identified for their office.

Internal checks in this program assure completeness and consistency of the output.

The ballot preparation program also produces machine-readable tables representing the ballot patterns for each precinct. These tables are read directly into the computer on election night by the ballot tabulation program.

Ballot tabulation

Ballot tabulation programs and procedures are designed to provide speedy reporting of election results with a complete audit trail and documented reconstruction of the count.

The first phase in ballot tabulation is media conversion. The ballots, having been delivered to the computation center, are unpacked, scanned for physical damage (e.g., hanging chad) and read into a computer which copies them onto magnetic tape. While the ballots could be read directly into the ballot tabulation program in card form, media conversion permits preliminary audit checks and produces a magnetic tape which can be used by later programs for generating various reports. The program includes a restart capability so that a core dump can be taken during the counting process. Furthermore, special provisions are made for recovering from card jams—a problem which often plagues such systems. The special cards used by most voting systems are particularly susceptible to card jams in high speed card readers. Because of equipment design characteristics, it is often difficult to determine which card caused the jam, and to identify the last card to be read. Consequently, votes may be counted twice—or not counted at all—during recovery from a card jam. To prevent such occurrences, the media conversion program is designed to restart without duplication every time a card jam occurs. Restart points are frequently printed out on the console, so that, following any card jam, the data stream may be reinitiated from the last ballot whose proper reading is confirmed.

The second phase of the tabulation process is the

* This specific change has been strongly recommended by the audit teams for both the Detroit and Los Angeles elections. See References 1 and 2.

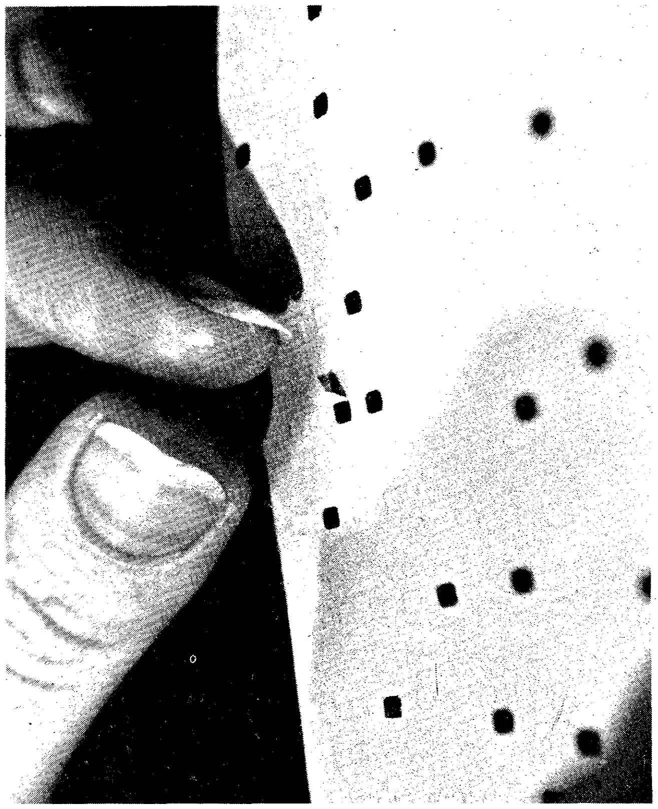


Figure 3—A common difficulty with the punched card as a ballot medium is the incomplete punch. The resultant hanging chad must be removed by inspectors on election night

actual vote counting and report generation. The vote tabulation program receives the magnetic tape containing the card images of the ballots and the pattern tables indicating ballot formats as generated by the ballot preparation program. The tabulation program reads the ballots and totals them by precinct and by office. Intermediate totals are maintained (1) on a direct access disk file, (2) on a sequential access tape file, and (3) in an intermediate totals report produced on the printer. The program prints out restart points so that an observer may call for the count to be stopped while a core dump is taken. This dump can be compared with one taken before the start of processing to verify the security of the vote count program itself. Once the dump has been taken, the program can be restarted at the point where counting was interrupted. In addition to displaying the contents of core memory, all data on disk files and intermediate counters can also be printed out.

In addition to the reports listing the results of the election by office and by precinct, the election night vote tabulation program leaves numerous independent audit trails. These trails—on the disk pack, on the tapes produced by the vote count program, and on independent tapes produced by the media conversion program—are all retained and used in the audit programs after the election count.

Audit

Additional programs are provided to form an audit of the election. This audit insures that all ballots (including those held at supply centers as well as those actually distributed) have been accounted for, and that all intermediate totals produced in the various stages of the tabulation process balance. In addition, quality control totals of undervotes, overvotes, and mutilations are maintained to insure that results satisfy certain minimal reasonability requirements. The audit programs also provide a report noting all disqualified and unmarked ballots in each race.

Recount system

In addition to the audit prepared and run by election officials, the system makes provision for recounts performed for or by candidates, parties, or impartial observers. Three kinds of checks may be performed at the option of the party requesting the recount.

A manual recount simply allows the parties involved or election officials to retabulate the votes by counting holes on the cards themselves or their printed images.

A machine card recount provides for recounting from the cards themselves (rather than tape-resident card images) by computer program.

A tape recount is done by computer on the actual tape used in the tabulation program.

A recount using a distributed copy of the card image tape can be run on any computer using any program chosen by the challenger.

HARDWARE AND SOFTWARE CHARACTERISTICS

The technical design of the system represents a departure from previous electronic vote tabulation systems. Previous systems have employed low-level, machine-oriented programming languages in an effort to maximize computational efficiency. Leaving aside the judgment about the wisdom of that decision for large municipalities, it appears clear that for smaller elections, computational efficiency is not of prime importance. Clarity, security, and ease of modification are much more vital to the design of such a vote tabulation system. Consequently, all programming should be done in a high-level, user-oriented programming language, such as COBOL, or preferably PL/1. While these languages do not generate the most efficient machine code, a knowledgeable programmer with no previous familiarity with the program would be able to understand the functioning of an intelligently-organized and well-commented PL/1 vote tabulation program.

This choice offers several advantages. First, the program could receive wide distribution to all interested parties who could then verify its accuracy to their own satisfaction without actually running the program. Second, modifications to the program to accommodate different equipment, different recording requirements, or different organizational features of an election would be made much easier using a high-level language and modular program construction.

SOFTWARE

The system design identified in preliminary research which would fill the requirements of such a system included eleven specific programs. They were;

- Ballot Assembly Preparation Program
- Assembly Audit Report Program
- Media Conversion Program (card to tape)
- Logic and Accuracy Test Program
- Vote Tabulation Program
- Inspector Audit Program A (for media conversion program)
- Inspector Audit Program B (for vote tabulation program)
- Ballot Inventory Reconciliation Report Program
- Ballot Image Print Program
- Core Dump Comparison Program
- Election Results Analysis Report Program

The system, unlike present systems, is heavily oriented toward audit and proof programs designed to check each operation through an independent program. The system uses the computer throughout the election process from ballot design through a quality control analysis of the counted ballots.

CONCLUSION

Regardless of what system is used, computers will continue to be employed in the tabulation of election returns. If insecure, poorly-documented and hastily-modified programs are used, the scare headlines, public investigations, and official indignation we have already witnessed will continue to be aimed at the computer profession, and we need only await the first major election scandal for the threat of government regulation to become a reality.*

At issue is the larger question of professional responsibility. At present, no mechanism exists for the self-regulation of the industry, and no universal code of 'good practices' has been defined. In a highly sensitive, highly visible system like vote counting, this oversight stands as an invitation to disaster. Even if we will not, the public will hold the profession accountable for its responsibility to society. If members of the profession are to undertake responsible assignments, those who assume these assignments must be held accountable for their work.

The vote count system described here is offered as an example of the kind of design approach which assumes social responsibility as a basic design requirement. It may not perform the vote counting task more quickly or inexpensively than do present systems, but does address the larger—and potentially vital—issues of security and credibility.

It is hoped that the kinds of problems it treats will be met by future vote counting system designs, and that the same kinds of issues will be recognized wherever they occur in the computer industry.

REFERENCES

1 H H ISAACS ET AL
Final report County of Los Angeles votomatic computer system audit
Isaacs Associates Inc Los Angeles California Volume II p 10 1970

2 ECONOMICS RESEARCH ASSOCIATES
Evaluation of planning and performance for the November 1970 general election
Economics Research Associates Los Angeles California pp II-1 V 1-2

3 J FARMER C H SPRINGER R STANTON M J STRUMWASSER
Vulnerabilities of present computer vote count systems to computer fraud
Intellectron International Inc Van Nuys California November 1969

4 R L PATRICK A DAHL
Voting systems
Datamation May 1970

* Eleven states have already forbidden the use of the computer in election tabulation. The California legislature is investigating the licensing of computer programmers and operators.

5 *And the winner is . . . ? Computer is the loser in Michigan election*
Wall Street Journal August 6 1970

6 *Primary highlighted by discrepancies, irregularities*
Computer World October 7 1970

7 *State, local probes ordered into ballot mixups, delays*
Los Angeles Times June 4 1970

8 ECONOMICS RESEARCH ASSOCIATES
Report to the Los Angeles County election security committee
Economics Research Associates Los Angeles California August 1970

Evaluation of hardware-firmware-software trade-offs with mathematical modeling

by H. BARSAMIAN and A. DeCEGAMA

The National Cash Register Company
Hawthorne, California

INTRODUCTION

The 70's are expected to become the beginning of the era of intellectual maturity of the information processing systems. Until now the main challenge was the adaptation of problems to the computers. The new challenge is the adaptation of computers to the problems.

Presently the engineering know-how and the state-of-the-art of the technology have reached a stage where the design of a "good" computer is no longer a mystery. However, the understanding and the efficient utilization of computers in terms of cost/performance/user need is far from satisfactory. The deficiencies in present computing systems are due mainly to the neglect of comprehensive disciplines and formal techniques for the integral design, analysis and evaluation of computer systems and user tasks. As a result, the computer user usually pays more than he should and/or gets less than he needs. To bridge the gap between designing a "good" computer and its efficient utilization, a major reorientation in the design and evaluation philosophy of information processing systems should take place. With optimum cost/performance indices as the ultimate goal, the main design guideline must be systems adaptation to user problems through tailored computer hardware and software.

From the user's standpoint, this is the ideal approach. For the computer manufacturer however, this approach has constituted a cascade of technological, design, manufacturing and maintenance problems which have compounded to produce almost prohibitive systems costs. These costs impact negatively on potential users, frequently surpassing the "psychological barrier" of the buyers, particularly in the commercial market. Recent technological advances, specifically semiconductor LSI technology, seem to promise a positive solution to most of the manufacturer's problems indicated above. LSI technology has the potential of bringing the practical capabilities of the computer manufacturer closer to the needs of the user. However, the satisfaction of the user's needs in terms of cost/performance efficiency of their computing installations continues to be a fundamental and complicated task. The specifics of user applications, as well as the functional and control characteristics of the entire computing system are highly interrelated. They need to be analyzed systematically and evaluated quantitatively as characteristics of a consolidated information processing system.

This paper emphasizes the necessity of employing the integral hardward-software approach for design and evaluation of computing systems. Three relevant topics are discussed:

1. Formulation of the task and the strategy for designing more efficient computing systems through hardware-firmware-software trade-offs.
2. The analytical tool applied for quantitative evaluation and optimization of these systems, namely the mathematical modeling techniques.
3. An example of the implementation of the proposed methodology.

HARDWARE-FIRMWARE-SOFTWARE TRADE-OFFS

Task formulation

The analysis of contemporary computing systems of conventional architecture (the overwhelming majority of present general purpose computers) reveals that performance inefficiencies are caused mainly by three principal factors:

1. Excessively complex, costly and often poorly designed software—caused by the neglect of comprehensive scientific disciplines in the design of both systems and applications software. As a result, the quality of the computer software depends heavily upon the subjective human factors of the designer.
2. The logical primitiveness of the computer hardware. A typical "general purpose" computer has only the adder as its sole decision-making mechanism. All other system components, e.g., memories, transmission channels and peripherals, do not have any intellectual capabilities. It is the burden, then, of the software to schedule and control the functions of these components, as well as perform all kinds of data manipulations in the CPU (factually in the adder).
3. The large gap between machine and programming languages. Software interpreters or compilers perform the enormous task of language translation and program conversion from the procedure oriented source languages to the object machine codes. As a result, a significant portion of computer power in terms of time and space (as much as 70 percent) may be spent for program compilation. The user, whose only interest is the solution of his problems, pays for this wasted computer power.

Thus, the epicenter of problems seems to be related to the hardware-software ratio used for performing processing functions within the computing system. Computing systems with improved cost/performance efficiency can be designed by redistributing the processing functions performed in the hardware and software. Such a redistribution must be balanced and evaluated quantitatively within the range of the optimum values of two target functions: cost and throughput (response time for on-line or real-time systems) for the whole computing system. Accordingly, an overall system design and evaluation strategy can be postulated:

1. Delegate more computational procedures and algorithmic functions to the computer hardware (both in the main-frame and the peripherals) by hardware-firmware-software trade-offs (HFS). These trade-offs must be aimed toward the following goals:
   (a) Simplify the structure and the design of the software systems.
   (b) Allow more space and time for the executive and user programs within the given system configuration.
   (c) Narrow the gap between the machine and source programming languages by incorporating more procedural contents and functional complexity into the primitives of the machine language; thus, allowing the computers to be driven primarily by input data rather than by programs.
2. Apply analytical techniques for evaluation and optimization of proposed system architectures. A mathematical model with acceptable accuracy, rather than a simulation model, appears to be a suitable and efficient tool for the fulfillment of these tasks. Two peculiarities of the mathematical model must be underlined:
   (a) The mathematical model considers the computing system as a complex stochastic model of queueing, resource sharing, scheduling and allocation processes.
   (b) The parameters of the computing system as well as the characteristics of the processing environment and the user requirements are considered as interrelated control and/or state variables of the model.

The strategy

Actually the HFS trade-offs are an economical problem facing the computer architect. Theoretically, any computable problem can be solved on a single bit adder operating with an infinitely long serial memory (Turing-type machine). However, any realistic computer architecture of mini-, midi-, or super-scale is the result of a compromise made between the computational speed (or the throughput) and the practicality of the systems cost. The criteria for this compromise are: (1) the satisfaction within a reasonable elapsed time of the computational needs of a certain class of users, (2) suitable man-machine communication features, i.e., programming and operating aids, and (3) an acceptable price/performance ratio. Thus, the speed-cost trade-off is the creed of the computer designer.

Contemporary computing systems can be considered as a superposition of physical and logical media. The physical medium (the hardware) is the host environment where the logic (the software) resides and carries out the information processing task. Although completely different by their physical and technological nature, the computer hardware and the software consitute together an indivisible architectural assembly designated to solve user problems. Thus, the computer system is an integral hardware-software system and speed-cost trade-offs must be optimized for the integral system, not for the hardware and the software separately. Although this may seem to be a trivial definition, its implications constitute a fundamental impact on the design and evaluation efficiency of computer systems.

In the process of computer design, the distribution of processing functions to be performed in the hardware and the software as constituents of an integral system architecture is subject to the same speed-cost trade-off rules. Traditionally the computer architect solved this dilemma based on engineering intution, experience and some superficial figures of cost and performance, assessing mainly isolated physical characteristics such as the CPU hardware cost, the add time and/or memory cycle time. Applying these and like criteria for the evaluation of computer systems performance is analogous to applying Newton's classical theory to quantum mechanics, i.e., adopting a static view of a multidimensional and highly dynamic process. The parameters mentioned above are necessary but no longer adequate in the performance evaluation of present (and future) computing systems for the following reasons:

1. The CPU hardware cost, specifically in this era of batch fabricated components technology, constitutes a very small percent of the total system cost (approximately 3 percent).
2. The add time, as any other machine instruction execution time, cannot by itself indicate system performance since most programming is done in higher level languages.
3. Faster memories (even if program codes are well optimized) do not directly benefit the user because they store more than just the user programs. Moreover, if the computing system is I/O bound, often the case in business applications, then the fast memory becomes an expensive luxury for the user.

The inadequacies of other static parameters—memory capacity, I/O transfer rate, machine word length, etc.—as system performance criteria are also obvious.

Program execution times or the number of programs executed per unit time, however, do reflect all the architectural and combined functional characteristics of the computer system. Thus, during the computer's architectural design period, the ideal speed-cost trade-offs should be evaluated and optimized on the executable program level rather than on the machine instruction level. In the case of general purpose computers, however, this is an enormously difficult if not impossible task. Such an approach seems to be more practicable on the next lower level of functional complexity i.e., on the level of independent processing sequences, algorithmic functions, routines or macros, which represent functional building blocks for both the user and executive programs (compilers, operating systems, utilities). Search, insertion and file retrieval, sorting, table reference and inversion, scanning (in compilers), stack manipulations, floating point and variable field arithmetic, trigonometric and transcendent functions, I/O interface control, links, error recovery, interrupt handling and reentry are representative sources of such functions. It is suggested that the procedure of dividing the computer's processing tasks into independent or semi-independent functional building blocks may be called ALGORITHMIC PARTITIONING.

The execution path of an algorithmic partition utilizes more of the various system resources and exercises their functional capabilities to a greater extent. Therefore, more optimized speed-cost trade-offs are achievable at this level of functional complexity than at the traditionally considered machine instruction level. Through algorithmic partitioning a noticeable software-hardware interface can be drawn, which is the necessary condition and the basic platform for efficient HFS trade-offs.

Microprogramming: the vehicle for implementation of HFS trade-offs

Microprogramming was first introduced in the early 50's as the "best way to design a computer."[5] Microprogramming, virtually, is a control discipline which allows the design of instruction execution flows and the functional sequencing of digital processors in a more systematic way. Microprogrammed interpreters can be designed for algorithmic partitions performing orderly structured processing functions. These interpreters, composed of micro-instructions and stored in special read-only (ROM) or read-write (RWM) control stores, are called firmware.

In choosing an approach for the implementation of HFS trade-offs, the following factors must be taken into consideration:

a. The importance of computer emulation, which requires the availability of additional (non-predesigned) primitive functions at the lowest possible logical level of processor's hardware.
b. The trend toward increasing the intellectual capabilities of computer hardware, which necessitiates the control of more complex data manipulation schemes and increased number of information flow paths.

Microprogramming is the most efficient technique for achieving such capabilities. Also, microprogramming offers more economical design of modern processors due to the availability of inexpensive and fast LSI semiconductor control memories for storing the coded control sequences (microprograms), and to the suitability of ordered and repetitive memory structures (rather than non-standard logic networks) to LSI implementation. These factors suggest that through microprogramming, optimum cost/performance and functionally flexible HFS trade-offs can be attained.

Although the implementation details of microprogrammed processors are outside of the scope of this paper, two aspects of control store organization, pertinent to HFS trade-offs, must be discussed: static versus dynamic and centralized versus distributed microprogramming.

One of the major benefits of using microprogramming as a control discipline is the possibility of tailoring a great number of primitive function combinations to facilitate the execution of programs and to allow more efficient emulation of different machines or languages. The use of a ROM of limited capacity for the control store, static microprogramming, considerably diminishes the functional flexibility of the microprocessor. Any microprogram modification after the original design and the microcoding is completed can be done only in an off-line fashion. The alternative is to provide a very large capacity of ROM which can accommodate all the possible combinations of primitive functions and allow microprogram switching. The estimates show that, in this case, an astronomically high number of bits are required for the control store and obviously such an approach is not practical. The use of a RWM for the control store will allow dynamic microprogram alterations under supervisory control. The microprograms now can be treated as user programs, subject to online editing and modifications, reloading and swapping, thus allowing some kind of multi-microprogramming capabiltities. This appears to be a highly desirable feature for multi-purpose emulation. Dynamic microprogramming is beneficial to the computer designer and the user. It permits accomplishment of a more flexible, more error free and more easily maintainable logic design, while the user gets a potentially more powerful processor within the given physical constraints and costs.

In a conventional microprogrammed processor a single control store is designated for all microprograms. The use of multiple control stores designated for separate microroutines or group of microroutines is another way to achieve optimum design and performance. Such an approach is feasible, specifically, with the LSI technology since the cost-capacity relationship of semiconductor memories is a nearly linear function. An efficient decentralization of the control medium can be accomplished with microroutines that are function and time exclusive. Additional factors to be considered are interface characteristics, registers, and other architectural features of the processor. The main advantages of distributed microprogramming are the achievement of a more compressed microcode, and its suitability for parallel execution or pipelining of different microroutines stored in physically separate control stores. Also, distributed microprogramming permits a higher degree of firmware maintainability and diagnosibility.

HFS trade-offs for algorithmic partitions implemented with dynamic and distributed microprogramming positively impact the overall computer system architecture by providing the following characteristics:

a. The realization of modularized and simplified design, maintenance and debugging of major software systems, e.g., compilers and operating systems.
b. Machine language which is more procedure oriented and less sensitive to the computer's hardware design specifics (it appears as machine-independent intermediate type language).
c. Better utilization of the available hardware and a more faceless logical structure due to the dynamic alterability of microinstructions. Thus, versatile and efficient emulation can be performed.

The justification of HFS trade-offs from the computer engineering and manufacturing standpoint is rationalized not only by the prospect of lowering production costs (mainly due to the advances of LSI technology) but also by substantially increasing systems throughput. The use of more complex machine instruction sets and increased numbers of processing and algorithmic functions performed in hardware-firmware will result in increased availabilty of memory space, fewer references to the slower main memories, and diminished systems interrupt overhead—hence, faster program generation and execution speeds.

These facts lead to rather an interesting conclusion, which at first seems controversial. All other conditons being equal, the HFS trade-offs allow the attainment of required systems throughput by using slower, and consequently cheaper, logic circuits, and main memories. Thus, "the faster, the better" approach for selecting the logic circuits appears not to be the axiomatic guideline for achieving increased performance of a computing system. The present circuit speeds have nearly reached their physical limits. Therefore, harmony between the execution speeds of decentralized microprocessors (the result of $\overline{HFS}$ trade-offs) and the speeds of their communications interfaces must be the prime consideration for improved performance in computing systems.

Thus, it seems apparent that through HFS trade-offs optimum integral hardware-software systems can be designed. Using firmware with its algorithmic contents and two logical interfaces—an interpretive interface with the software and an executive interface with the hardware—as a buffering medium, the ratio of hardware and software functions can be balanced. As mentioned earlier, this ratio has an overwhelming impact on the cost/performance index of a computing system, therefore, its quantitative evaluation and optimization are tasks of paramount importance.

THE MATHEMATICAL MODEL

The Mathematical Model (MAMO) used for the evaluation of HFS trade-offs constitutes a recent development in the evaluation and optimization of computer systems.[1] It can predict and optimize the performance of priority-partitioned multi-processing multi-programming systems by taking into account diverse environmental conditions, hardware configurations and characteristics as well as system control policies. The MAMO solves a system of non-linear equations correlating these parameters and obtains the desired quantitative results in a relatively short time.

In contrasting the MAMO with simulation models, factors such as complexity and cost of development, computer time and space requirements, accuracy and efficiency in obtaining the desired results must be considered. The development of a realistic MAMO is a difficult and costly undertaking. The difficulty stems from the fact that the available mathematical techniques cannot provide a sufficiently accurate solution to the complex probabilistics problems characterizing computer systems. Therefore, in order to formulate a MAMO useful for practical purposes, numerous approximations must be made. These approximations require extensive testings by simulation models as well as a thorough understanding of the stochastic processes involved. The high development cost of a MAMO is due to the fact that its formulation and verification require the development, maintenance and application of highly sophisticated simulation models which would reflect real computing systems as closely as possible.

However, once a satisfactory MAMO is developed and verified, the higher degree of operational capabilities and efficiency afforded by MAMO, as compared to a simulation model, compensates for the efforts and resources invested.

For the studies described in Reference 1, an experimental task was formulated for a hypothetical multiprogramming system. The simulation model of this task run on an IBM 360/65 required five to six hours of computer time and approximately 250K words of space; whereas, the mathematical model of the same task processed on a UNIVAC 1108 system required only 10 seconds of machine time and 32K words of space.

The superiority of MAMO is more apparent in performance optimization tasks. A simulation model produces optimized results by analyzing the values of control variables and target functions accumulated by numerous simulation runs. Consequently, the validity and the accuracy of the results are simply proportional to the number of simulation runs for the given task. By virtue of its nature, the MAMO solves the optimization task through a set of non-linear equations, by manipulation of the state and control variables characterizing the computer system being studied. The optimization process consists of the search for a set of nominal control variables on the multi-dimensional performance space which provides an extremum value (maximum or minimum) for the selected target function. In a multi-programming system having three partitions of programs, the optimum performance will be influenced by more than 30 variables (see Appendix). The number of simulation runs, then, would be over 2^{30} and would require thousands of hours of machine time (as estimated for a 360/65 system). On a UNIVAC 1108 system, the MAMO used less than one hour of computer time to solve such an optimization task.

Thus, it seems that simulation models should be used mainly as tools for developing and verifying mathematical models of computer systems. However, for complicated service disciplines and for some of the processes in computing systems which cannot be properly formulated mathematically, simulation techniques can be used advantageously by incorporating them into mathematical models. In general, studies of computer system performance evaluation and optimization are more economical and practical when conducted by mathematical models.

Figure 1 is the simplified block diagram of the MAMO.[1] The computer system's performance and cost are functions of three groups of variables:

1. Environmental parameters defining the computer system under study, mainly the physical characteristics of programs and equipment;
2. system control variables which can be varied by the MAMO to obtain optimum cost/perform-

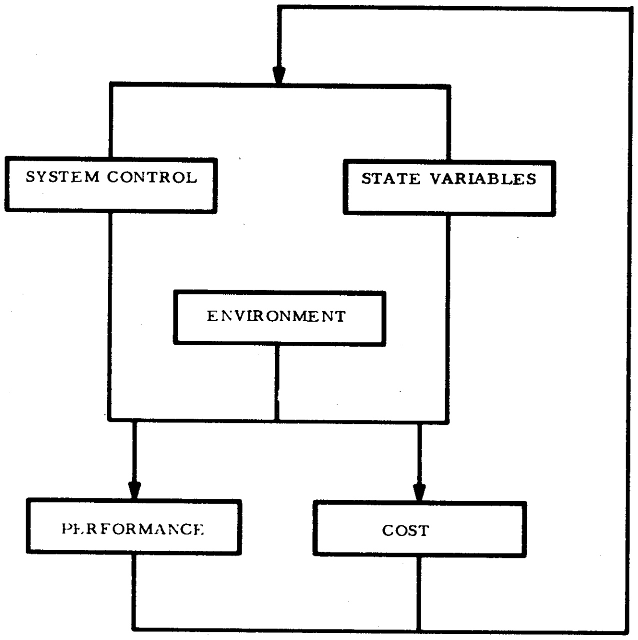


Figure 1—Mathematical model block diagram

ance balance (memory partition sizes, CPU quantum size, page sizes, etc.), and

3. state variables (CPU utilization/priority, probabilities of interrupts/priority, probabilities of compilation interrupts/priority) which influence the two main evaluation criteria—cost and performance.

The optimization procedure, shown by the feedback path in Figure 1, uses a direct search technique to define the optimum set of control variables. This means that the MAMO uses the interim solutions of each step to choose the direction of the next step toward the optimum. As a result of this optimization process, the MAMO can calculate the maximum values of the following target functions:

- Throughput for a given cost with or without constraints on response time,
- quality of service (minimum average response time) for a given throughput and cost,
- uniformity of service (minimum variance of response time) for a given cost, throughput and average response time,

and, also, the minimum value of the cost to achieve a given throughput and quality of service.

The formulation of the MAMO

The formulation of the MAMO requires an in-depth analysis of a complex queueing system in which several service processes are taking place simultaneously. The study of this queueing system is purposeful to the determination of the service times required by each program as a function of CPU time and I/O service requirements. Thus, two queueing systems must be considered: the queues of programs attempting to receive service from the CPU and the main memory, and the queues of I/O requests made by the programs. The I/O request queues must be analyzed based on the existing queues formed at each of the controllers of the file system, e.g., drum, disk, tape. The MAMO is a set of non-linear equations which constitutes the mathematical expressions of these two interrelated queueing systems. The solution of this system of equations must satisfy the condition that the utilization factor of all system components—CPU, main memory, I/O controllers and devices—must remain less than one.

The design of the MAMO is based on the assumption that the time periods between consecutive CPU interrupts should follow an exponential density function rather closely. The occurrence of CPU interrupts in a multi-programming system can be considered the result of the superposition of many random and independent events (paging, I/O file accesses, ends of programs, new arrivals, quantizing, etc.). Then the Pooled Output Theorem would seem to indicate that the distribution of inter-interrupt times is asymptotically exponential. The only question is whether known results of queueing theory involving exactly exponential density functions of times between events can be applied to the different service systems constituted by the I/O devices, CPU's and memories.

In order to apply known queueing theory results for the design of the MAMO, two factors must be verified:

1. Is the assumption on the distribution of inter-interrupt times valid?
2. Is the deviation from the exact exponential distribution tolerable?

The detailed simulation of many different systems indicates that, indeed, the results of queueing theory were applicable to the design of the MAMO. It is believed that this conclusion opens new avenues in the development of practically applicable mathematical models for complex computing systems.

The detailed description of the MAMO is out of the scope of this paper and is provided in Reference 1.

EVALUATION OF THE FIRMWARE SORT PROCESSOR WITH THE MATHEMATICAL MODEL

To obtain a reasonable justification for any HFS trade-off a conclusive cost-performance evaluation must be carried out. The evaluation of the HFS trade-offs and their effect on the performance of the whole computing system is a task of dynamic nature. The characteristics and the performance of the routines or algorithmic partitions subject of HFS trade-offs must be analyzed and evaluated under the actual conditions of the computational environment, taking into account the following major factors:

1. The computer time spent to perform the object routine in software,
2. the memory space occupied by the object software routine,
3. the user's processing environment characterized by well-defined (measured and/or calculated) input, state and control parameters.
   (Detailed listings of these parameters are provided in the Appendix.)

The objectives of the task are to determine and evaluate the dynamic values of the increased availability of computer time and memory space due to the hardware-firmware implementation of the object software routine.

The MAMO described briefly in the previous section was applied to the performance evaluation of a firmware Sort Processor (SP).[2] The SP is a functionally autonomous microprogrammed processor dedicated exclusively to sorting, thereby relieving the computer's CPU of this function. It can easily be integrated into any general purpose computer system and will behave as an intelligent peripheral controller (Figure 2). As such, the SP represents a typical example of an HFS trade-off. (See Reference 2 for detailed description of the SP.)

The evaluation of the SP is conducted by comparing the performance of the host computer system executing a sort with conventional software routines against its performance of the same function with the SP. The host computer systems studied, represent third generation, business oriented small and medium class systems with multi-programming capabilities. They are assumed to operate in a typical business environment of the following average values of major characteristics:

- program size (code): 20,000 bytes
- defined data space for execution; 30,000 bytes
- CPU time required by a program: 60 seconds
- CPU time between data file references: 1.25 seconds
- CPU time between jump instructions: 50 microseconds
- CPU time between data references: 13 microseconds
- percentage of CPU time in execution of sort routine; $S = 20$ percent

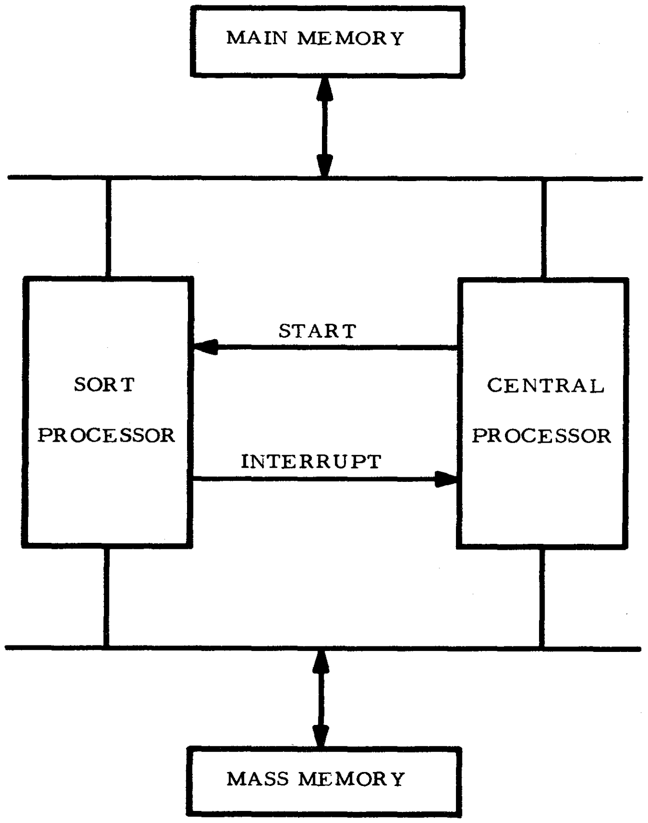


Figure 2—Computer system configuration with sort processor

- percentage of CPU time in execution of library subroutines: $L = 30$ percent
- probability of executing the sort routine from the library of subroutines: $P_s = 67$ percent
- probability of requiring I/O access per subroutine call: 40 percent.
  (Note: $S = L \times P_s$) The sort routine has been assumed to occupy 6 KB of main memory space.

The influence on maximum system throughput of the amount of processing done by the SP is shown in Figure 3 for one system studied and two different memory sizes. The limitation in performance that can be observed from the leveling off of the curves is due to a main memory bottleneck for the smaller memory system and to a disk system bottleneck for the larger memory system. However, in both cases, the throughput of the computer system with the firmware SP was improved. (In Figure 3, zero percent indicates no hardware-firmware implementation of the sort routine.)

Table I summarizes the results obtained for different types and classes of host computers operating in batch mode. Main memory sizes (256 KB) and the sorting workload (20 percent) are the same for all four systems.

TABLE I—Computer Performance Evaluation with the Sort Processor in a Business Environment of Batch Programs

SYSTEM	B_1	C_1	B_2	C_2
Throughput increase (percent)	13.2	20	9.85	14.4
Cost/Program decrease (percent)	11.6	15	8.5	11.8
Average service time/program decrease (percent)	9.9	15.5	5	10.3

NOTE: Main memory size = 256 kilobytes, percentage of sort routine usage = 20 percent

Throughput in Table I indicates the maximum program arrival rate that the system can handle; this rate is also equal to the maximum service rate that can be sustained by the system. The service rate is given by the ratio N_{pm}/T_{e1}, where N_{pm} is the average number of programs in memory and T_{e1} is the average service time/program. In evaluating the influence of the SP on throughput it should be noted that $T_{e1}=T_{c1}+T_{i1}$, where T_{c1} is the average CPU time required/program and T_{i1} is the average interrupt time/program.

The SP will affect N_{pm} and T_{e1}. N_{pm} will be slightly greater with the SP due to the increased availability of memory space, while T_{i1} will decrease due to the smaller number of I/O interrupts required per program. The use of the SP diminishes the occurrence of two types of interrupts: paging activity interrupts and, more importantly, the I/O interrupts needed to transfer the sort routine into the main memory. Thus, the efficiency of the SP depends on the value of Q defined for a system without SP as follows:

$$Q=\frac{\text{number of sort routine calls}}{\text{total number of interrupts}}$$

A higher value of this ratio yields more throughput increase for a host system having an SP.

The improvement in cost per program (C) is defined as $C=(E_o/P_o)-(E_s/P_s)$ where E_o and E_s are the computer system costs per unit of time without and with an SP, respectively. P_o and P_s are corresponding throughput values.

For further analysis and evaluation of the SP, the following characteristics of the host computer systems studied must be stated:

a. The main memory of system C2 is the fastest, followed by systems C1, B1 and B2, respectively.
b. Systems C1 and B1 have similar secondary storage configurations as do systems C2 and B2, the latter systems being more powerful.

Accordingly, based on the results of Table I, the following conclusions are valid, provided that all environmental conditions are equal: the performance improvement with SP is greater for,

- systems having faster main memories with similar secondary storage characteristics. (This conclusion is confirmed also by the results shown in Figure 4.)
- systems having slower secondary storages with similar main memory characteristics.

The increased availability of main memories for user programs—due to the release of the space traditionally required by the sort routine—results in less paging activity for all systems with SP. Therefore, the percentage of performance improvement is greater for computer systems with slower secondary storage due to the decreased utilization of slower I/O devices for paging.

Thus, Table I indicates that substantial cost/performance improvements, i.e., increased throughput,

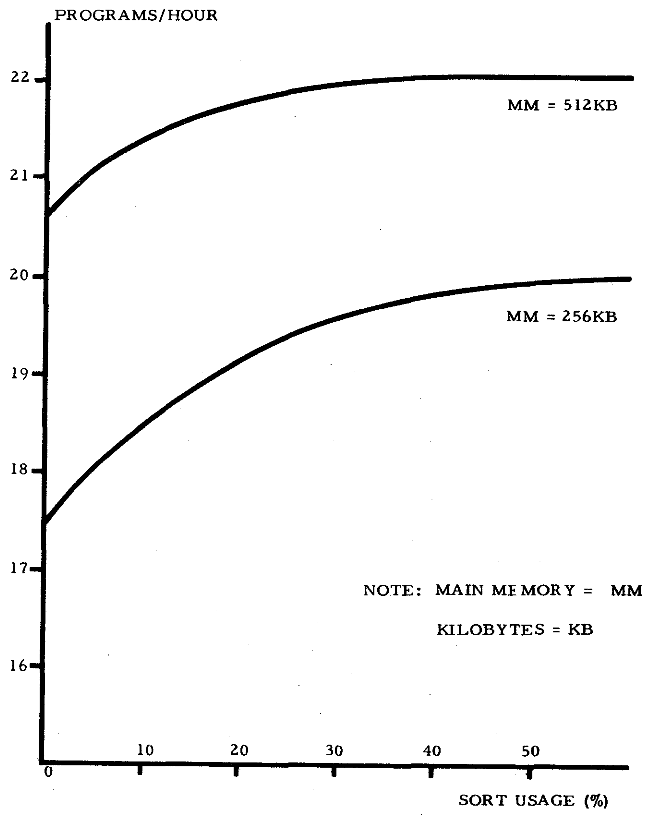


Figure 3—Throughput versus the percentage of sort usage (Batch programs)

decreased service time, and lower costs per program, are achievable for various computer system configurations with SP. The utilization of the computer equipment, i.e., CPUs, main memories, I/O controllers and secondary storage units, is also higher.

Similar results have been obtained for systems with on-line programs only and for systems with on-line and background batch programs (see Appendix for on-line program characteristics). The effect of main memory size on the performance of an on-line system having the program characteristics listed in the Appendix is shown in Figure 5. It appears that the particular system studied was I/O bound, and increasing the size of the main memory more than a certain value does not result in improved performance. However, whether the system is I/O bound or not, the addition of the SP results in a marked improvement of computer system throughput.

Finally, the results of the study as shown in the preceding table and figures demonstrate,

1. the capabilities of the MAMO in producing quantitative results about system performance,

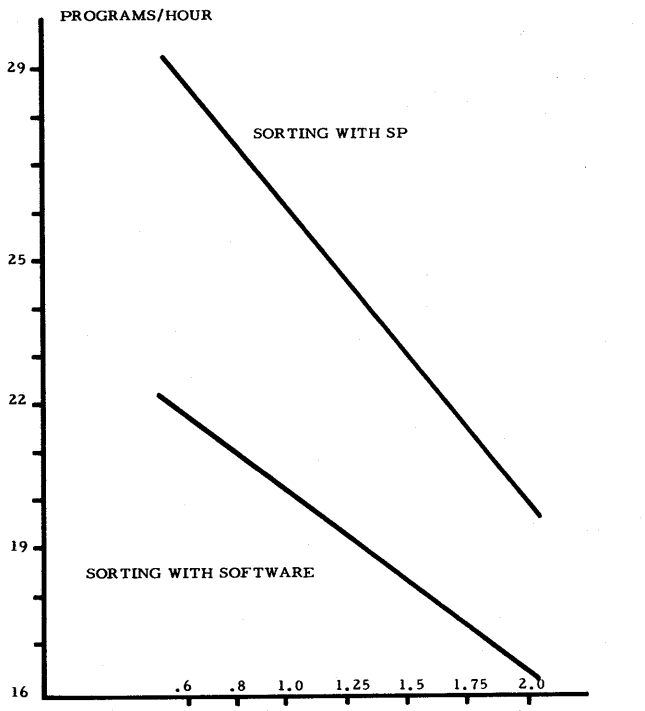


FIGURE 4. THROUGHPUT VS. MAIN MEMORY CYCLE TIME. (BATCH PROGRAMS); MAIN MEMORY SIZE = 256 KILOBYTES, SORT USAGE = 20%

Figure 4—Throughput versus main memory cycle time (Batch programs)

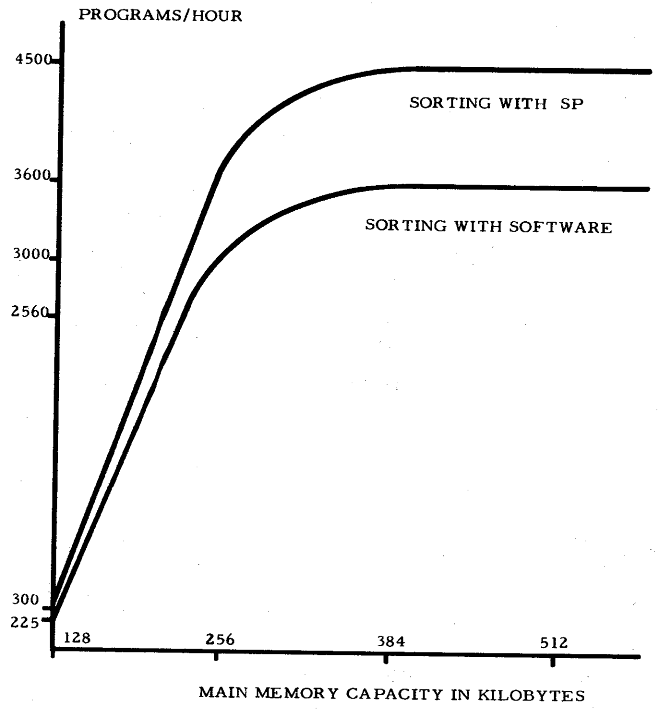


Figure 5—Throughput versus main memory size (On-line programs)

2. the usefulness of the MAMO for making decisions on new system architectures and HFS trade-offs,
3. the power of the Sort Processor for substantially improving the cost/performance index of computing systems.

It is clear that through the combined application of mathematical modeling and HFS techniques new and better computer systems can be built.

SUMMARY

In the era of LSI and increased computer usage, the importance of designing new computer architectures with optimum hardware/software balance has been discussed. Algorithmic partitioning is suggested as the proper vehicle to accomplish the hardware-firmware-software trade-offs. Carefully selected algorithmic partitions, traditionally performed by software routines, are implemented by microprogrammed hardware interpreters stored in distributed writable control stores. Selection and cost/performance evaluations of algorithmic partitions for hardware-firmware-software trade-offs are accomplished by applying a realistic Mathematical Model of multiprogramming and multiprocessing computer systems. The possible computer

architectures are evaluated under diverse environmental conditions.

As an example of this system design philosophy, a recently developed Mathematical Model[1] has been applied to the cost/performance evaluation of a firmware Sort Processor.[2] The results obtained indicate that more efficient computer systems can be designed by means of increasing the logical complexity of the hardware and by extensive use of microprogramming techniques. The results also show the great analytical power and practical utility of mathematical models for performance evaluation and optimization of computer systems. It is believed that the system design philosophy set forth in this paper will be prevalent in the years to come.

ACKNOWLEDGMENTS

The cooperation and encouragement offered by the NCR-DPD Research Department has been instrumental in the preparation of this paper.

The authors wish to express their appreciation to Mr. A. G. Hanlon and Mrs. V. J. Miller for their valuable suggestions and editorial efforts.

Mmes. A. Peralta's and E. Mackay's contributions in typing this paper deserve a full-hearted thanks from the authors.

REFERENCES

1 A DeCEGAMA
Performance optimization of multiprogramming systems
Doctoral Dissertation Computer Science Department Carnegie-Mellon University Pittsburgh Pa April 1970

2 H BARSAMIAN
Firmware sort processor with LSI components
AFIPS Conference Proceedings Spring Joint Computer Conference Volume 36 1970

3 C F WOOD
Recent developments in direct search techniques
Research Report 62-159-522-R1 Westinghouse Research Laboratories Pittsburgh Pa

4 A L SCHERR
Analysis of storage performance and dynamic relocation techniques
IBM Research Report TR-00-1494 September 1966

5 M V WILKES
The best way to design an automatic calculation machine
Manchester University Computer Inaugural Conference Proceeding p 16 1951

6 R F ROSIN
Contemporary concepts of microprogramming and emulation
Computing Surveys December 1969

7 A OPLER
Fourth generation software
Datamation January 1967

8 L L CONSTANTINE
Integral hardware/software design
Modern Data April 1968—February 1969

9 R W COOK M J FLYNN
System design of a dynamic microprocessor
IEEE Transactions on Computers March 1970

10 H W LAWSON
Programming language-oriented instruction streams
IEEE Transactions C-17 pp 476-485 1968

APPENDIX

Parameters defining machine space requirements

Operating system space
Number of compilers
Number of compiler code segments/compiler
Sizes of compiler segments
Number of data segments/compiler
Sizes of data segments/compiler
Number of library subroutines
Sizes of library subroutines
Distributions of program sizes
Distributions of data space/program

Parameters defining machine time requirements

Distributions of CPU time/program
Distributions of input times (card-reader, teletype, console, etc.)
Distributions of output times (printer, teletype, console, etc.)
Compilation/execution time ratios
Probabilities of a source card error
Probabilities of a compilation error
Distribution of CPU times between files accesses during compilation
Distribution of operator times to make files accessible to the programs
Probabilities of execution error
Probabilities of using data files by programs
Distribution of CPU times between data file accesses
Probabilities of using serial files by programs
Probabilities of serial file accesses versus random file accesses
Probabilities of disk random file accesses versus drum random file accesses
Probabilities of accessing print files (spooling) versus data files
Distribution of interrupt overhead times

Parameters that influence both time and space requirements

Distribution of inter-arrival times
Distribution of amounts of source data

Distribution of number of interactions for conversational programs
Peak number of conversational users
Distribution of CPU times between jump instructions (compilers and programs)
Probabilities of jumping to a new compiler segment during compilation
Probabilities that a jump instruction is a library call
Distribution of CPU times between data references (compilers and programs)
Probabilities of data segment swap per I/O access
Distribution of jump distances
Distribution of data access distances
Main memory address generation rate by CPU (code and data)
Distribution of program file sizes
Distribution of data per file access
Distribution of dump data
Tape start-up time
Tape transfer time
Disk rotational speed
Disk seek time
Disk transfer time
Drum rotational speed
Drum seek time
Drum transfer rate
Size of the routine or program subject to HFS trade off
Frequency of usage of the routine or program subject to HFS trade off
Hardware/software execution speed ratio of the routine subject to HFS trade off
Access times of the different memory hierarchies
Amount of data/access for the different memory hierarchies
CPU speed
Hardware costs

Control variables influencing optimum system performance

Number of CPU's
Cache capacity
Total amount of memory (main memory and bulk memory) for non-partitioned systems
Partition sizes (main memory and bulk memory) for a partitioned system
Number of disk and drum controllers
Number of spindles/controller
Number of tape controllers
Number of tapes/controller
Proportion of data files kept directly accessible to the programs as opposed to the files requiring manual intervention by the operator
Maximum data segment size (for each priority i)
Maximum program segment size (for each priority i)
Maximum amount of main memory space allowed for data (for each priority i)
Maximum amount of main memory space allowed for instructions (for each priority i)
Input buffer sizes (for each priority i)
Output buffer sizes (for each priority i)
Compiler buffer sizes (for each priority i)
Print buffer sizes (for each priority i)
Probabilities of finding a system routine or a user data segment in each level of secondary storage
CPU quantum size (for each priority i)
Number of printers for priority i
Number of card readers for priority i

Major characteristics of programs in on-line mode (average values)

Program size (code) = 30,000 bytes
Defined data space for execution = 10,000 bytes
CPU time required by a program = 60 milliseconds
CPU time between data file references = 4 milliseconds
CPU time between jump instructions = 50 microseconds
CPU time between data references = 10 microseconds
Percentage of CPU time spent in execution of the sort routine = 12 percent
Percentage of CPU time spent in execution of library subroutines = 30 percent
Probability of execution of the sort routine = 40 percent
Probability of requiring I/O access/subroutine call = 40 percent

System/370 integrated emulation under OS and DOS

by GARY R. ALLRED

International Business Machines Corporation
Kingston, New York

INTRODUCTION

The purpose of this paper is to discuss the design and development of integrated emulators for the IBM System/370. Specifically, emulation of IBM 1400 and 7000 series systems on the IBM System/370 Models 145, 155 and 165, integrated under an Operating System. While the author acknowledges the development and presence of emulation outside of IBM, it is not the intent of this article to conduct a comparative survey of all emulator products. Rather, the discussion will be restricted to the design and development considerations involved in producing the System/370 integrated emulators.

EMULATOR HISTORY

The System/370 integrated emulators are evolutionary products of earlier IBM emulators on System/360. Before discussing the design and functional characteristics of the System/370 emulators, a review of emulation as it existed prior to System/370 is presented in order to form a base of comparison for the new system.

Three methods are employed by the emulators to effect the execution of prior systems programs on the new system and are referred to throughout this article:

1. Interpretation/execution via hardware (referred to as *emulation*).
2. Interpretation/execution via software routine (referred to as *simulation*).
3. A combination of the above (referred to as emulation).

In System/360, emulation was composed of three distinct design types:

1. Hardware: Some emulators, such as the IBM 1401 emulator on System/360 Model 30, were exclusively implemented by hardware. The Model 30 became, in effect, a 1401, with the appropriate registers, addressing, I and E-time execution, etc., handled by the hardware. While performance and operating characteristics were quite good, the system was dedicated to a specific mode of operation, i.e., either emulation or "native" mode. The resultant loading and reloading necessary to attain the desired mode of operation imposed unproductive overhead on the user.
2. Hardware/Software: Other emulators, such as the IBM 7000 series emulators on System/360 Model 65, were comprised of both hardware and software. By adding software, the emulator offered more flexibility in device support and operational characteristics, while at the same time retained the desirable performance attributes of hardware execution. (Pure software implementation would become total simulation, with the obvious degradation of performance.) These emulators required a total dedication to the system of a specific operating mode—emulation or native, with the unproductive overhead of loading and reloading. This overhead was more noticeable when the native mode operation involved an operating system. Additionally, terminal applications were not possible because of the necessity to "shut down" the operating system for emulator loading.
3. Hardware/Software/Operating System: Two programs, Compatibility Operating System (COS)/30 and COS/40 were developed by IBM which integrated the 1401 emulator on System/360 Models 30 and 40 under DOS. At first considered to be interim programs, these programs, because of their wide acceptance and usage, were subsequently upgraded through hardware and software refinements and renamed Compati-

bility System (CS)/30 and CS/40. For the first time, 1401 jobs and System/360 native-mode jobs could be run concurrently in a limited multiprogramming environment. (Limited multiprogramming in the sense that there were certain restrictions on the Foreground/Background allocation of jobs under DOS.) Single job stream input was also possible. Overall system thruput was significantly improved by eliminating the need to reload the system between emulator and System/360 jobs.

In addition to the CS emulators, there were other applications such as Hypervisors and "hook loaders," which, to a lesser degree, provided a single operating environment by eliminating the need to re-IPL between emulator and System/360 jobs. Hypervisors enabled two emulators to run concurrently or, an emulator to run with a System/360 job.

The Hypervisor concept was relatively simple. It consisted of an addendum to the emulator program and a hardware modification on a Model 65 having a compatibility feature. The hardware modification divided the Model 65 into two partitions, each addressable from 0-n. The program addendum, having overlaid the system Program Status Words (PSW) with its own, became the interrupt handler for the entire system. After determining which partition had initiated the event causing the interrupt, control was transferred accordingly. The Hypervisor required dedicated I/O devices for each partition and, because of this, the I/O configurations were usually quite large, and, therefore, prohibitive to the majority of users.

Hook loaders, developed by individual installations, effected a "roll-in/roll-out" of the emulator or System/360 job. The decision to swap operating modes could be interrupt driven or initiated by the operator. The basic attribute of this application was to eliminate the need for IPL when changing operating modes.

DESIGN CONSIDERATIONS AND OBJECTIVES

At the time they were initially released, the System/360 emulators were considered to be short term programs. They were intended to provide the user with the facility to grow from a second generation system to the improved facilities of System/360 with little or no reprogramming. To this end, they served their purpose very well. Their predicted demise however, did not take place as expected. Emulation usage continued at a high rate, with installation resources directed at new applications rather than conversion of existing applications.

Clearly, as system and customer applications became more complex, the need for expanded emulator support became more evident. Early in the planning cycle of System/370, IBM began a design study to determine the most efficient architecture for emulators on System/370. Based on an analysis of existing and projected future operating environments, feedback from user groups, and the experience gained to date with emulation, the following key design points were established as objectives for System/370 emulators:

1. Emulators must be fully integrated with the operating system and run as a problem program.
2. Complete multiprogramming facilities must be available including multiprogramming of emulators.
3. Device independence, with all device allocation performed by the operating system.
4. Data compatibility with the operating system.
5. A single jobstream environment.
6. A common, modular architecture for improved maintenance and portability.
7. An improved hardware feature design with emulator mode restrictions eliminated and all feature operations interruptible.

MODELING

While the COS/CS emulators had proved the basic feasibility of integrating an emulator as a problem program under an operating system, in this case DOS, extending this feasibility to include a large scale, complex system with the full multiprogramming facilities of OS/360 remained to be proven. Therefore, it was decided that a model should be built which would integrate a large scale system into OS/360.

The system selected was the 7094 Emulator on System/360 Model 65. The 7094 and the 7094 Operating System (IBSYS) represented the most complex and sophisticated second generation system available. If this system could be successfully integrated with OS/360, the design and technology could certainly be applied to smaller, less complex systems.

The OS/360 option selected was MFT II. This system, with its fixed partition requirement, could be more easily adapted to the 7094 Emulator design which also included fixed locations and addressing.

This particular feasibility study proved to be an excellent subject for modeling. The goals were well defined, the emulator itself was relatively self contained, and the design alternatives were varied enough to make multiple design evaluations necessary. Modeling was primarily concerned with the assessment of four major areas: input/output techniques, operation under an operating system, hardware design/requirements, and operating system interfaces. There were a number of key recommendations and resolutions achieved in these areas as the result of modeling.

Input/output techniques

- To provide the most efficient means of I/O Simulation, an emulator access method with standard interfaces to the operating system was developed. OS/360 Basic Sequential Access Method (BSAM) was used for tape operations and Queued Sequential Access Method (QSAM) for support of Unit Record devices. Basic Direct Access Method (BDAM) support was later added for those systems that support disk. This access method was subsequently expanded to be usable by any System/370 emulator, regardless of the emulated system. This access method is currently used by the 1400 and 7000 series emulators on System/370.
- To solve the problem of prohibitively long tape records (32K maximum), and some file formats which were unacceptable to OS/360, a tape preprocessor program was developed to format second-generation tapes into a spanned variable length record format. A post-processor was also developed to convert these tapes back to their original format, if desired.
- To enable selective processing for Operating System error recovery procedures, parity switching and density switching modifications were made to the data management facilities of OS/360.

Operation under an operating system

- Whereas the stand alone emulators had used privileged instructions at will, this could not be done if the emulator was to run as a problem program under the operating system. Those routines requiring privileged Op-Codes were either replaced by operating system routines or redesigned to use only standard Op-Codes.
  To achieve a common, portable architecture, emulator routines were standardized as emulator dependent and operating system dependent modules.

Hardware design/requirements

The need to operate in emulator mode should be eliminated. The emulator program should be transparent to the operating system.

- There should be no fixed addresses and the emulator including the target memory, should be relocatable.
- Emulator Op-Codes should be standardized.
- Emulator Op-Codes should be interruptible and capable of retry. (In emulation, it is possible to remain in E-time simulation for an unusually long period of time, relative to normal System/370 E-time. Therefore, the hardware feature, must be fully interruptible if functions requiring the immediate dispatch of asynchronous interrupts are to be supported.)
- Hardware/Software communication should be done via General Purpose Registers and Floating Point Registers rather than through special hardware registers and/or fixed tables. This is required if emulators are to be multiprogrammed.

Operating system interfaces

- Three standard interfaces were defined. These interfaces are emulator and operating system dependent.
  1. An interface was established between the compatibility feature and the emulator modules which performed CPU simulation, I/O simulation and Operator Services.
  2. A second interface was established between the CPU, I/O and Operator Service modules and the emulator access method.
  3. A third interface was established between the emulator access method and the operating system
- By implementing the emulator to these standard defined interfaces, the goal of a common, modular design with the inherent facility of portability was realized.

In summary, the modeling effort successfully demonstrated the feasibility of large scale integrated emulation, while at the same time meeting all of the design and performance objectives. The architecture which evolved from the model was used by the OS/M85/7094 emulator and was released in early 1970. This architecture, with further refinements, is used by all of the System/370 emulators:

Models 145 and 155	Model 165
DOS/1401–1440–1460	OS/7074
DOS/1410–7010	OS/7080
OS/1401–1440–1460	OS/7094
OS/1410–7010	

These systems represent the most advanced emulators ever offered in the IBM product line, combining the powerful new System/370, its high performance I/O devices, the multiprogramming facilities of Operating System (OS)/360 and Disk Operating System (DOS)/360, and an improved technology in emulator design.

SYSTEM REQUIREMENTS AND FEATURES

On the Model 155 there are four emulator combinations available. The 1401/1440/1460 Emulator under both DOS and OS and the 1410/7010 Emulators under DOS and OS. These are four separate programs, each with an individual program number. The compatibility feature on System/370 Model 155 is an integrated feature which provides the facility to emulate the 1401/1440/1460/ and 1410/7010. These emulators can be multiprogrammed in any combination.

On the model 165—7074, 7080 and 7094 emulators are provided. These emulators run under OS/360 and can be multiprogrammed. Each emulator consists of a compatibility feature and a corresponding emulator program that has a unique feature and program number. Only one feature can be installed in the system at one time.

The System/370 emulators have a number of requirements, considerations and support functions in common:

Minimum Requirements

- Compatibility Feature
- A sufficient number of System/370 I/O devices to correspond to the devices on the system being emulated, plus the devices required by the Operating System.
- Sufficient System/370 processor storage for: (1) the version of the operating system being used (MFT, MVT or DOS), (2) emulator functions needed for the system being emulated, and (3) the program being executed.

Additional Features

- Two tape formatting programs are provided: (1) to convert 1400/7000 series tape files to Operating System (spanned variable length) format for more efficient data handling by the emulator, and, (2) to convert output records in spanned variable length format to original 1400/7000 series format.
- A disk formatting program is provided to assist in converting 1400/7010 disk files to the standard Operating System format.

Data File Restrictions:

- 1400/7000 series tape files must be converted if record lengths exceed 32,755 bytes or, if data is in mixed densities.
- All 1400/7010 disk files must be converted.

COMPATIBILITY FEATURES

The Compatibility Features on System/370 Models 155 and 165 are under microprogram control. The feature on the Model 155 is an installed resident feature, whereas on the Model 165 it is loaded into "Writable Control Storage" via the console file.

The compatibility feature is, in effect, a number of special instructions added to the base System/370. These special instructions are used by the emulator program to emulate target machine operations. The selection of operations to be performed by the special instructions is based on an analysis of the target machine operations relative to complexity and frequency of use.

The most significant special instruction (since it is used once for each target machine instruction executed) is called DO INTERPRETIVE LOOP or simply, DIL (Figure 1). The DIL instruction replaces with a single instruction the subroutine that a pure software sub-

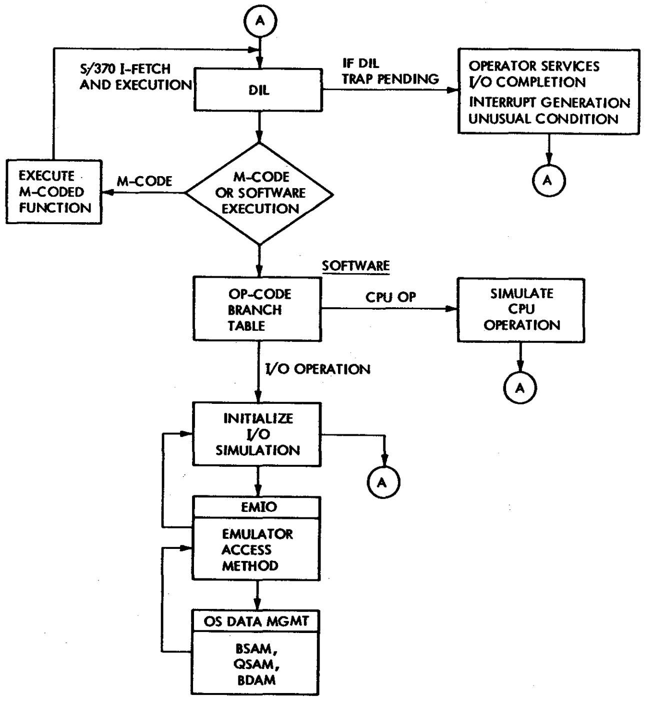


Figure 1—Overview of emulator instruction execution

routine would use to:

1. Access the simulated instruction counter (IC).
2. Convert the IC to a System/370 address in the simulated target machine storage which contains the instruction to be interpreted.
3. Fetch the instruction.
4. Update and restore the simulated IC.
5. Perform any indexing required for the subject instruction.
6. Convert the effective address obtained to the System/370 address in the simulated target machine storage which contains the subject operand.
7. Interpret the instruction Op-Code and branch to the appropriate simulator routine which will simulate the instruction.

INPUT/OUTPUT DEVICE CORRESPONDENCE

Expanded support of I/O devices is provided with the System/370 integrated emulators. The OS Emulators employ the QSAM, BSAM and BDAM facilities of OS/360 Data Management, and offer device independence within the support capabilities of these access methods. The DOS emulators provide device independence only for Unit Record devices.

DISTRIBUTION

DOS

The DOS emulators for 1400/7010 are distributed as components of DOS. Standard DOS system generation procedures are followed in generating an emulator system.

OS

The OS emulators for System/370 Models 155 and 165 are distributed independently of OS/360. Independent distribution was chosen inasmuch as the emulator modules would be superfluous to System/360 users and take up unnecessary space on the distributed system libraries.

SUPPLEMENTAL PROGRAMS

Tape formatting programs

Two tape formatting programs are distributed with the emulator program. The Preprocessor program converts tapes in original 1400/7000 series format to spanned variable-length record format. Any 1400/7000 series tape containing records longer than 32,755 characters must be preprocessed. Preprocessing of other tapes is optional, although greater buffering efficiency can be obtained because the emulator is intended to normally operate with a spanned variable-length format.

The post-processor program converts tape data sets from spanned variable-length format to 1400/7000 series format. The programs support tapes at 200, 556, 800 and 1600 BPI density and handle mixed density tapes. The programs support even, odd and mixed parity tapes.

Disk formatting program

A disk formatting program is provided to assist in converting 1400 disk files to a format acceptable to the emulator program. The disk formatting program runs as a problem program under the operating system. The program creates a data set composed of control information and of blank records whose size and number are determined by the device being emulated.

COMMUNICATING WITH THE EMULATOR PROGRAM

A full range of operator services are provided for operator communication with the emulator program. 1400/7000 series console operations are simulated through commands entered by the operator.

In an integrated, multiprogramming environment, the operating characteristics are expected to initially be more difficult for the operator. However, every effort has been made to ease the transition from stand-alone to integrated operation. Messages from the emulator program are identified by a unique message ID, including a sequentially-incremented message number and the job name of the program being emulated. The user has the option of including multiple console support and directing emulation messages to the second console.

SUMMARY

The System/370 integrated emulators have significantly extended the technology of emulation. They bring to the user an improved, more efficient operating environment for emulator and native mode System/370 jobs, while at the same time providing a nondisruptive growth path for today's System/360 user.

REFERENCE MATERIAL

System/370 Emulators—SRL Publications

Emulating the IBM 1401, 1440, 1460 on the IBM System/370 Models 145 and 155 Using DOS/360- #GC33-2004

Emulating the IBM 1410 and 7010 on the IBM System/370 Models 145 and 155 Using DOS/360- #GC33-2005.

Emulating the IBM 1401, 1440, 1460 on the IBM System/370 Models 145 and 155 Using OS/360- #GC27-6945.

Emulating the IBM 1410 and 7010 on the IBM System/370 Models 145 and 155 Using OS/360- #GC27-6946

Emulating the IBM 7070/7074 on the IBM System/370 Model 165 Using OS/360- #GC27-6948

Emulating the IBM 7080 on the IBM System/370 Model 165 Using OS/360- #GC27-6952

Emulating the IBM 709, 7090, 7094, 7094II on the IBM System/370 Model 165 Using OS/360- #GC27-6951

Hypervisor Documentation

Hypervisor for Running 7074 Emulation as an OS/360 Task- #360D-05.2.005

Double 7074 Emulation on a System/360 Model 65- #360D-05.2.008

Hypervisor RPQ's

Shared Storage RPQ for a System/360 Model 65- #E 880801

Shared Storage RPQ for a System/360 Model 50- #E 56222

A high-level microprogramming language (MPL)

by R. H. ECKHOUSE, JR.

S.U.N.Y. at Buffalo
Amherst, New York

INTRODUCTION

As late as 1967, a prominent researcher reported to his organization[1] that he believed a successful higher-level microprogramming language seemed unlikely. At the same time, other members of the same organization were describing what they termed "A Microprogram Compiler".[2] Meanwhile, other hardware and software designers, equally oblivious of each other, were generating useful and powerful higher-level languages to assist them in their work. As the reader will see, the stage had been set for the development of a higher-level, machine-independent language to be used for the task of writing microprograms.

The research here reported describes a microprogramming language, called MPL, and includes several aspects of the development and use of such a language. The objectives for the language, the advantages and disadvantages, the work which has preceded the development, and the importance and relevance of developing such a language are considered. Finally, we shall consider this current research, showing some preliminary but very promising results.

HIGHER-LEVEL LANGUAGES FOR MICROPROGRAMMING

The area of microprogramming has opened new possibilities for both software and hardware designers because microprogramming has, to a certain extent, blurred the once clear separation of level between the two. In microprogrammable machines we find hardware circuits that incorporate read-only or read/write memory which can determine both the computer's actions and its language. It is therefore beneficial to consider the needs of both the hardware designer and the software designer in the development of a microprogramming language.

Objectives

The hardware designer (the traditional microprogrammer) needs the ability to express the relevant behavior and structural properties of the system. The software designer needs the flexibility of a programming system which allows him to describe the procedures by which a machine can execute a desired function. In combining both of these needs, as microprogramming does, we find that a suitable microprogramming language must be one that is high-level, procedural, descriptive, flexible, and possibly machine-independent.

Advantages

A high-level microprogramming language will free the users from such non-essential considerations as table layouts, register assignments, and trivial bookkeeping details. The language will have the obvious benefits of improved programmer productivity, greater program flexibility, better documentation, and more transferability. By providing the necessary tools for the hardware designer, the software designer, and the machine user, this language can be part of a larger system which is viable for all phases of system design: description, simulation, interpretation, and code generation.

Disadvantages

The seemingly obvious (and traditional) disadvantages of utilizing a higher-level language for microprogramming are loss of efficiency, inflexibility, and high cost. The critics cite the need for a high degree of machine usage, tight code and maximum utilization of every bit in a microinstruction as the major factors for ruling out the use of such a language. "Basically, a compiler would generally be forced to compete with a microprogrammer who can justifiably spend many hours trying to squeeze a cycle out of his code and who may make changes in the data path to do so".[1]

In light of the larger aspects of microprogramming, the above criticisms seem much less tenable. First, the current users of machines which can be microprogrammed are not only their hardware designers. These users do not wish to exercise the microprocessor to its fullest extent if this leads to "tricky" code, or code that is difficult to write, debug, and test. Instead, these users wish to be able to write their own emulation or application software and be able to use higher-level languages with all of their benefits.

The second point is that there exist scant measurement standards for determining efficiency, flexibility, and cost of the presently used methods. Manufacturers, when asked questions concerning hardware utilization, concurrency, and efficiency, tend to state that "the total core size of the microprogram is only X", or "our machine has achieved the desired speed Y", leaving us puzzled and unenlightened.

The real costs and savings inherent in a microprogrammable machine should not be measured in terms of raw speed or core size alone but must be concerned with the unique flexibility that such a machine offers. When we discover bugs in the virtual system, we know that it is clearly less costly to write and implement new microroutines in a microprogrammable machine than to rewire a non-microprogrammable machine. And when we desire to add new features such as virtual instructions or hardware I/O options, it is again less costly to do so to a microprogrammable machine. Thus it would seem that if a higher-level language can aid this process by further reducing the cost of writing microroutines, then clearly such an approach is viable and well worthwhile. The reader is referred to the work of Corbato[3] for additional discussion of the approach.

A suitable language

In developing a suitable language for writing microprograms, the language developer should ask himself if his language would be new, better, more enlighted or useful than some existing, well known language. Instead of adding to the proliferation of languages, it would be well worthwhile to utilize some existing language, with extentions if necessary, as the basis for the development. Fortunately, an appropriate, higher-level language does exist and can be used to not only write microprograms, but also be used to describe, simulate, and design hardware.

A small dialect of PL/I, akin to XPL, represents a suitably modified language amenable to microprogramming. This paper will report on the on-going effort of the design and use of this dialect, the author's higher-level language for writing microprograms called MPL. Another dialect of PL/I described in CASD[4] has already been used to aid in the design of computers (both microprogrammable and not). Thus, the use of a higher-level language has already been demonstrated to be a viable technique for the design, description, and simulation of computer systems, and need not be treated further in this paper.

Importance and relevance

In the process of developing a microprogramming language such as MPL we must be concerned with the relevance of the language. We must find out how effectively the language may be used, and how capable it is in meeting the needs of the user. Thus, performance is a criterion of acceptance and we must be able to demonstrate the ease of producing a meaningful high-level program which can be suitably translated into efficient microcode.

BACKGROUND

Previous work in developing higher-level languages for software and hardware designers is rather extensive. Unfortunately neither side has been concerned with the other, and we find few attempts to reconcile the two. APL is one exception, and its proponents have catagorized it as a universal, systematic language which is satisfactory for all areas of application.[5] Papers have been written to show how APL can be used to describe hardware, to formulate problems, and to design systems. Another exception, previously discussed, is CASD.[4] However, the CASD project has since been disbanded, and no attempt has been made to write the microcode translator discussed in the report of that project.

Systems programming languages

For all of its contributions and contributors, APL does not adequately describe systems programming or microprogramming problems (e.g., timing, asynchronicity, and multiprogramming) without additional explanations in English. In addition, only a subset of APL has been implemented and the whole language remains significant but unimplemented.

Other contributions to higher-level, systems programming languages have included EPL, ESPOL, SYMPL, and IMP. These languages possess block structure, compound statements, and logical connectives which make the job of system design much easier. The MULTICS project[3] and the development of the B5500, with its unconventional "machine language," have demonstrated the successful utilization of higher-level languages to operating systems design.

Hardware design languages

Recent papers by hardware designers seem to indicate a strong trend toward the use of higher-level, machine-independent languages for hardware design. The objectives of these papers appear to be:

(1) To describe digital machines
  (a) Their logic
  (b) Their timing and sequencing
(2) To simulate digital machines
  (a) Verify new designs
  (b) Verify new features
(3) To have machine translatable, formal, hardware description languages
  (a) Supporting (1) and (2) above
  (b) To simplify machine design

The objectives have been met to various degrees as evidenced by the work of Metze and Seshu,[6] Chu,[7, 8] Darringer,[9] Schlaeppi,[10] Schorr,[11] Proctor,[12] Gorman and Anderson,[13] and Zucker.[14] Much of their work seems amenable in its application to microprogramming, and all of it represents the application of an existing higher-level language structure (FORTRAN or ALGOL) to the hardware specification and design problem.

Microprogramming languages

The first evidence of a language structure for writing microprograms appears to be in the work of Husson et al.[2] The authors present their views on the more general concepts for designing a microprogram compiler but they do not have the experience of a working compiler. They discuss a compiler-language which is high-level, procedural, descriptive, and machine-independent. They suggest that such a language will require an intermediary language (some form of an UNCOL) which will allow for the successful generation of a simulator and a machine-dependent interpretation of the microcode.

The authors go on to suggest that there should be compatability between adjacent, architecturally similar processors or classes of machines. Thus, the compiler-language must permit hierarchical descriptions of the particular machine class.

Universal languages

Many of the problems encountered by the hardware and software designers which concern machine-independence are discussed in papers on SLANG[15] and UNCOL.[16, 17] The SLANG system is concerned with the basic question, "Is it possible to describe in a machine-independent language processes which in themselves are machine-dependent?" In the papers addressing the UNCOL concept, we find the discussion on whether or not there exists some intermediate language(s) between any problem-oriented language and any machine language, and whether or not the separation of machine-independent aspects of a procedure oriented language from the machine-dependent aspect is feasible.

THE LANGUAGE AND ITS TRANSLATOR

The choice of a higher-level, machine-independent language for this research required consideration of several aspects in its development. Some of these aspects and the conclusions to which their consideration lead included:

(1) A survey of a representative sample of microprogrammable machines, i.e., how machine-independent or widely applicable is the proposed language?
(2) What is the syntax of the language? What are its syntactic elements and how do they relate to microprogramming?
(3) What is the objective of the language? Is it ease of translation into efficient microcode, or is it ease of describing application problems which can be converted into microcode?
(4) How is translation into microcode performed? If the language is machine-independent, at what stage in its translation is machine-dependence introduced? At what stage do we tailor the code toward the particular microprogrammable machine?
(5) How do we evaluate the code produced? How do we know it is correct or good? Is it "concise"?

What follows are answers to these questions, and an analysis of the effects these answers had in dictating the ultimate results.

Microprogrammable hardware

The objective in surveying current hardware was to attempt to classify the similarities and differences in the various microprogrammable machines. As expected, the architectural differences are not overwhelming, and in many cases are manifest more in terms of the "state of the art" technology, than in differences in type of instruction set, testable conditions, types of addressing, etc. Indeed, all of the machines can be classified as classical, Von Neumann in nature with only minor perturbations.

Syntax

The literature abounds with various languages for writing systems programs (MOL-360, BCPL, PL/360, etc.) and for describing and simulating hardware (LOTIS, CASD, Computer Compiler, etc.). In all cases, the syntax is simple and easily translatable into hardware implementable semantics. Such an approach was taken in specifying the PL/I-like syntax of MPL.

Procedures and declarations

As in PL/I, the basic building block of MPL is the procedure. The concepts of local and global, scope of names, etc., have been preserved and represent the block structure of the language.

Declarations of the various data items (including registers, central memory, and events) give the attributes of the items. By use of the PL/I "DEFINE" syntax, register data items are subdivided into their principal parts (i.e., we may declare a virtual 2n-bit register and then define its true constituent n-bit parts).

Data items

There are basically six types of data items. First, there are the machine registers, both true and virtual. Second, there is central and micro memory. Third, there is both local and auxilary storage which can be similar to the register data type or the central memory data type, depending on its implementation in the actual microprogrammable machine. Fourth, there are "events," unlike events in PL/I, which correspond to testable machine conditions (carry, overflow, etc.). Fifth, there are constants of the type decimal (e.g., 2), binary (e.g., 1011B), and hexadecimal (e.g., OFX). The traditional enclosing quotes around binary and hexadecimal constants may be dropped as long as the constant begins with a digit and ends with a B or X. There are also label constants and symbol constants (or literal constants ala XPL). Finally, there are variables which will take on constant values.

Statements

Assignment statements have been modified in MPL to allow concatenated items to appear on either side of the equal sign. Thus, the concatenation of two registers R1 and R2 becomes R1//R2. This newly defined, double length register can be used logically as if it actually existed such as:

$$R1//R2 = R1//R2 + 2;$$

Additional binary and logical operators have been added or modified in MPL and include:

a .RSH. b	Shift a right b places
a .LSH. b	Shift a left b places
a $\wedge$ b	a and b
a $\vee$ b	a or b
a $\nparallel$ b	a exclusive-or b

Finally, the IF statement is able to test an EVENT previously declared. Thus, a convenient means exists for a transfer on carry, overflow, etc.

Objectives of the language

With microprogrammable machines, two emulation objectives are commonly identified. First, the hardware may be used to emulate a particular system (S/360 on the IBM 2025, 1130 on the Meta IV). Second, the microprogrammable hardware may be used to emulate a particular environment (SNOBOL4, a banking system, etc.). Traditionally, the former objective requires tight microcoding with efficiency of the produced microcode the end goal. The latter objective requires a good run-time environment which can support, through emulated primitives, those features peculiar to the application environment.

The first objective generally requires an efficient translator, with various techniques of optimization, including the use of an intermediate language.[18] The second generally does not require an intermediate language, since in most cases the primary language can be directly translated into the emulated primitives implemented on the host machine.

Translation procedure

In this research the use of an intermediate language called SML has greatly facilitated the translation process from a higher-level machine-independent language into microcode. The basis for this intermediate language can be found in an early paper by Melvin Conway on

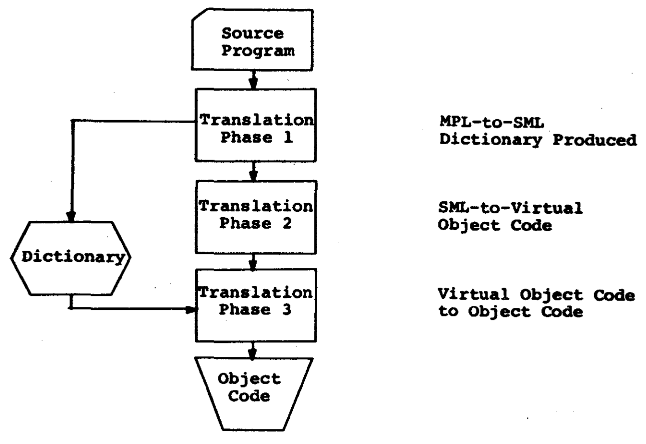


Figure 1—Organization of the translator

the use of an UNCOL.[19] SML-to-microcode translators have been written for the INTERDATA 3, and are capable of producing "compact" code (see Appendix A). In addition, the translation algorithm for converting the MPL code into microcode allows for multiple precision data manipulations, a feature very common to emulator programs. The result is that the process of emulating a 2n-bit word machine on an n-bit microprogrammable machine is easily done at the highest level (MPL level) in a most natural fashion.

The general organization of the translator is shown in Figure 1. Source code is initially translated into SML in phase 1. At the same time, a dictionary is constructed for later use in phase 3. Items entered into the dictionary include real and virtual registers, testable conditions, literals, and other items DECLAREd. Although the SML produced is machine-independent, the dictionary is not in that it relates virtual data items to their real equivalents.

Phase 2 of the translator produces virtual object code from the SML input. The code is virtual in the sense that the operands of machine instructions may be virtual data items (concatenated registers, multiple precision data items, literals, etc.) and need not be of equal widths.

The conversion from virtual object code to object code is resolved in phase 3. Operands of unequal or virtual nature must be converted to true machine instructions. Literals, immediate operands, virtual and concatenated registers must be looked up in the dictionary in order that their virtual representations may be replaced by their object representations.

In general, phase 3 will cause additional lines of object code to be generated. This code results from the conversion of virtual operands into true operands.

Preliminary evaluation and future work

The current translators from SML to microcode are written in SNOBOL4. They are capable of producing microcoded routines from SML which closely resembles the same code supplied by the manufacturer (see Appendix A). The whole process of coding the emulation routines has been made considerably easier by using MPL, and removes much of the busy work required in writing microprograms from the programmer's shoulders.

There are certain drawbacks, however, in using any systems languages such as MPL. In particular, when one allows the use of every facility at the highest level, conflicts may arise (such as that which can occur in any high-level language where assembler code may be generated in-line), and indiscriminate use of those facilities may lead to reasonable but unexpected results. This seems a small price to pay in terms of the original objectives set forth in the section on microprogramming languages.

The original objectives of a high-level, procedural, descriptive, flexible, and possibly machine-independent language for writing microprograms have been met so far. The entire process from the higher-level language to the microcode has been considered, and the feasibility seems clear. As Husson points out in his book,[2] the value of this project is in its use for:

(1) Designing
(2) Debugging
(3) Translating

In each case, the programs must be organized, written, tested, and debugged. At each level, the organization and flexibility provided are enough to justify the existence of MPL.

Future work will concern itself with further refinements to the language, and with its application to a multiple data-path machine such as the IBM 2050. Techniques for both local and global optimization of the code produced will also be considered.

APPENDIX A

Figure 2 is a portion of the INTERDATA 3 emulator written in MPL. The emulated environment is that of a simplified 360, and Figure 2 shows part of the initializing routine (to fetch the PSW) and the instruction fetch and decode routines.

In the outer procedure "INITIAL", we find the ex-

```
INTERDATA3: PROCEDURE OPTIONS(MAIN);

    DECLARE (R0,R1,R2,R3,R4,R5,R6,R7,AR,DFR) BIT (8),

            CM (0:32767) BIT (16),

            MAR BIT (16),
                 MAH BIT (8) DEFINED MAR POSITION (1),
                 MAL BIT (8) DEFINED MAR POSITION (9),

            MDR BIT (16),
                 MDH BIT (8) DEFINED MDR POSITION (1),
                 MDL BIT (8) DEFINED MDR POSITION (9),

            LOCCNT BIT (16),

            (CARRY,SNGL,CATN,TRUE,FALSE) EVENT;

    INITIAL:
         /* FETCH THE LOCATION COUNTER AND PUT IT INTO R0 AND R1 */
         MAR = LOCCNT;
         MDR = CM(MAR);
         R0//R1 = MDR;
         GO TO DISPLY;                /* GO CHECK ON CONSOLE SETTINGS */

PHASE1: PROCEDURE;
         /* INSTRUCTION FETCH, LOC CNTR UPDATE & OP CODE DECODE  */

            MAR = R0//R1;                      /* INSTRUCTION ADDRESS */
            MDR = CM(MAR);
            R0//R1 = R0//R1+2;   /* INCREMENT LOCATION COUNTER */
            R4//R3 = MDR;                             /* GET OP CODE */
            R7 = R3.RSH.3;                   /* RIGHT JUSTIFY R1/M1 */
            AR = (R3.LSH.1)v1;                  /* LEFT SHIFT R2/X2 */
            R2,DFR = R4.RSH.4             /* INTO AR WITH LSB SET */
            IF CARRY THEN GO TO RXFORM;
    RRFORM:  R6 = AR^1;                          /* REG-REG FORMAT */
            R5 = 0;
    DECODE:  IF SNGLvCATN THEN GO TO SUPORT;
    SUPRET:  R3 = R4^0FX;                           /* MASK OP CODE */
            AR = R3+(R3.LSH.1);                  /* MULTIPLY BY 3 */
            DFR = R2;
            IF TRUE THEN GO TO ILLEG;
                    ELSE IF FALSEvCARRY THEN GO TO TROUBL;
            .
            .
            .
         END PHASE1;
         .
         .
         .
END INTERDATA3;
```

```
   INITIAL:                                                      INITIAL:
        /* FETCH THE LOCATION COUNTER AND PUT IT INTO R0 AND R1 */
        MAR = LOCCNT;                                            (*LOCCNT,A+0)
                                                                 X(LOAD)
                                                                 (R+0,MAR);

        MDR = CM(MAR);                                           (CM,A+0)
                                                                 X(READ);

        R0//R1 = MDR;                                            (MDR,A+0)
                                                                 X(LOAD)
                                                                 (R+0,R0//R1);

        GO TO DISPLY;            /* GO CHECK ON CONSOLE SETTINGS */ (*DISPLY,A+0)
                                                                 X(JUMP);

PHASE1: PROCEDURE;                                               PHASE1:
        /* INSTRUCTION FETCH, LOC CNTR UPDATE & OP CODE DECODE  */
            MAR = R0//R1;               /* INSTRUCTION ADDRESS */ (R0//R1,A+0)
                                                                 X(LOAD)
                                                                 (R+0,MAR);

            MDR = CM(MAR);                                       (*CM,A+0)
                                                                 X(READ);

            R0//R1 = R0//R1+2;   /* INCREMENT LOCATION COUNTER */ (R0//R1,A+0)
                                                                 (*2,A+1)
                                                                 X(ADD)
                                                                 (R+0,R0//R1);

            R4//R3 = MDR;                         /* GET OP CODE */ (MDR,A+0)
                                                                 X(LOAD)
                                                                 (R+0,R4//R3);

            R7 = R3.RSH.3;              /* RIGHT JUSTIFY R1/M1 */ (R3,A+0)
                                                                 (*3,A+1)
                                                                 X(RIGHTSHIFT)
                                                                 (R+0,R7);

            AR = (R3.LSH.1)v1;             /* LEFT SHIFT R2/X2 */ (R3,A+0)
                                                                 (*1,A+1)
                                                                 X(LEFTSHIFT)
                                                                 (R+0,A+0)
                                                                 (*1,A+1)
                                                                 X(OR)
                                                                 (R+0,AR);

            R2,DFR = R4.RSH.4        /* INTO AR WITH LSB SET */ (R4,A+0)
                                                                 (*4,A+1)
                                                                 X(RIGHTSHIFT)
                                                                 (R+0,DFR)
                                                                 (R+0,R2);

            IF CARRY THEN GO TO RXFORM;                          (*RXFORM,A+0)
                                                                 X(JUMP-ON-CARRY);

   RRFORM:  R6 = AR^1;                      /* REG-REG FORMAT */ RRFORM:
                                                                 (AR,A+0)
                                                                 (*1,A+1)
                                                                 X(AND)
                                                                 (R+0,R6);
```

Figure 2—An example using the PL/I-like syntax

```
         R5 = 0;                                              (*0,A+0)
                                                              X(LOAD)
                                                              (R+0,R5);

DECODE:  IF SNGLvCATN THEN GO TO SUPORT;                      DECODE:
                                                              (*SUPORT,A+0)
                                                              X(JUMP-ON-SNGL)
                                                              X(JUMP-ON-CATN);

SUPRET:  R3 = R4^0FX;                     /* MASK OP CODE */  SUPRET:
                                                              (R4,A+0)
                                                              (*0FX,A+1)
                                                              X(AND)
                                                              (R+0,R3);

         AR = R3+(R3.LSH.1);            /* MULTIPLY BY 3 */  (R3,A+0)
                                                              (*1,A+1)
                                                              X(LEFTSHIFT)
                                                              (R+0,A+0)
                                                              (R3,A+1)
                                                              X(ADD)
                                                              (R+0,AR);

         DFR = R2;                                            (R2,A+0)
                                                              X(LOAD)
                                                              (R+0,DFR);

         IF TRUE THEN GO TO ILLEG;                            (*ILLEG,A+0)
                ELSE IF FALSEvCARRY THEN GO TO TROUBL;        X(JUMP-ON-TRUE)
                                                              (*TROUBL,A+0)
                                                              X(JUMP-ON-FALSE)
                                                              X(JUMP-ON-CARRY);
```

Figure 3—PL/I-like code translated into SML

```
INITIAL    L    MAR,*LOCCNT
           C    MR
           L    R0//R1,MDR
           L    RAR,*DISPLY
PHASE1     L    MAR,R0//R1
           C    MR
           L    AR,*2
           A    R0//R1,R0//R1
           L    R4//R3,MDR
           L    AR,R3,SR+NC
           L    AR,AR,SR+NC
           L    R7,AR,SR+NC
           L    AR,R3,SL+NC
           O    AR,*1
           L    AR,R4,SR+NC
           L    AR,AR,SR+NC
           L    AR,AR,SR+NC
           L    DFR,AR,SR+NC
           L    R2,DFR
           B    C,RXFORM
RRFORM     N    R6,*1
           L    R5,*0
DECODE     T    SNGL
           B    G,SUPORT
           T    CATN
           B    G,SUPORT
SUPRET     L    AR,*0FX
           N    R3,R4
           L    AR,R3,SL+NC
           A    AR,R3
           L    DFR,R2
           B    G,ILLEG
           B    L,TROUBL
           B    C,TROUBL
```

Figure 4—Second phase of the translation

```
INITIAL   L    MAL,=L(LOCCNT)
          L    MAH,=H(LOCCNT)
          C    MR
          L    R1,MDL
          L    R0,MDH
          L    RAL,=L(DISPLY)
          L    RAH,=H(DISPLY)
PHASE1    L    MAL,R1
          L    MAH,R0
          C    MR
          L    AR,X'02'
          A    R1,R1,CO
          A    R0,R0,CI
          L    R3,MDL
          L    R4,MDH
          L    AR,R3,SR+NC
          L    AR,AR,SR+NC
          L    R7,AR,SR+NC
          L    AR,R3,SL+NC
          O    AR,X'01'
          L    AR,R4,SR+NC
          L    AR,AR,SR+NC
          L    AR,AR,SR+NC
          L    DFR,AR,SR+NC
          L    R2,DFR
          B    C,RXFORM
RRFORM    N    R6,X'01'
          L    R5,X'00'
DECODE    T    SNGL
          B    G,SUPORT
          T    CATN
          B    G,SUPORT
SUPRET    L    AR,X'0F'
          N    R3,R4
          L    AR,R3,SL+NC
          A    AR,R3
          L    DFR,R2
          B    G,ILLEG
          B    L,TROUBL
          B    C,TROUBL
```

Figure 5—Third phase of the translation

pected sequence:

(1) Put address into memory address register
(2) Read central memory
(3) Copy data out of memory data register

common to emulator programs. The concatenation of R0 and R1 occurs because central memory is actually 16-bits wide, and reads and writes require double length registers for both addressing and data handling.

The inner procedure "PHASE 1" represents the instruction fetch, location counter update, and op-code format recognizer routines. In it can be found two features not normally found in PL/I. First, the binary operators .RSH. and .LSH. have been added to represent right-shift and left-shift respectively. Additional operators such as exclusive-or have also been included in the syntax since they occur frequently in the instruction sets of microprogrammable machines.

Second, the occurrence of the "events" CARRY, SNGL, CATN, TRUE, etc., in the IF statements are taken to imply that special conditions within the machine can be tested directly. The actual relationship of the event to the physical hardware is specified in the DECLARE statement.

Figure 3 shows the same code as Figure 2, but the translation into SML can be found interspersed on the right-hand side of the figure. Operations are denoted by an "X" followed by parentheses enclosing the name of the operation. Arguments needed for operations must first be loaded into argument or A-registers. Results of operations are placed in result or R-registers. Temporary or T-registers are available for intermediate results. Finally, literals are indicated by preceding their names (values) by an asterisk.

Figure 4 represents the output of phase two of the translator. This is the traditional and more difficult code emission phase of the translation process. The results are not true INTERDATA code, however, and must go through another phase to relate the actual facilities and data-path widths to the virtual facilities and data paths which the programmer assumes.

Figure 5 is the output of the third phase of the translator. The code produced here is actual INTERDATA code in assembler format. The output has required a dictionary to relate the virtual and physical registers, data-paths, etc., to each other. Construction of such a dictionary is accomplished in the MPL to SML phase of the translation, and is facilitated by the declarations in the MPL code.

APPENDIX B

The INTERDATA 3 is a very fast, simple and uncomplicated machine. Control instructions reside in a Read-Only-Memory (ROM) that is 16-bits wide. Thus, the microinstructions of the machine are 16-bits long. However, the data paths are only 8-bits wide.

Microinstructions for the INTERDATA are somewhat similar to the instructions for a two address (register-to-register) machine with an accumulator (AR). The instruction types include:

L	Load	X	Exclusive Or
A	Add	B	Branch On Condition

S	Subtract	T	Test
N	And	C	Command
O	Or	D	Do

The four formats for the ten instruction of the machine are:

op	destination	source	modifiers

for op codes A,S,N,O,X,L

op	destination	data

for immediate instruction forms as above

op	test or command

for test or command instructions

op	C V G L	address

for branch instructions where:

C = carry
V = overflow
G = greater than zero
L = less than zero

Thus an add instruction might look like:

A MAH,R1

where the contents of the source register R1 are added to the contents of the accumulator AR and the result is stored in the destination register MAH.

Modifiers for the various instruction types include:

NA	AR is not added to the source register.
SR	Shift the contents of the source register right one bit and then perform the operation.
SL	Shift left as above.
CI	If the carry flip-flop is on, add a one to this instruction.
CO	Set the carry flip-flop if a carry is generated out of the most significant bit.
NC	No carry.

The assembler allows literals to be specified as hexadecimal constants and labels. Since labels may be 16-bit values, their high and low parts are specified by prefixing the literal by an H or L respectively.

REFERENCES

AFIPS-171, (18), p. 402

1 S G TUCKER
Microprogram control
IBM Systems Journal Volume 6 pp 222-241 1967

2 S S HUSSON
Microprogramming: Principles and practices
Prentice-Hall Engelwood Cliffs New Jersey pp 125-143 1970

3 F J CORBATO
PL/I as a tool for systems programming
MIT Project MAC Memorandum M-378 1968

4 E D CROCKETT et al
Computer-aided system design
AFIPS Conference Proceedings Fall Joint Computer Conference 1970

5 K E IVERSON
Programming notation in systems design
IBM Systems Journal Volume 2 pp 117-128 1963

6 G METZE S SESHU
A proposal for a computer compiler
AFIPS Conference Proceedings Spring Joint Computer Conference pp 253-263 1966

7 X CHU
An ALGOL-like computer design language
Communications of the ACM Volume 8 pp 607-615 1965

8 Y CHU
A higher-order language for describing microprogrammed computers
University of Maryland Computer Science Center Technical Report 68-78 College Park Maryland 1968

9 J A DARRINGER
The description, simulation, and automatic implementation of digiatal computer processors
Ph D Thesis Carnegie-Mellon University Pittsburgh Pennsylvania 1969

10 H P SCHLAEPPI
A formal language for describing machine logic, timing, and sequencing (LOTIS)
IEEE Transactions on Electronic Computers Volume EC-13 pp 439-448 1964

11 H SCHORR
Computer-aided digital system design and analysis using a register transfer language
IEEE Transactions on Electronic Computers Volume EC-13 pp 730-737 1964

12 R M PROCTOR
A logic design translator experiment demonstrating relationships of languages to systems and logic design
IEEE Transactions on Electronic Computers Volume EC-13 pp 422-430 1964

13 D F GORMAN J P ANDERSON
A logic design translator
AFIPS Conference Proceedings Fall Joint Computer Conference pp 251-261 1962

14 M S ZUCKER
LOCS: An EDP machine logic and control simulator

IEEE Transactions on Electronic Computers Volume EC-14 pp 403-416 1965

15 R A SIBLEY
The SLANG system
Communications of the ACM Volume 4 pp 75-84 1961

16 P R BAGLEY
Principles and problems of a universal computer-oriented language
Computer Journal Volume 4 pp 305-312 1962

17 T B STEEL
A first version of UNCOL
Proceedings of the Western Joint Computer Conference pp 371-378 1961

18 D J FRAILEY
Expression optimation using unary complement operators
SIGPLAN Notices Volume 5 pp 67-85 1970

19 M E CONWAY
Proposal for an UNCOL
Communications of the ACM Volume 1 pp 5-8 1958

A firmware APL time-sharing system

by RODNAY ZAKS,* DAVID STEINGART,* and JEFFREY MOORE**

University of California
Berkely, California

INTRODUCTION

Incremental advances in technological design often result in qualitative advances in utilization of technology. The recent introduction of low-cost, microprogrammed computers makes it feasible to dedicate highly sophisticated and powerful computation systems where previously the needed performance could not be economically justified. Historically, the contribution made by the computing sciences to the behavioral sciences has been limited largely to statistical analysis precisely because sufficiently sophisticated computing equipment was available only outside the experimental situation. Inexpensive time-sharing systems have recently made it possible to integrate the computer in a novel way as a tool for conducting experiments to measure human behavior in laboratory situations. A detailed presentation of computerized control of social science experimentation is presented later. However, many aspects of the system are of general interest because they exploit the possibilities of a newly available computer generation.

Iverson's APL language has been found to be very effective in complex decision-making simulations, and the source language for the interpreter to be described is a home-grown dialect of APL. It is in the nature of interpreters that relatively complex operations are performed by single operators, thus making the ratio of primitive executions to operand fetches higher than in any other mode of computation. This is especially true in APL, where most bookkeeping is internal to particular operators, and a single APL operator may replace a FOR block in ALGOL, for example. This high ratio places a premium on the ability of microprogrammed processors to execute highly complex instruction sequences drawing instructions from a very fast control memory instead of from much slower core memory.

In the new generation of microprogrammable computers the microinstructions are powerful enough and the control memories large and fast enough to permit an on-line interpreter and monitor to be implemented in microcode. If a sufficient number of fast hardware registers is available, core memory serves only for storage of source strings and operands. The speed advantages of such a mode of execution are obvious.

* Department of Computer Science

** School of Business Administration

SYSTEM ARCHITECTURE***

META-APL is a multi-processor system for APL time-sharing. One processor ("the language processor") is microprogrammed to interpret APL statements and provide some monitor functions, while a second one (the "Interface processor") handles all input-output, scheduling and provides preprocessing capabilities: formatting, file editing, conversion from external APL to internal APL. Editing capabilities are also provided offline by the display stations. In the language processor's control memory reside the APL interpreter and the executive. In the Interface processor's reside the monitor and the translator.

An APL statement is thus typed and edited at a display station, then shipped to the Interface processor which translates and normalizes this external APL string to "internal APL," a string of tokens whose left part is a descriptor and right part an alphanumeric value or i.d. number corresponding to an entry in the user tables (see appendix A). External APL may be reconstructed directly from internal APL and internal tables, so that the external string is discarded and only its internal form is stored. This internal APL string is shipped to the language processor's memory: the APL processor will now execute this string at the user's request.

*** The concepts presented here are being implemented under the auspices of the Center for Research in Management Science. Research on the final configuration is continuing at the Center and the views expressed in this paper represent the ideas of the authors.

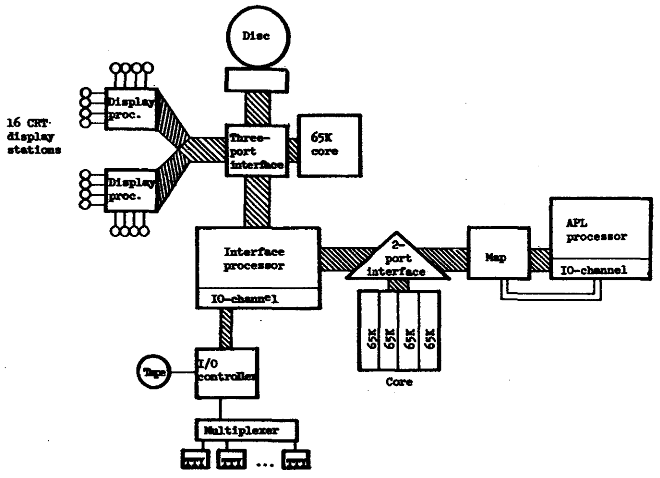


Figure 1—The Meta-APL time-sharing system (projected)

The variable's i.d. numbers ("OC#") are assigned by the system on a first-come-first-served basis and are used for all further internal references to the variable. This OC# is the index which will be used from now on to reference the variable within the OAT (Operand Address Table) of the language processor. The set of internal APL strings constitutes the "program strings."

Microprogramming encourages complex interpretation because the time spent interpreting a given bit or string of bits is negligible. We have taken advantage of this ability to allow short descriptors to replace "dead-data" wherever possible so as to minimize the inert-data flow and maximize the active-data flow. All external program symbols are translated to tokens internally—however, as we have previously mentioned, the grammar and semantics of the internal notation are isomorphic to the external symbolic.

APL PROCESSOR: HARDWARE

The laboratory requirements called for a very fast APL processor capable of executing at least sixteen independent or intercommunicating experimental programs, each program responding in real time to textual and analog input from the subject of the experiment.

Once the possibilities of a microprogrammed interpreter became apparent, the search for a machine centered on those with extensive microprogramming facilities. Of these the Digital Scientific Meta-4 was chosen by the Center for Research in Management Science for its fast instruction cycle, extensive register complement, and capable instruction set.

The processor fetches 32-bit instruction from a read-only memory on an average of every 90 nsec. Instructions fetch operands from thirty-one 16-bit-registers through two buses and deposit results through a third into any register. Most instructions may thus address three registers independently—there are no accumulators as such. Up to 65K of 750 nsec cycle core may be accessed through two of the registers on the buses, IO through another pair, and sixty-four words of 40 nsec scratch pad memory through yet another pair. These registers are addressed as any others and the external communications are initiated by appropriate bits present in all microinstructions.

Triple addressing and a well-rationalized register structure promote compact coding. The entire APL interpreter and executive reside in under 2,000 words of read-only memory.

Although special multiply and divide step microinstructions are implemented in the hardware of the Meta-4, the arithmetic capability of the processor is not on a par with the parsing, stack management, and other nonarithmetic capabilities of the interpreter. Adding a pair of 32-bit floating operands takes about 5 μsec, a very respectable figure for a processor of this size and more than adequate for the laboratory environment. A floating multiply or divide takes 20-25 μsec.

On the other hand, a pass through the decision tree of the parser takes 1-2 μsec, and as will be seen from the descriptor codes this tree is fairly deep. This speed is a consequence of the facility to test any bit or byte in any register and execute a branch in 120 nsec, or mask a register in less than 90 nsec.

APL PROCESSOR: MEMORY

The experimental situation demands that response time of the computer system to external communication be imperceptible. We were forced by this consideration to make all active programs resident in core, and

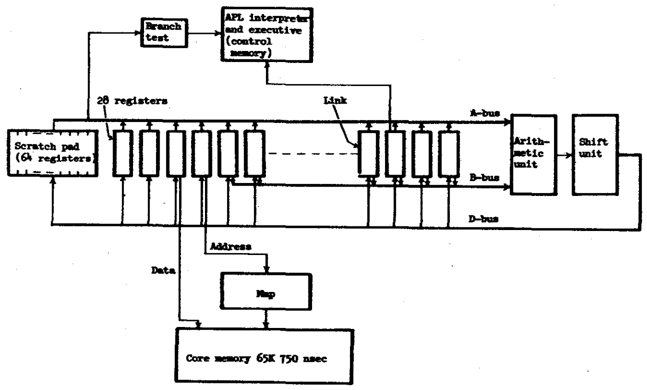


Figure 2—The APL processor

in order to maximize the utilization of the available address space of 65K, several designs evolved.

1. Through a hardware map, the virtual address space of 65K is mapped into 256K core.
2. Since many of the APL operators are implemented in core, and since the experimental situation normally requires many identical environments (programs with respect to the computer), all program strings are accessible concurrently to all processes or users.
3. Through dynamic mapping of the available physical memory space, individual processes may be allocated pages as needed, and pages are released to free space when no longer occupied by a process. Optimal use is made of the waxing and waning of individual processes.

The entire virtual memory space is divided into three contiguous areas: system tables; system and user program strings, processes work space. Within the processes work space, memory is allocated to the stack and management table (MT) for each process. The stack and MT start in separate pages, the stack from the bottom and the MT from the top, and these two are mapped as the bottom and top pages of the virtual work space, regardless of their physical location. As the process grows during execution, pages are allocated to it from free space within the process work space and are mapped as contiguous memory to the existing stack and MT.

The specifics of the memory and map design were constrained primarily by available hardware. The computer used has a 16-bit address field—65K is the maximum direct address space but not adequate for 40 plus processes. Mapping by hardware 256K into 65K eliminates the need for carrying two-word addresses inside the computer. Pages are 512 words long, 128 pages in the 65K virtual space, 512 pages in the real space, keeping fragmentation to a minimum.

The map is a hardware unit built integrally with the memory interface. The core cycle is 750 nsec, the map adds 80-100 nsec to this time.

The map incorporates a 128-word, 12-bit, 40 nsec random access memory which is loaded every time a user is activated. The data comprising the user page map are obtained from a linear scan of the general system memory map.

Each word in the map contains three fields.

- In the n-th word in the hardware map, the right-hand seven bits contain the physical address of the page whose virtual address is n.

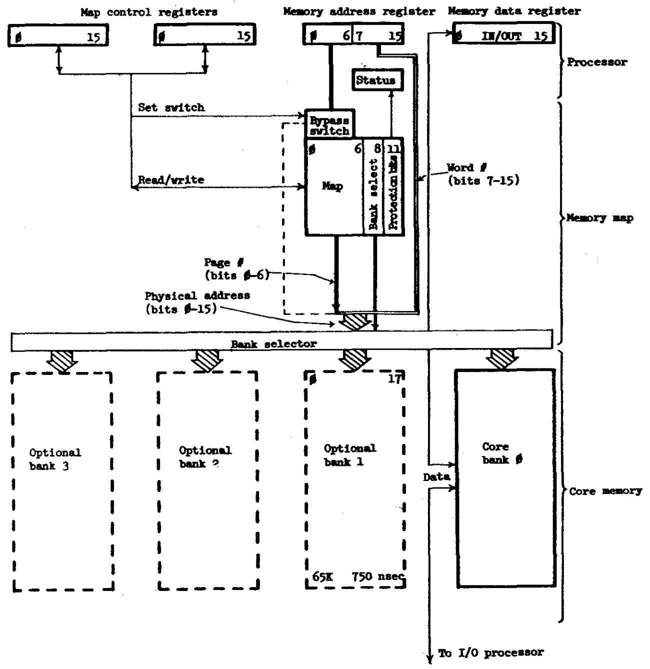


Figure 3—The hardware map

- The two bits adjacent to this field (bits 7, 8) map the 65K space into 256K (i.e., bank select).
- The three remaining have control functions and are returned in about 100 nsec to the status register associated with the memory address register. These bits are thus interpreted by the microprogram and any actions necessary are taken before the memory returns data 350 nsec later, 450 nsec after the initiation of the read (or write).
- The first bit, when set, causes an automatic write abort, thereby providing read-only protection of a given page.
- The second bit, when set, indicates a page fault. When detected in the status register, a special routine is executed which allocates a new page to the user.
- The third bit indicates that a page is under the control of the interface processor and prevents the APL processor from modifying or reading that page.

In general, the program string area is protected by the read-only bit; it may be modified only by the interface processor. All free storage in the processes work space, and all virtual pages not allocated to the process activated at a given time are protected by the page fault bit. Thus, when a process references outside its unprotected area, the request is interpreted as a request for additional storage. When the interface processor is modifying either program strings or input-output buffers, those areas are protected by the read-write abort bit.

The map may be bypassed by setting the appropriate bits in the map access register. This is to permit loading of the map to proceed simultaneously with core fetches, while the map's memory settles down, and to avoid the bootstrapping necessary if the map always intervened in the addressing of core.

The system memory map, stored in the top section of the user's virtual storage establishes the system-wide mapping from physical to virtual pages. Each of the 128 entries, one per physical block contains the owner's ID number (or zero) and the corresponding virtual location within its storage. Free pages are chained with terminal pointers in CPU registers. The overhead incurred in a page fault or page release is thus minimum (3 μsec).

APL PROCESSOR: INTERPRETER SOURCE

The APL interpreter accepts strings of tokens resident in the program strings area of core. The translation from symbolic source to token source is performed externally to the APL processor by an interface processor. The translation process is a one-pass assembly with fix-ups at the end of the pass for forward-defined symbols. The translation preserves the isomorphism between internal and external source and is *naturally* a bidirectional procedure—external source may be regenerated from internal source and a symbol table.

Meta-APL closely resembles standard APL with some restrictions and extensions. The only major restriction is that arrays may have at most two dimensions—a concession toward terseness of the indexing routines. There are two significant extensions.

Functions

Functions may have parameters, which are called by name, in addition to arguments. This is to facilitate the development of a "procedure library" akin to that available in ALGOL or FORTRAN. Parameters eliminate the naming problem inherent in shared code.

The BNF of a Meta-APL function call:

```
⟨FUNCTION CALL⟩:= {⟨ARGUMENT⟩}
                  ⟨FUNCTION NAME⟩
                  {(⟨PARAMETER LIST⟩)}
                  {⟨ARGUMENT⟩}

⟨PARAMETER LIST⟩:= ⟨VARIABLE NAME⟩ |
                   ⟨PARAMETER LIST⟩,
                   ⟨VARIABLE NAME⟩
```

The variables specified as parameters may either be local to the calling function or global. The mechanics of the function call will be described later, as this is one of the aspects of this implementation which is particularly smooth.

Processes

The other significant extension in Meta-APL is the facility of one program to create and communicate with concurrently executing daughter programs, all of which are called processes. Briefly, a process is each executing sequence represented by a stack and management table in the "processes work space." Any process can create a subprocess and communicate with it through a parameter list, although only from global level to global level. The latter restriction is necessary because processes are asynchronous and the only level of a program guaranteed to exist and be accessible at any time is the global level (of both mother and daughter processes).

The activation of a new program,

$NUprog{P1, P2, P3, ..., Pn} PROGRAM NAME

establishes a communication mechanism, the "umbilical cord" between calling program A and called program AA. AA constitutes a new process and will run in parallel with all other processes of the system. The cord however, establishes a special relationship between AA and A:

—the cord may be severed by either A or AA, causing the death of the tree of processes (if any) whose root is AA.

—the parameter list of the $NUprog command establishes the communication channel for transmitting values between A and AA. All these parameters may thus be referenced by either process A or process AA and will cause the appropriate changes in the other process. To prevent critical races, two commands have been introduced.

$WA (WAITFOR) which dismisses the program until some condition holds true.

$CH (CHECK) which returns 1 if the variable has already been assigned a value by the program, ϕ otherwise. It expects a logical argument. Thus $WA ($CH V1 ∨ $CH V2) will hang program A until either V1 or V2 have been evaluated by program AA on the other side of the umbilical cord. It will then resume processing.

Among the parameters that may be passed are IO device descriptors. Hence, a mother process can temporarily assign to daughters any IO device assigned to her. This is to facilitate use of simple reentrant IO communications routines to control multi-terminal interactive experiments under the control of one mother process. The mother may identify daughters executing the same program string by assigning them distinct variables as parameters.

The usual characteristics of well ordering apply to process tree structures.

The BNF of Meta-APL is included as an appendix.

THE DESCRIPTOR MECHANISM

The formats of the internal tokens are as follows: numerical *scalar* quantities are all represented in floating point and fixed only for internal use. The left-most one bit identifies the two words of a floating operand: the 1-bit descriptor allows maximum data transit.

Operand calls

Variables are identified by descriptors called *operand calls* (after Burroughs). The i.d. field of the OC locates an entry in the Operand Address Table (OAT) which gives the current relative address of the variable in the stack, or specifies that a given variable is a parameter or undefined.

Another bit specifies the domain of the variable, local or global. Unlike ALGOL, there is one global block in Meta-APL. The possible addressing is indicated graphically.

When a process is created or a function is called, a block of storage is allocated to the Management Table to store the stack addresses of all the variables of that block—the block known as the Operand Address Table (OAT). The i.d. field of the OC is an index to the OAT. When an OC is encountered as the argument of an operator, the address of the variable is obtained in or through the OAT. If the variable is local to the current block and defined, the current stack address is found in the appropriate location of the local OAT. If the variable is global, as specified by the domain bit, the global level OAT is accessed. In either case, if the variable is undefined, a zero will be found in the OAT entry for that variable, and an error message will result. If the variable is a parameter, an operand call to either the calling or global level will be found in the OAT. In the case of a function, this OC points to either the calling block OAT or the global OAT and the address/zero/parameter OC will be found there. If the OC was found in the global OAT, it is a parameter from the mother process as described above.

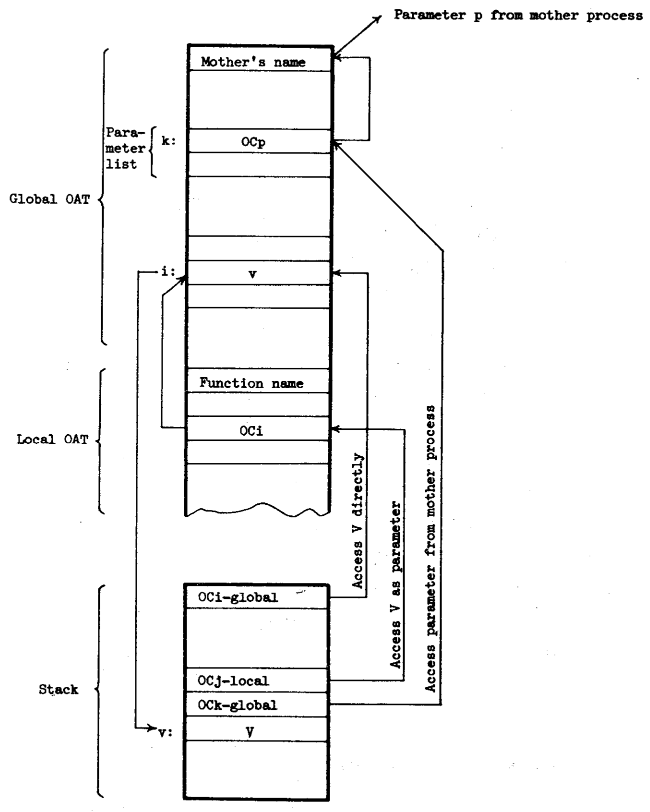


Figure 4—Stack addressing mechanism

Obviously, parameters may be linked through every level of process and function.

Operators

Operators are represented by such word descriptors containing tag bits, identification bits, and some redundant function bits specifying particular operations to the interpreter (marking the stack, initiating execution, terminating execution).

During parsing, operators are placed on an Operator Push Down List created for every block immediately below the OAT for that block. During the execution phase, operators are popped off the OPDL and decoded first for executive action (special bits) and number of arguments. The addresses of the actual operands are calculated as explained under *Variables* and those addresses passed to the operator front end. This routine analyzes the operands for conformability, moves them in some cases, and calls the operator routine to calculate results, either once for scalars, or many times as it indexes through vector operands.

The operator front end represents most of the complexity of the execution phase since the variety of APL operands is so great.

Function call

The mechanism of *function call* uses the OPDL. If a function descriptor is encountered during the parse, it is pushed onto the OPDL and three zeroed words are left after it for storage of the dynamic history pointers. The specifications of the function are looked up in the function table and one additional zeroed word is left for each variable which appears in the function header before the function name, i.e., A←B FOO . . . would result in two spaces zeroed one for A, one for B.

Then, as the parse continues, if a left brace is encountered (as in A←B FOO{P_1, . . . , P_n}C), parameter OCs P_1 through P_n are pushed onto the OPDL until the right brace is encountered. The number of parameters (n) is entered, the function descriptor duplicated, and parsing proceeds on its merry way.

During execution the last entered function descriptor

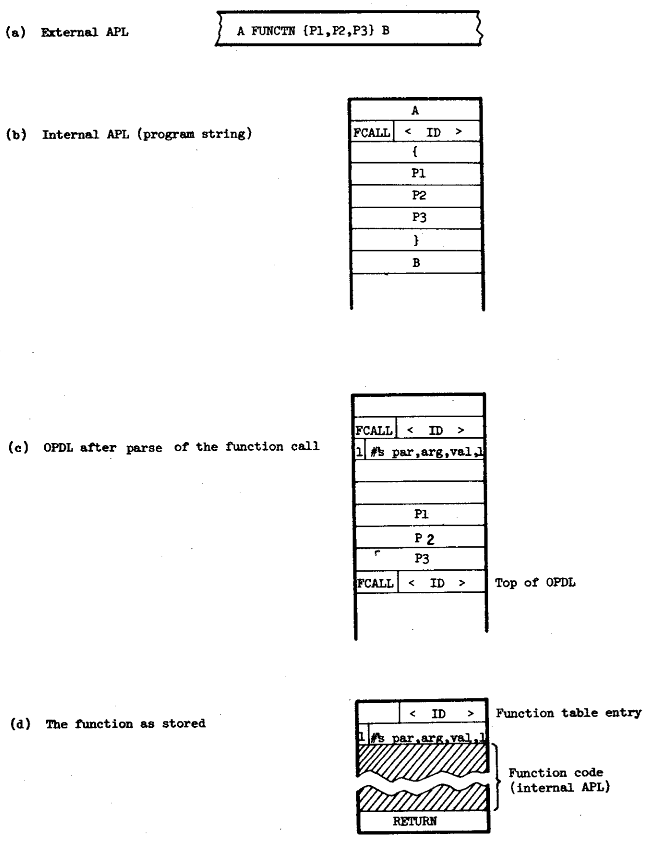


Figure 5—Function call

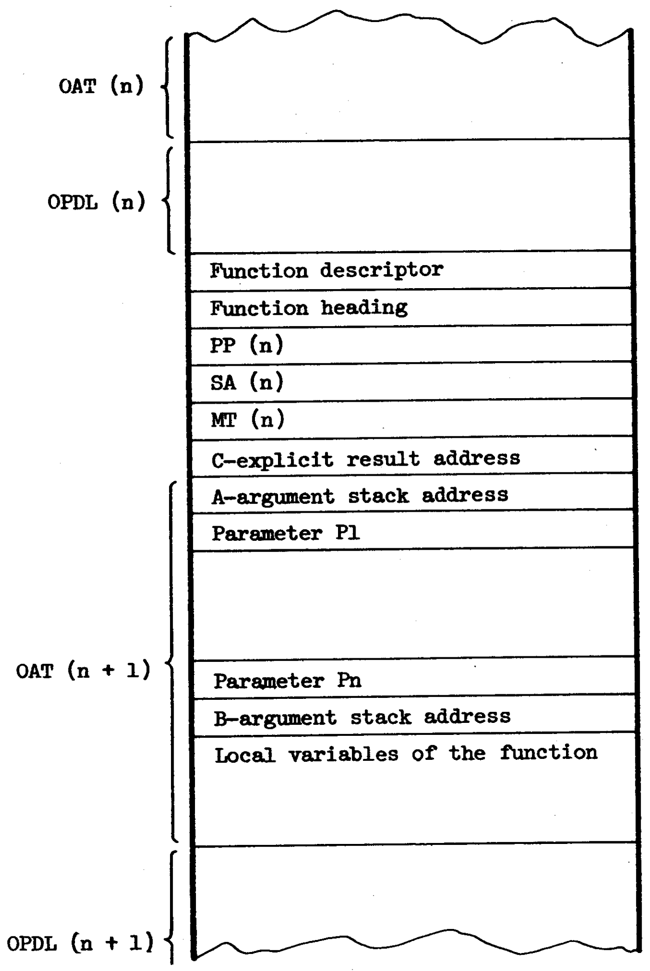


Figure 6—Management table after activation of C←A F00 (P1, P2, . . . , Pn) B

is popped. This initiates the function call. First, the number n above is compared with the number of parameters specified in the function header. Then the address for arguments B and C are entered (these are the current top two elements of the stack). The current stack pointer, MT pointer, and program string pointer are saved in the appropriate locations and x words after the C argument are zeroed to accommodate the x new variables to be defined in the new function; (x was obtained from the function string header).

At this point, control is passed to the function program string with the new OAT already formatted.

The purpose of the preceding description is to indicate the kind of manipulation which is cheap in time and instructions in a microprogrammed interpreter. The function call routine takes under fifty instructions and takes about 10 μsec to execute (plus .9 x μsec to zero x locations).

Other descriptor types:

SYMBOL	#	BINARY	OCTAL	DESCRIPTION	
		ø 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15			
A	ø	ø X P O N E N T < . M . . A . . N . . T . . 1 . . S . . S . A >	ø - - - - - - - - - - -	Arithmetic	Exponent=<1-7> 2's complement Mantissa=<8-31> →first bit is sign(ø=+,1=-)
OC	1-4	1 1 ø ø D T < V A R . . I D # >	1 4 - - - -	Operand call	Domain: 1=global, ø local Trace: 1=on, ø=off Variable I.D. #=<6-15>
V	5-8	1 ø ø ø L Y . . L E N G T H > < X . . L . E . N . G . T . H > < N U M B E R O F . W O R D S >	1 ø - - - -	Vector	Y length up to 11 bits X*Y must be < 16 bits followed by X elements and Y elements L=ø=double words 1=half-words
PH	9	1 ø ø 1 - - - - - - - - - - - -	1 1 - - - -	Phantom	
OP	1ø-41	1 ø 1 ø T R M D E < O P E R A >	1 2 - - - -	Operator	TR=trace M=mark stack]← D=dyadic=1, monadic=ø E=execution delimiter:])⍎ Opera is the operator code(1 bit tells scalar operator)
SEG	42-45	1 ø 1 1 D D - - - - - - - - - -	1 3 - - - -	Segment-operator goes to OPDL	DD=øø=function call: bits 6-15=I.D.# ø1=empty marker (NOOP for parser) 1ø=program call 11=beginning of line + line number
IO	46-49	1 1 1 1 F F - - - - - - - - - -	1 7 - - - -	IO descriptor goes to stack	FF=øø=output program# ø1=input 1ø=unused 11=unused
	5ø	1 1 ø 1 - - - - - - - - - - - -	1 5 - - - -	Unused	
RET	51	1 1 1 ø - - - - - - - - - - - -	1 6 - - - -	End of function definition=RETURN	

Figure 7—The descriptors

A TIME-SHARING SYSTEM FOR BEHAVIORAL SCIENCE EXPERIMENTATION

Historically, the contribution made by the computing sciences in the behavioral-science area has been limited almost exclusively to the utilization of computational resources for statistical analysis of social-science data and simulation studies. The advantages offered by the computer technology in other areas have until very recently been lost to the behavioral sciences. The advent of reliable and economical time-sharing systems has opened new vistas to the research horizons of a social-science experimenter. The use of time-sharing systems in programmed learning for teaching and other educational purposes has been well documented. The objective of this paper is to outline a science whereby the process-control technology combined with time-sharing can be used in a novel way as a tool for conducting experiments to measure human behavior in laboratory situations. Traditionally, attempts to monitor human behavior in decision-making situations have had less than desirable results, due primarily to the extreme difficulty in maintaining control over the course of the experiment. That is, subjects of a behavioral-science experiment often do not behave in a manner which is conducive to exercise of experimental control so that certain variables can be measured in a controlled environment. To meet the challenge of properly controlling experiments in the social sciences, a laboratory for that purpose was created by a grant from the National Science Foundation in the early 1960s.* The intent was to utilize computerized time-sharing with special-purpose hardware, combined with a suitable cubicle arrangement so that subjects participating in, for example, economic-gaming situations could have their decisions monitored and recorded by the time-shared computer. The idea was to have the experimenter model his economic game by writing his mathematical model as a computer program. At run-time, the resulting program serves as the executive controlling subject input from the time-sharing terminal. In this fashion, the computer serves two functions: to provide the medium whereby the experimenter may mathematically express his experimental design and to serve as the data-collection and process-control device during a time-shared experiment in which subjects at the terminals are communicating with the computer.

The requirements placed upon a time-sharing system

* A C HOGGATT J ESHERICK J T WHEELER "A Laboratory to Facilitate Computer Controlled Behavioral Experiments," *Administrative Science Quarterly*, Vol. 14, No. 2, June 1969.

when it is utilized for computer-controlled experiments differ markedly from those placed upon a conventional time-sharing system. The actual implementation of the model itself requires a general-purpose computational capability combined with the usual string-handling capabilities found on any general-purpose time-sharing system; and hence, these are a minimal requirement of any experimental-control computer. The features which most notably distinguish a time-shared computer system when used for experiments are as follows: (1) Response of the processor to input from the remote terminals must be virtually instantaneous, that is, in experimental situations the usual delay of one or more seconds by the time-shared processor after user input is prohibitively long. In some measurement situations, in order not to introduce an additional and uncontrolled variation in his behavior, such as might be caused by even minor frustration with the responsiveness of the time-sharing system, feedback of a response to a subject's input on a time-shared terminal must be less than approximately five hundred milliseconds. In other less rigorous experimental situations in which rapid feedback is important, the relatively lengthy response time of most time-sharing systems has also introduced significant variation in subject behavior. (2) The measurement of human behavior is a costly and time-consuming process, and hence the successful completion of an economic-gaming experiment requires the utmost in reliability of the time-sharing system. Even minor system fall-downs are usually intolerable to the experimental design; for they, at the very least, introduce possible lost data, i.e., lost observations on subject behavior or time delays in the operation of the experiment. A system crash normally causes the experimenter to terminate the experiment and dismiss the subjects and can even force cancellation or modification of an entire sequence of experiments if the system fall-down occurred at a particularly crucial point in the experimental design. For these reasons, the existence of an on-site time-sharing system is crucial to providing reliable service to experimenters. Only through on-site installations can control be exercised over the reliability of the hardware and of the system software. (3) the necessity of an on-site installation, combined with the meagre finances of most researchers in the behavioral sciences, requires that concessions be made in the design of hardware and software to provide economical service. Historically, these concessions have been the development of a language tailored to the needs of those programming experimental-gaming situations and the development of a single-language time-sharing system for that purpose. Further, the cost of extremely reliable mass-storage devices has prohibited their use thus far. (4) In addition to meeting the above constraints of fast time-shared operation on a small computing system, the language utilized by the system must (a) have provision for the usual computational requirements of a behavioral-science experiment. For example, the experimental program normally modeled in gaming situations requires that the language have facilities for matrix manipulations and elementary string operations. (b) The language must be relatively easy for novice programmers to learn and use; that is, behavioral scientists with little or no background in the computing sciences must readily comprehend the language. (c) The program should be self documenting, i.e., the language in which the model is programmed must be general enough so that the code is virtually machine independent. (d) The language must allow a limited real-time report of subject performance to the experimenter. The experimenter must be able to sample a subject's performance while the experiment is in progress; further, he must do so without the subject's recognizing that his performance is being monitored. (e) The language must enable the experimenter easily to exercise control over segments of an in-progress experiment. Very frequently, in the course of an experiment, the need arises for the experimenter to modify the nature of the experiment itself or to communicate with the subject by sending messages to him. This requirement and the previous one translate into the necessity of allowing a controlled amount of interaction between the time-shared terminals used by the subjects and the terminal used by the experimenter. (f) The language must permit a controlled amount of subject-to-subject interaction for bargaining and other small-group experiments. Again, this translates into a need for some degree of interaction among the users of the time-sharing system. (g) The system must store all relevant information on subject behavior in machine-readable form for subsequent data analysis. Data on subject behavior is usually analyzed statistically at the conclusion of the experiment, often on another com-

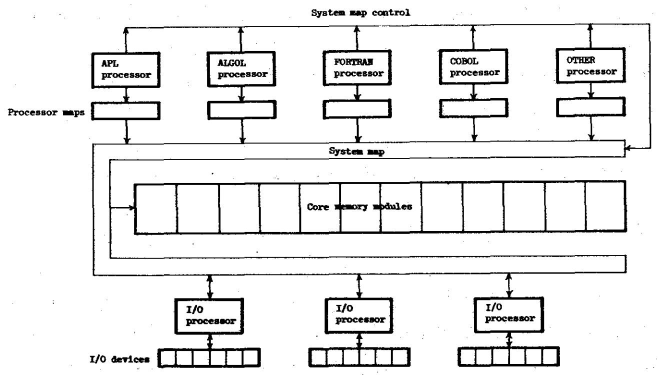


Figure 8—A time-shared array of language processors

puter and the need for any keypunching is eliminated if all information can be recorded on a peripheral device in machine-readable form. (h) The language must interface the experiment to a variety of special-purpose input-output peripherals, such as galvanic skin response and other analogue equipment, video displays, sense switches and pulse relays for slide projectors, reward machines and the like. (i) A final requirement of the experimental-control language is the need for reentrancy. Reentrant coding permits the use of shared functions among users of the time-sharing system, thereby conserving core.

AN ARRAY OF LANGUAGE PROCESSORS (ALPS)

The Meta-APL Time-Sharing system which has been described represents a conceptual prototype for a time-shared array of language processors. (See Figure 8) The ALPS consist of an array of independent dedicated language processors which communicate with the outside world and/or each other exclusively through core memory. These processors are completely independent, and could indeed be different machines. The physical memory consists of core memory plus secondary storage and is divided in 4K blocks allocated to the various processors through the system map. These blocks appear to each processor through the system map as a continuous memory which is in turn paged via the processor maps.

Let us consider the successive transformations of an

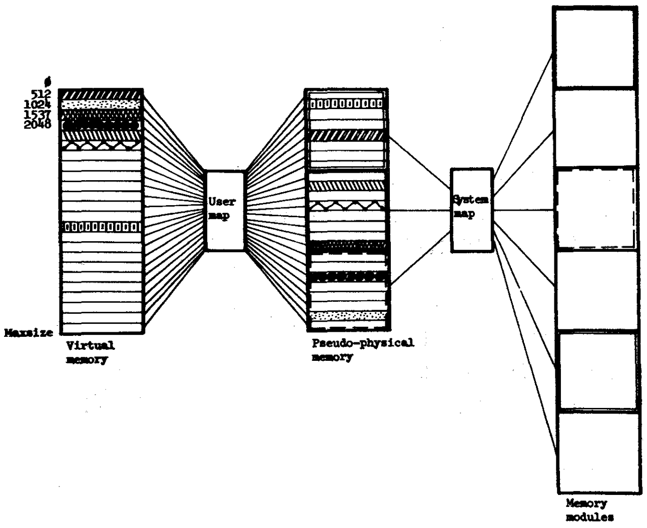


Figure 9—Memory mappings

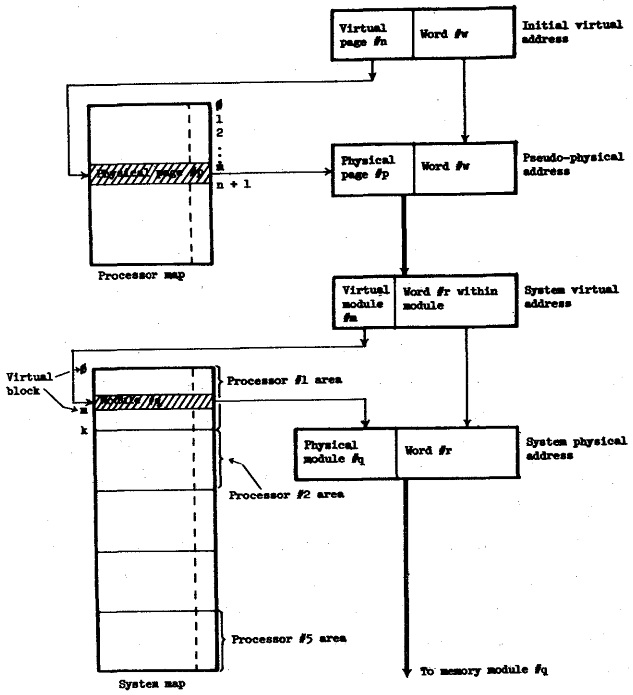


Figure 10—General address computation

address issued by a language processor (Figure 9). The logical page number field of the address is used to access a location in the processor map whose contents represent the physical page number and are substituted in the page field of the address. The reconstituted address is then interpreted by the system map in a different way: an initial field of shorter length than the page number represents the virtual module number and is used to access a location within the system map which substitutes automatically its contents (physical module number) for the original field.

This physical module number is then used to access a memory module while the rest of the address specifies a word within the module.

Note that if no processor is expected to monopolize all of core memory, there will be many more physical memory modules than virtual ones for each processor.

The physical module number field will then be much larger than the original virtual module number so that the size of physical memory which can be accessed by any one processor over a period of time can be much larger than its maximum addressing capability, as defined by the length of its instruction address field.

A user logging in on one of the ALPS terminals obtains the attention of the corresponding I/O processor and communicates with it via the system-wide command language. Once input has been completed a language processor is flagged by having the user's string assigned to its portion of the system map. Switching between languages is handled as a transfer within the system's virtual memory and therefore implemented as a mere system map modification. For this reason, all map handling routines are common to all processors, including I/O processors. Map protection is provided by hardware lock-out.

Finally, the modularity of the system provides a high degree of reliability. Core modules can be added or removed by merely marking their locations within the system map as full or empty. The same holds for the languages processors; to each of them corresponds a system map block containing one word per core module that may be allocated to it up to the maximum size of the processor's storage compatible with its addressing capabilities. Furthermore, each language processor say #n-1 might have access to an interpreter for language n written in language n-1 (mod the number of language processors), so that, should processor n be removed, processor n-1 could still interpret language **n**, the penalty being in this case a lower speed in execution.

Similarly, the number of I/O processors can be adjusted to the needs for input-output choices, terminals, or secondary storage choices.

It shou'd be noted that although the system is asynchronous and modular, the modules, memory as well as processors need not be identical. In fact, it seems highly desirable to use different processor architectures to interpret the various languages.

In summary, the essential features of the ALPS time-sharing system are:

—automatic address translation via a multilevel hardware mapping system; each user and each processor operates in its own virtual storage.
—the type, architecture, and characteristics of each processor are optimized for the computer language that it interprets, allowing for maximum technological efficiency for the language considered.

This is essentially a low-cost system since each processor has to worry about a single language, and the overhead for language swapping is reduced to merely switching the user to a different processor, allowing a smaller low-cost processor to operate efficiently in this environment.

ACKNOWLEDGMENTS

The authors are indebted to Messrs. A. C. Hoggatt and F. E. Balderston, who encouraged the development of the APL system. Appreciation is due to Mr. M. B. Garman for editorial suggestions and stimulating discussions, to Messrs. E. A. Stohr and Simcha Sadan for their work on APL operators, and to Mr. R. Rawson for his effort on the temporary IO processor. We are also grateful to Mrs. I. Workman for her very careful drawings and typing.

APPENDIX—SIMPLIFIED BNF META-APL EXTERNL SYNTAX

Notes: (1) {...} denotes ∅ or 1 times .... **( )** are symbols of Meta-APL.
(2) Lower-case letters are used for comments to avoid lengthy repetitions.
(3) cr denotes a carriage-return.
(4) PROGRAM in BNF is equivalent to PROCESS in the text.

Group 1

```
⟨FUNCTION BLOCK⟩            ::= ⟨FUNCTION DEFINITION⟩ ⟨STATEMENTS⟩ ∇
⟨PROGRAM BLOCK⟩             ::= {⟨PROGRAM DEFINITION⟩} ⟨PROGRAM⟩
⟨PROGRAM DEFINITION⟩        ::= ⟨PROGRAM NAME⟩ {(⟨PARAMETER LIST⟩)}
⟨PROGRAM NAME⟩              ::= ⟨NAME⟩
⟨PROGRAM⟩                   ::= ⟨STATEMENT⟩ | ⟨PROGRAM⟩ ⟨STATEMENT⟩
                              | ⟨PROGRAM⟩ ⟨FUNCTION BLOCK⟩
⟨STATEMENT⟩                 ::= {⟨LABEL⟩} ⟨STATEMENT LINE⟩ cr
⟨STATEMENT LINE⟩            ::= ⟨BRANCH⟩ | ⟨SPECIFICATION⟩ | ⟨SYSTEM COMMAND⟩
                              | ⟨IMMEDIATE⟩
⟨SYSTEM COMMAND⟩            ::= ⟨see system commands⟩
⟨BRANCH⟩                    ::= → ⟨EXPRESSION⟩
⟨IMMEDIATE⟩                 ::= ⟨EXPRESSION⟩ cr | ⟨EXPRESSION⟩ ; ⟨IMMEDIATE⟩
⟨SPECIFICATION⟩             ::= ⟨VARIABLE⟩←⟨EXPRESSION⟩ cr | ⟨OUTPUT SYMBOL⟩
                               ←⟨EXPRESSION⟩
```

Group 2

```
⟨NUMBER⟩                          ::= ⟨DECIMAL FORM⟩ | ⟨EXPONENTIAL FORM⟩
⟨DECIMAL FORM⟩                    ::= {⟨INTEGER⟩} . {⟨INTEGER⟩} | ⟨INTEGER⟩
⟨INTEGER⟩                         ::= ⟨DIGIT⟩ | ⟨INTEGER⟩ ⟨DIGIT⟩
⟨DIGIT⟩                           ::= ∅ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
⟨EXPONENTIAL FORM⟩                ::= ⟨DECIMAL FORM⟩ E ⟨INTEGER⟩
⟨VECTOR⟩                          ::= ⟨SCALAR VECTOR⟩ | ⟨CHARACTER VECTOR⟩
                                    | ⟨EMPTY VECTOR⟩
⟨SCALAR VECTOR⟩                   ::= ⟨NUMBER⟩ | ⟨SCALAR VECTOR⟩ ⟨SPACES⟩ ⟨NUMBER⟩
⟨SPACES⟩                          ::= ⟨SPACE⟩ | ⟨SPACES⟩ ⟨SPACE⟩
⟨SPACE⟩                           ::= ⟨one blank⟩
⟨EMPTY VECTOR⟩                    ::= '' | ι∅ | ρ ⟨SCALAR⟩
⟨CHARACTER VECTOR⟩                ::= '⟨CHARACTER STRING⟩'
⟨CHARACTER STRING⟩                ::= ⟨CHARACTER⟩ | ⟨CHARACTER STRING⟩ ⟨CHARACTER⟩
⟨CHARACTER⟩                       ::= ⟨LETTER⟩ | ⟨DIGIT⟩ | ⟨SYMBOL⟩
⟨NAME⟩                            ::= ⟨LETTER⟩ | ⟨LETTER⟩ ⟨ALPHANUMERIC STRING⟩
⟨ALPHANUMERIC STRING⟩             ::= ⟨ALPHANUMERIC⟩ | ⟨ALPHANUMERIC STRING⟩
                                    ⟨ALPHANUMERIC⟩
⟨ALPHANUMERIC⟩                    ::= ⟨LETTER⟩ | ⟨DIGIT⟩
⟨NUMERICAL TYPE⟩                  ::= ⟨NUMBER⟩ | ⟨VECTOR⟩
⟨LOGICAL⟩                         ::= ∅ | 1
⟨LABEL⟩                           ::= ⟨NAME⟩ :
⟨IOSYMBOL⟩                        ::= ⟨INPUT SYMBOL⟩ | ⟨OUTPUT SYMBOL⟩
⟨OUTPUT SYMBOL⟩                   ::= □ | ⟨DEVICE ID⟩
⟨INPUT SYMBOL⟩                    ::= □ | quote-quad
⟨DEVICE ID⟩                       ::= ⟨undefined as yet⟩
```

Group 3

```
⟨SCALAR OPERATOR⟩                 ::= ⟨MONADIC SCALOP⟩ | ⟨DYADIC SCALOP⟩
⟨MONADIC OPERATOR⟩                ::= ⟨MONADIC SCALOP⟩ | ⟨MONADIC MIXEDOP⟩
                                    | ⟨MONADIC EXTENDED SCALOP⟩
⟨DYADIC OPERATOR⟩                 ::= ⟨DYADIC SCALOP⟩ | ⟨DYADIC MIXEDOP⟩
                                    | ⟨DYADIC EXTENDED SCALOP⟩
⟨MONADIC SCALOP⟩                  ::= + | − | × | ÷ | ⌈ | ⌊ | * | log | | | ! | ? | ○ | ∼
⟨DYADIC SCALOP⟩                   ::= ⟨MONADIC SCALOP⟩ | ∧ | ∨ | nand | nor | < | ≤ |
                                    | ≥ | $> | $≠
⟨EXTENDED SCALAR OPERATOR⟩        ::= ⟨MONADIC EXTENDED SCALOP⟩
                                    | ⟨DYADIC EXTENDED SCALOP⟩
⟨MONADIC EXTENDED SCALOP⟩         ::= ⟨SCALAR OPERATOR⟩ /
                                    | ⟨SCALAR OPERATOR⟩ /[COORD]
⟨DYADIC EXTENDED SCALOP⟩            | ⟨DYADIC SCALOP⟩ . ⟨DYADIC SCALOP⟩
                                    | ○. ⟨DYADIC SCALOP⟩
⟨COORD⟩                           ::= 1 | 2
⟨MIXED OPERATOR⟩                  ::= ⟨DYADIC MIXEDOP⟩ ⟨MONADIC MIXEDOP⟩
⟨MONADIC MIXEDOP⟩                 ::= ρ | , | ∼ | ϕ | transpose/grade-up/grade-down | ∇ | Δ
⟨DYADIC MIXEDOP⟩                  ::= ρ | , | ι | ϕ | transpose | / | \ | ↑ | ↓ | ϵ | ⊥ | ⊤ |
```

Group 4

```
⟨EXPRESSION⟩                ::= ⟨NUMERICAL TYPE⟩ | ⟨VARIABLE⟩ | ⟨INPUT SYMBOL⟩
                              | ⟨MONADIC EXPRESSION⟩ | ⟨DYADIC EXPRESSION⟩
                              | ⟨MONADIC EXPRESSION⟩ | ⟨Ø-ARG FUNCTION⟩
⟨MONADIC EXPRESSION⟩        ::= ⟨MONADIC OPERATOR⟩ ⟨EXPRESSION⟩
                              | ⟨1-ARG FUNCTION⟩
⟨DYADIC EXPRESSION⟩         ::= ⟨EXPRESSION⟩ ⟨DYADIC OPERATOR⟩ ⟨EXPRESSION⟩
                              | ⟨ALPHANUMERIC STRING⟩ ⟨RELOP⟩
                                ⟨ALPHANUMERIC STRING⟩
                              | ⟨LOGICAL⟩ ⟨RELOP⟩ ⟨LOGICAL⟩ | ⟨2-ARG FUNCTION⟩
⟨RELOP⟩                     ::= < | ≤ | = | ≥ | > | ≠
⟨FUNCTION NAME⟩             ::= ⟨NAME⟩
⟨Ø-ARG FUNCTION⟩            ::= ⟨FUNCTION NAME⟩ {(⟨PARAMETER LIST⟩)}
⟨1-ARG FUNCTION⟩            ::= ⟨FUNCTION NAME⟩ {(⟨PARAMETER LIST⟩)}
                                ⟨EXPRESSION⟩
⟨2-ARG FUNCTION⟩            ::= ⟨EXPRESSION⟩ ⟨FUNCTION NAME⟩
                                {(⟨PARAMETER LIST⟩)} ⟨EXPRESSION⟩
⟨FUNCTION DEFINITION⟩       ::= ∇{⟨VARIABLE NAME⟩}←{⟨VARIABLE NAME⟩}
                                ⟨FUNCTION NAME⟩ {(⟨PARAMETER LIST⟩)}
                                ⟨VARIABLE NAME⟩ {⟨LOCAL VARIABLES⟩}
                              | ∇{⟨VARIABLE NAME⟩}←⟨FUNCTION NAME⟩
                                {(⟨PARAMETER LIST⟩)} {⟨VARIABLE NAME⟩}
                                {⟨LOCAL VARIABLES⟩}
⟨VARIABLE NAME⟩             ::= ⟨NAME⟩
⟨VARIABLE⟩                  ::= ⟨VARIABLE NAME⟩ | ⟨INDEXED VARIABLE⟩
⟨INDEXED VARIABLE⟩          ::= ⟨NAME⟩ [⟨EXPRESSION⟩ {;⟨EXPRESSION⟩}]
⟨PARAMETER LIST⟩            ::= ⟨PARAMETER NAME⟩ | ⟨PARAMETER LIST⟩
                               ,⟨PARAMETER NAME⟩
⟨LOCAL VARIABLES⟩           ::= ;⟨VARIABLE NAME⟩
                              | ⟨LOCAL VARIABLE⟩ ; ⟨VARIABLE NAME⟩
⟨PARAMETER NAME⟩            ::= ⟨VARIABLE NAME⟩
⟨LETTER⟩                    ::= A | B | C | D | E | F | G | H | I | J | K | L | M
                              | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
⟨SYMBOL⟩                    ::= ] | [ | ← | → | + | × | / | \ | , | . | ¨ | ¯ | <
                              | > | ≠ | ≤ | ≥ | = | ) | ( | ∨ | ∧ | ϵ | : | ⊥ | ⊤
                              | _ | ↓ | ↑ | ι | ~ | ○ | ? | ⌊ | ⌈ | − | * | ρ | ∪
                              | ∩ | α | ⊂ | ⊃ | ÷ | ω | ⎕ | ∘ | ∇ | Δ | ′ | ( | )
```

Designing a large-scale on-line real-time system

by SUMIO ISHIZAKI

The Fuji Bank, Limited
Tokyo, Japan

OBJECTIVES AND BACKGROUND OF TOTAL BANKING SYSTEM

The Fuji Bank, Ltd., now employs a computerized total banking system. The objective of the system and the main fields of application are described below.*

Major business activities covered by the system

Data processing

All kinds of deposit accounts, loans, domestic remittances, foreign exchange, stock transfer, management of portfolio, calculation of depreciation on furnishings and equipment, payroll, etc.**

Management information system

(a) Organization of Various Data Filing and Retrieval Systems.
- Customers information files
  - individual customers
  - corporate customers
- Management reports file
- Corporate accounts file
- Personnel information file

(b) Management Science
- Forecasting macro-economic activities
- Forecasting deposits
- Estimation of market share
- Selection of branch sites
- Study of personnel requirements
- Financial analysis of corporate customers

Automated customer services

- Services to correspondent banks
- Billing services for professionals, credit cards, electricity, rent for living quarters, pension funds, hospital fees, etc.
- Collection of tuition fees
- Scoring of school entrance examinations
- Repayments to scholarship funds
- Inventory analysis and control, etc.

Objectives of total banking system

Fuji Bank has invested over $30 million in computerization for three major objectives.

Cost saving

Cost reduction is the first objective of computerization. Most important is the saving in personnel expenses. The imbalance in Japan's labor market has been getting worse from year to year. Last spring, the number of high school graduates intending to take up jobs was only 657,000 against 4,701,000 openings, so that only 14 percent of demand could be met.

The increase in the volume of business would have required the addition of 2,000 new employees to the staff of Fuji Bank over the next six years if computerization had remained within the limits of an off-line system. In view of the conditions in the labor market, it would have been nearly impossible to recruit 2,000 new employees in addition to the 1,500 needed each year for filling the vacancies created by retirement.

* Fuji Bank has 3 million ordinary deposit accounts for which passbooks are used and on which interest is paid. In addition to ordinary deposits and withdrawals, these accounts are used for the payment of all kinds of bills (telephone, electricity, gas and water), for the settlement of credit card balances and other automatic transfers. Ordinary deposits account for 53 percent of all individual deposits.

** In Japan, instead of mailing checks, remittances are usually sent by teletype or by a computer message switching system.

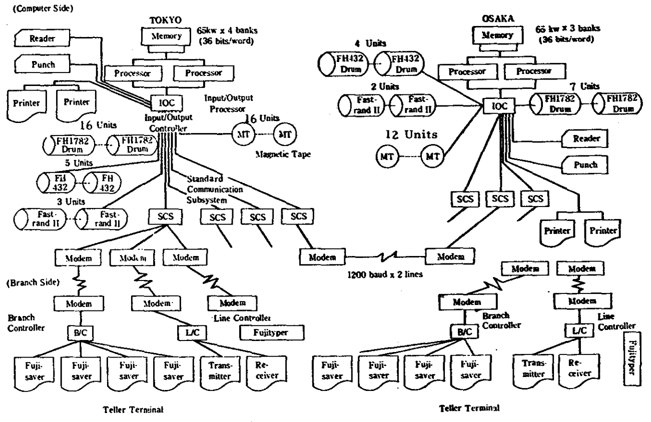


Figure 1—Layout of multi-processor system

By installing an on-line real-time system for all major business operations, even the growing workload can be handled with the present number of employees in the next six years, thus saving not only personnel costs but also the additional office space which would have been required.

Better management

The following advantages are gained by large-scale computerization:

(a) Greater accuracy in office work
Prevention of errors and unauthorized payments
(b) Greater speed in office work
Increase in labor productivity
(c) Better management reports
Management reports whose preparation by hand would simply be impossible can be compiled quickly and exactly.
(d) New personnel management
Computerization relieves the staff of monotonous or "mechanical" routine work. An on-line system fitted with various subordinate checking systems significantly decreases the errors in clerical work; even inexperienced operators quickly become experienced and the burden on the supervisory staff is reduced.

Customer services

(a) The principle of accurate and fast data processing can be applied to customer services.
An on-line real-time system not only reduces the possibility of faulty or unauthorized operations but also reduces the customers' waiting time because all major banking operations, including ledger retrieval, can be computerized.
(b) Through computerization, central control of all deposit ledgers is achieved so that customers can be allowed to use any one of the 206 branches of Fuji Bank in Japan for unlimited deposits and withdrawals.
(c) Customers of the 206 branches can use any branch for making remittances to customers of other branches and remittances can be effected within seconds.
(d) Organization of new customer services: The growth of banking activities involves new bank services such as consumer credits, payroll deposits, the credit card business and automatic debiting of public charges (telephone, electricity, gas and water) whose increasing volume could hardly be handled without computers. As a matter of fact, many of the new services have been developed in response to computerization.
(e) In this sense, the automated business procedures described above can be regarded as computer-related customer services.

OUTLINE OF TOTAL BANKING SERVICES

The following discussion will focus on the role of the on-line real-time system within the total banking system.

Nationwide network

The Fuji Bank System, the largest private system in operation, consists of a network of about 1,000 on-line teller terminals in the Bank's 206 branches linked by a data communication network comprising 516 lines with a capacity of 1,200 or 200 bits per second.

System configuration

The main computer of the system consists of four UNIVAC 1108 multi-processors installed at the computer centers located in Tokyo and Osaka, two units at each center (Figure 1). The system further includes two units IBM 360, three units UNIVAC 418, two units NCR Century, three units UNIVAC 9300 and other for batch processing.

Configuration of the computers

Various I/O units such as 23 FH 1782 magnetic drums, 5 Fastrand II drums, 9 FH 432 drums, 17 standard communication subsystems and 28 VIII C-magnetic tape units are connected with the UNIVAC 1108 multi-processors and moreover connected with processors of different systems acting as back-ups in case of failure.

CHARACTERISTICS OF SYSTEMS DESIGN

The objective of the systems design was to determine among the possible alternatives a systems capability which would meet the requirements of maximum performance and lowest cost.

Diversified applications and voluminous data processing (high traffic rate)	compromise	Systems economy and efficiency (quick response)

Variety of applications

Scope of application

The present system is used for all major banking operations including all kinds of deposits (checking accounts, ordinary deposits, time deposits, deposits at notice, savings accounts, special deposits for tax payment), domestic remittances and preparation of balance sheets for all branch offices.

Combination of inquiry and answer system and message switching system

In the actual use of the installation, an organic combination of several systems with different modes or operations, e.g., the inquiry-and-answer system and the message switching system, will be necessary. Below are a few examples of such combinations:

(a) For funds paid in through the remittance system from a distant location, the computer retrieves the account of the payee and automatically makes the entry into the customer's ledger.
(b) Another example is the so-called network service which allows deposits and withdrawals at any of the Bank's 206 branch offices. This system relies on separate data files at each of the two computer centers in Tokyo and Osaka. If a customer of a branch in Tokyo withdraws money at a branch in Osaka, the Osaka computer must also retrieve and update the data file of the Tokyo Center. The procedure involves the following steps which are taken automatically and almost instantaneously: Osaka branch (deposit section)⇄Osaka Computer (deposit section⇄remittance section)⇄Tokyo Computer (remittance section⇄deposit section). This case illustrates the assimilation of deposits and remittances.

Simultaneous real-time processing and batch processing

Another feature of the system is the possibility of providing access to the computer file used for real-time processing also for batch processing. This has improved the performance of the system and opened the way to multi-programming including both real-time and batch programs. Practically, the system allows the automatic posting of salaries or stock dividends in customers' accounts, debiting customers with electricity, gas, water and telephone charges and credit card purchases, debiting of large batches of checks returned from the clearinghouse, and dispatch of accumulated items to a branch office which had been closed on account of a local holiday. This can be done not only before or after business hours but also at other times.

Handling of complex office work

Although used for a great variety of applications, the operation of the terminals has been standardized as much as possible. As mentioned above, the applications include the handling of several types of deposits, such as checking accounts, ordinary and time deposits, transfer from one account to another, remote processing of files and other on-line as well as off-line operations.

Efficiency and economy through order-made system

Efficiency and economy have been the ultimate objectives in having the entire system, from the system design to the terminals, made to the Bank's specifications. In this way, the system can handle greatly diversified operations (in addition to all kinds of deposits and domestic remittances, foreign exchange and loans) all involving a high traffic rate.

Terminals

In view of the variety of applications and the large volume of transactions, two types of terminals have been adopted. The first is the deposit terminal designed for processing fixed-length messages with the emphasis on efficiency. The other is the remittance terminal built for processing variable-length messages with the emphasis on flexibility. But the remittance terminal can also be used for handling deposits and, by shifting the connector, can serve as a transmitter as well as a receiver, depending on the conditions at a particular time.

The total number of terminals required for the system amounts to about 1,000 units so that a reduction in the unit costs of the terminals results in substantial savings. The deposit terminal, called Fujisaver, costs $5,000, the remittance terminal, named Fujityper, $3,000. The Fujisaver, in particular, possesses several noteworthy features. It simultaneously imprints the passbook, the journal and a slip; its panel of indicator lamps (Figure 2) shows the condition of the system and, if necessary, the machine locks until the required corrective action is taken.

Supervisory program

While the overhead load of the operating system (EXEC 8) has been reduced, the programming burden of the user has been lessened by incorporating a supervisory program into the system, thus improving its performance (Figures 3 and 4).

Project management

UNIVAC undertook the development of the computers and Oki Electric Co. the development of the terminals while Fuji Bank assumed responsibility for the user program. Fuji Bank was also in charge of the overall management of the project whose completion required about six years, due to the size of the project and the necessity of developing entirely new banking terminals.

Peak workload processing techniques

Computer Load Distribution

Since the system was to cover about 6 million accounts and an average of 650,000 transactions a day was foreseen, the load was distributed between two computer centers, Tokyo and Osaka, each equipped with a UNIVAC 1108 multi-processor system. A further reason for this arrangement was the high cost in Japan of long-distance communication lines used exclusively for data transmission.

Load distribution related to terminals

In order to avoid too frequent interruptions of the computers ordinarily doing batch processing, both deposit and remittance terminals have been equipped with buffer memories (576 characters) so that transmission is in block units.

Development of supervisory program

In addition to the standard operating system, the above-mentioned supervisory program has been developed which exercises activity control over the real-time program as well as input/output control.

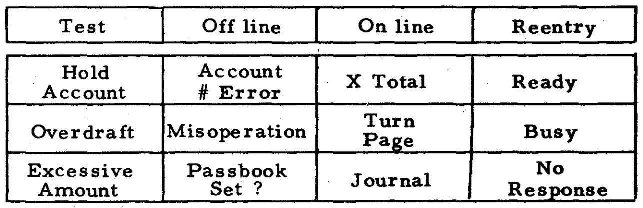
Reduction of access frequency to random access file

Most of the deposit ledgers are recorded on the 23 high-speed magnetic drums with an average access time of 17 ms., but in view of the extremely high traffic rates, the access frequency to the random access file should be reduced as much as possible. Each magnetic drum contains the ledgers of the customers of about 11 branches. For the retrieval of an account, the location of the drum unit is ascertained from the code number of the branch recorded on the magnetic core and the table showing the corresponding drum unit. The code number of the account is divided by the number of blocks per drum and the account is located by transferring the remaining "*n*th" block to the magnetic core. This procedure speeds up the operation (Figure 5).

For time deposits, the drum number is identical with the number of the deposit certificate which makes direct addressing possible and speeds up the operation.

Balance between real-time and batch processing

In order to achieve the twin objectives of lessening the load at peak hours and, at the same time, increasing



Test	Off line	On line	Reentry
Hold Account	Account # Error	X Total	Ready
Overdraft	Misoperation	Turn Page	Busy
Excessive Amount	Passbook Set ?	Journal	No Response

Figure 2—Fujisaver—Indicator lamp arrangement

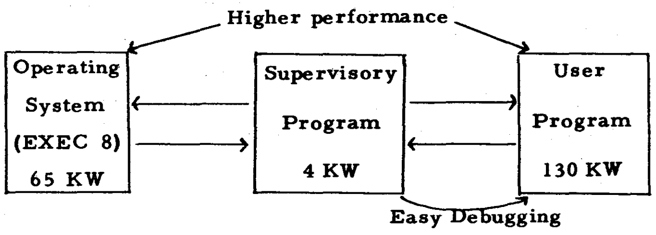


Figure 3—Program configuration

efficiency, the proportion of real-time to batch processing has been fixed with great care for each specific operation.

Programming by Assembler Language

A compiler is more advantageous for programming and program maintenance but for the most efficient handling of a large random access file by "bit" units and for increasing throughput at peak hours, an assembler proved much more efficient. Hence, the entire program amounting to over 50,000 steps has been coded by an assembler.

Traffic simulation

The results of a traffic simulation for which a GPSS II was used are shown in Figure 6. The simulation proved that despite a heavy future increase in the data volume the life cycle of the system can be prolonged considerably by adding more magnetic drums such as FH 1782 or Fastrand II or their control units.

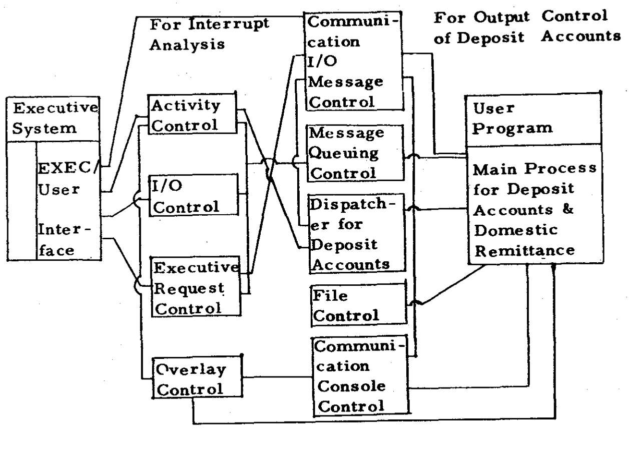

For Interrupt Analysis
Communication I/O Message Control
For Output Control of Deposit Accounts
Executive System
EXEC/User Interface
Activity Control
I/O Control
Executive Request Control
Overlay Control
Message Queuing Control
Dispatcher for Deposit Accounts
File Control
Communication Console Control
User Program
Main Process for Deposit Accounts & Domestic Remittance

Figure 4—Block diagram of supervisory program

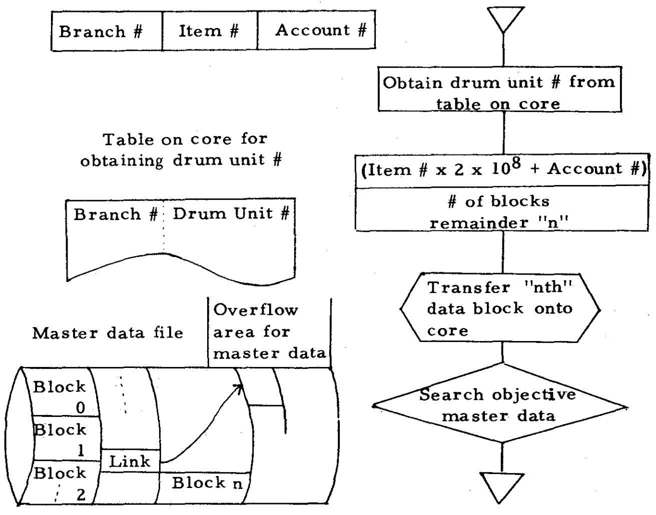

Branch #
Item #
Account #
Obtain drum unit # from table on core
Table on core for obtaining drum unit #
Branch #
Drum Unit #
(Item # x 2 x 10⁸ + Account #)
# of blocks remainder "n"
Master data file
Overflow area for master data
Transfer "nth" data block onto core
Block 0
Block 1
Block 2
Link
Block n
Search objective master data

Figure 5—Master data retrieval sequence

Techniques for economizing memory capacity

Main memory unit

In order to reduce the demands on the main memory unit and simplify program maintenance, the common parts of different items have been consolidated as much as possible and brought together in a subroutine.

Within the limits of the queuing time resulting from the traffic volume, program overlay has been attempted for the FH 432 magnetic drums with an average access time of 4.25 ms.

Random access file

Similar to the main memory unit, a master file system consolidating the common items of the random access file has been adopted. All items subject to frequent change with regard to overflow, item length and data numbers are recorded in the slave file. A continuous chaining of the two files is possible through the link address. All items are recorded and processed by "bit" units.

For the master file, one FH 1782 magnetic drum contains the records of 11 branch offices and up to 299,915 accounts. For retrieval, each magnetic drum has been divided into 3,157 blocks; this number corresponds to the number of items and accounts divided by a multiple of 11. Each block consists of 95 accounts, a number fixed by taking into consideration the buffer capacity of the core memory and the frequency of access to the drum. The starting position of block No. 0

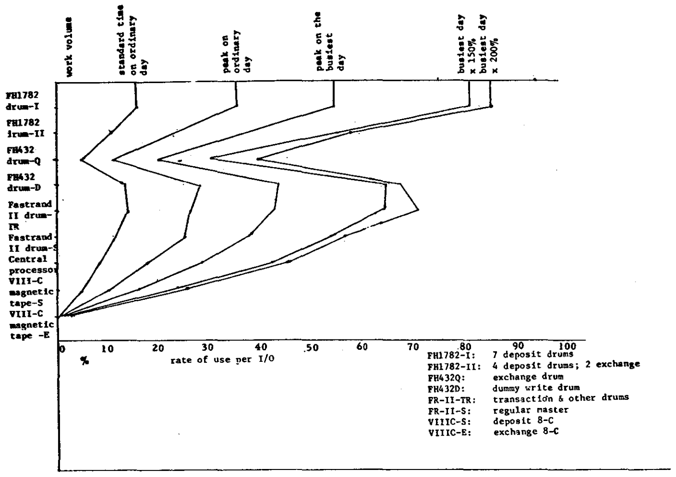


Figure 6—Work volume and rate of use per I/O

has been staggered by 287 blocks for each branch so as to average the number of accounts in each block.

Prevention of failure

In order to reduce as much as possible loss of time from failure and to expedite recovery, arrangements have been made in three fields, namely, hardware, software and business procedures.

Hardware

The central processors of the multi-processor system back each other up so that even in case the I/O Controller breaks down, either Processor No. 1 or Processor No. 2 can step into the breach. One of the four memory banks of the main memory is usually assigned to batch processing but if one of the other three banks fails, the batch processing can be suspended at once and the bank switched to the operating system or the real-time program. The standard communication subsystems, and the equipment ordinarily used for batch processing, such as various magnetic drums, magnetic tapes and printers, are also connected with different systems and can be switched immediately to real-time processing if an accident occurs.

The data communication network comprises different systems for deposits and remittances which lessens the probability that both operations will be interrupted at the same time. In case the equipment at a branch office gets out of order, a neighboring branch office can take over transmission and reception of messages according to a prearranged plan.

Similar precautions have been taken for the terminals. By shoring up the functions of off-line processing, passbook entries can be made and operational errors or over-payment prevented even if a failure occurs in the computer or the transmission lines.

Software

Quick recovery

Since the system handles more than half the Bank's daily business, it is very important to keep interruptions due to breakdowns to a minimum. No matter when the failure occurs, a comprehensive and instantaneous check must be possible to ascertain exactly how far the operation had proceeded and at what point the process came to an end so that the trouble can be corrected not only as quickly as possible but also without missing a single input or repeating the same input.

To this end, all transactions are recorded in strict chronological sequence and full detail on both magnetic drums and magnetic tape. If a failure occurs, the drum master file is checked against the transaction file and in case of discrepancies, the master file can be corrected immediately. In most cases, the system can be repaired and restarted in 10-20 minutes.

Prevention of complete breakdown by partial failure

Because so many different applications are linked to the same nationwide network, care has been taken lest a failure in one part of the system affect the entire network. (a) Recovery during continuing real-time processing.

Repairs can be made while real-time processing continues so that the entire operation need not be shut down on account of the breakdown of a single unit. If, e.g., one of the magnetic drums gets out of order, the files of the branches recorded on this particular drum will be transferred to a spare drum, reproducing

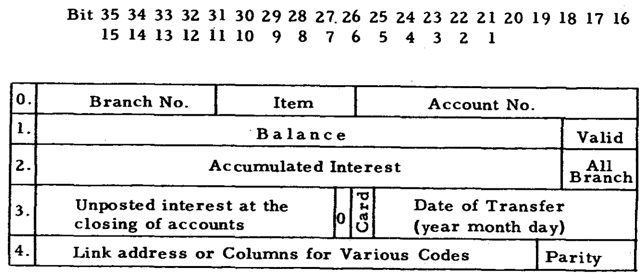

Bit 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
0.
Branch No.
Item
Account No.
1.
Balance
Valid
2.
Accumulated Interest
All Branch
3.
Unposted interest at the closing of accounts
0
Card
Date of Transfer (year month day)
4.
Link address or Columns for Various Codes
Parity

Figure 7—Layout of deposit master file

the balances brought forward from the previous day and all transactions from the beginning of the day until the time of the failure from the magnetic tape. This makes it unnecessary to wait for the physical repair of the faulty drum. The arrangement that all transactions are recorded both on magnetic tape and on magnetic drums which forms part of the supervisory program, makes this recovery procedure possible.

(b) Since different applications are handled by real-time processing, care must be taken in the programming that trouble in one business routine will not adversely affect others. In and by themselves, the different business activities are independent of each other but actually there is a great deal of interaction. For instance, a remittance sent through the remittance system is automatically debited to the deposit account of the payor and credited to the account of the payee so that the transaction necessarily involves the deposit files. Special techniques, therefore, are required to prevent a breakdown in one sector from shutting down others.

(c) Since remittances and the so-called network service necessitate a constant exchange of messages between the Tokyo and Osaka computer centers, the failure of one computer should not influence the other. If money is transferred from a branch within the limits of the Tokyo Center to a branch belonging to the Osaka Center, a breakdown of the Osaka computer should not cause the transfer to go astray or the same sum to be transferred twice. The prevention of such accidents must be planned in the program.

Assimilation of ordinary operations and recovery process

There is much to be gained from making the recovery process for correcting breakdowns as similar as possible to the ordinary processing procedure. First, such an arrangement will economize the capacity of the expensive main memory and facilitate program maintenance in case the system or the program is modified. Secondly, it will make the recovery process easier for the computer operators as well as the operators of the branch terminals. It is of particular value for the Osaka Center which has no programmers.

Office routine

Care has been taken to prevent the disruption of the everyday office work even for the short time it takes to correct a failure. For ordinary deposits, a list of unposted items is prepared during the night—compilation of the list takes about four hours—and distributed to all tellers before nine o'clock the next morning. A similar list is prepared for current deposits showing outstanding balances after debiting public charges or checks returned during the night from the clearinghouse. In this way, the tellers can comply with demands for payment even if the computer or the communication lines are out of order.

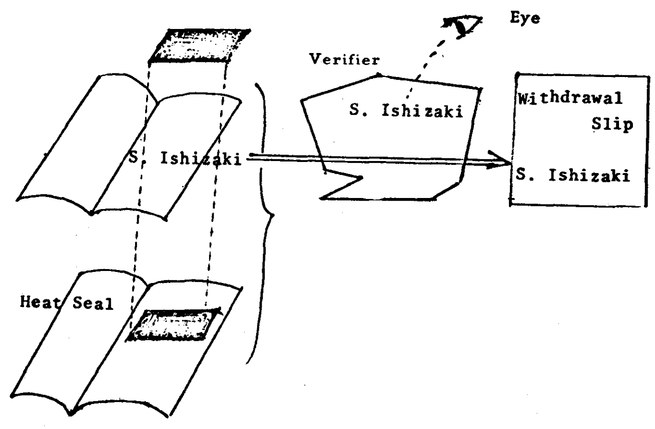


Figure 8—Signature verification system

Automation of related operations

For the best performance of the on-line system, it is also necessary to improve the functions of related operations. The real-time system has reduced to nearly zero the time needed for retrieving the ledger from the files, but it is only in the case of deposits that no further search is required. For withdrawals, it is also necessary to check the signature file and this prolongs the time the customer must be kept waiting. In 1963, therefore, Fuji Bank, in cooperation with Canon Inc., began the development of a signature verification system. The signature inscribed in the passbook is covered with a black vinyl seal so that it is invisible to the naked eye and can only be read through a newly developed verifying machine. Thus, the customer's signature can be verified at the same time that he presents his passbook. (Figure 8).

FUTURE DEVELOPMENTS

Total banking system and MIS

By transforming daily operations into a real-time system, an accurate and up-to-date customers information file can be prepared. But the information obtained through on-line processing consists mainly of statistical data related to deposits and withdrawals. For a really useful customers information file, a consolidated file would have to be prepared which would also include descriptive data such as the extent to which the customer uses automatic debiting of public charges, consumer loans, rental safes or credit cards, information on the customer's occupation, family status, income, relations with other banks, codes of business clients and code of the bank officer in charge of the account. The system would also have to provide for fast and simple retrieval. But the number of deposit accounts alone covered by the present system amounts to six million and the addition of even a single item would require an enormous investment for input and maintenance. The inputs, therefore, have to be selected with great care after examining repeatedly how often the information will be used. At present, the number of accounts covered by the customers information file is gradually being expanded. In the future, when the file will also cover potential customers, it may play an important role in assessing market potential.

By organizing various data banks and combining them with IR or management science techniques, the system could be expanded into a complete MIS, our goal after the installation of the on-line system.

Common Data Transmission System for All Banks

The second phase in the development of the on-line systems of individual banks would be the organization of a common data transmission system linking all 87 Japanese banks. Such a system would require the installation of huge computers able to process 700,000-1,350,000 transactions a day. The system would have to be able to effect an exchange between input/output messages of different formats. The year 1973 has been set as target date for the implementation of this system which, if completed, would represent one of the world's most extensive data communication systems.

PERT—A computer-aided game

by J. A. RICHTER-NIELSEN

Technical University of Denmark
DK 2800 Lyngby, Denmark

INTRODUCTION

PERT networks in the education of electric power engineers is only a tool among a host of more important tools. Thus, the time allotted for the training of the students must be cut down to an absolute minimum. To introduce experience in an effective manner and to arouse the interest of the students a computer-aided PERT game seems to be the ideal educational approach. The purpose of this game is to give the students an introduction into the subject area so that later they can, on their own, use the more advanced literature and specialize in that subject. In this paper an educational tool will be described which, through its effective layout and direct appeal to the special knowledge already acquired by the students, promotes the wanted motivation and engagement.

Accordingly, the educational tool is presented in the form of a realistic project with aspects of competition. It was initiated by a Master Thesis[1] at the Technical University of Denmark in 1968 and has in a developed form been used in the past years as a part of the education. The administration of the game is completely documented in a User's Manual,[2] an Operator's Guide,[3] and the original thesis.

Headed by four lectures about the theories in modern network planning and control methods, the students plan and follow up a constructional problem or project with a time consumption of two periods of each four hours. The constructional problem or project presented in this paper does often occur in the electric power industry, and it is simulated on the IBM 1800 computer installed in the department.

Two major advantages are fulfilled by the tool:

It is designed to handle projects so complicated that it becomes impossible for the students to get a comprehensive view and therefore giving them a feeling of being in the world of reality. On the other hand, the project is made so simple that the instructor can control the learning process. The experience has shown us that projects, with a complexity corresponding to the use of about 70 activities, are relevant.

The computer program used by the tool is so designed that it allows a very high degree of flexibility. Any instructor with special industrial knowledge can develop a new data set for the program within 2-3 days and, thereby, interchange the whole project with an other of a similar realistic nature. Such an interchange does not call for special knowledge of programming.

WHY A GAME?

The purpose of the computer-aided PERT game is, through the planning and organization of a project, to train the participants in the possibilities of varying the construction time and project composition. At the same time as these possibilities there are constraints, expressing the causal relationship of the project and the resources available for its completion. They must be incorporated in order to obtain, within a fixed finishing date, the minimum constructional cost.

The aspect of competition is obtained by having up to 9 teams, each trying to build up a network in its optimal configuration, to prevent unforeseen occurrences, and to minimize the total cost. All this has to be done under the constraints to the time and resources. In addition to this, it is calculated, depending upon the total duration of the project, whether a penalty shall come into action or not.

The fact that we can accept it as a game is then involved in the following two parts:

1. The instructor can, directly by inspection of the total cost for each group, point out the winner.
2. The unforeseen occurrences happen for each group as a function of the delayed time with respect to the fixed finishing date. This is quite

```
I-----------------------------------------------------------------------------------------------------------I
I                                                                                                           I
I REPORT DATE   31/ 8/1971        C A P E R T S I M        FINAL   DECISION   PERIOD   2 SORT  KEY ..... SLACK I
I                                                                                                           I
I-----------------------------------------------------------------------------------------------------------I
I                                                                                                           I
I STARTING ENDING    ACTIVITY DESCRIPTION          ACTUAL     EXPECTED     LATEST      SCHEDULE              I
I                                                                                                           I
I EVENT    EVENT                                    DATE        DATE         DATE        DATE       SLACK    I
I                                                                                                           I
I-----------------------------------------------------------------------------------------------------------I
I                                                                                                           I
I   BB  EE   5  190 DAYS 150 DAYS 240 DAYS  92500 $   -100    6/ 9/1972   5/30/1972                -1.42    I
I                                                                                                           I
I   EE  FB  28    2 DAYS   2 DAYS   2 DAYS    300 $    000    6/11/1972   6/ 1/1972                -1.42    I
I                                                                                                           I
I   FB  BI   1    2 DAYS   2 DAYS   2 DAYS   1000 $    400    6/13/1972   6/ 3/1972                -1.42    I
I                                                                                                           I
I   GD  HC  27   13 DAYS  10 DAYS  19 DAYS   3000 $   -200    6/26/1972   6/16/1972                -1.42    I
I                                                                                                           I
I   BI  HC  48   13 DAYS  12 DAYS  19 DAYS   5200 $   -200    6/26/1972   6/16/1972                -1.42    I
I                                                                                                           I
I   HC  HD  58    4 DAYS   3 DAYS   4 DAYS   1500 $    500    6/30/1972   6/20/1972                -1.42    I
I                                                                                                           I
I   HD  HE  21    1 DAYS   1 DAYS   2 DAYS    400 $    400    7/ 1/1972   6/21/1972                -1.42    I
I                                                                                                           I
I   AE  HF  42  200 DAYS   1 DAYS 300 DAYS    000 $    000    7/ 4/1972   6/24/1972                -1.42    I
I                                                                                                           I
I   HE  HF  80    3 DAYS   3 DAYS   5 DAYS    700 $    200    7/ 4/1972   6/24/1972                -1.42    I
I                                                                                                           I
I   HF  HG  49    3 DAYS   3 DAYS   3 DAYS    300 $    000    7/ 7/1972   6/27/1972   06/27/1972   -1.42    I
I                                                                                                           I
I   GA  HC  87   12 DAYS   7 DAYS  12 DAYS  21000 $   -800    6/25/1972   6/16/1972                -1.28    I
I                                                                                                           I
I   BB  EG  98  188 DAYS 150 DAYS 240 DAYS  19200 $   -100    6/ 7/1972   5/30/1972                -1.14    I
I                                                                                                           I
I   EG  FB   4    2 DAYS   2 DAYS   2 DAYS    300 $    000    6/ 9/1972   6/ 1/1972                -1.14    I
I                                                                                                           I
I   AA  AH  63   15 DAYS  14 DAYS  15 DAYS    000 $    000    9/15/1971   9/12/1971                -0.42    I
I                                                                                                           I
I   FD  BI  97   31 DAYS  30 DAYS  31 DAYS   6800 $   -100    3/ 9/1972   6/ 3/1972                12.28    I
I                                                                                                           I
I   BB  BG  70    6 DAYS   6 DAYS   6 DAYS    000 $    000   12/ 7/1971   3/ 6/1972                12.71    I
I                                                                                                           I
I   BG  BH  20   25 DAYS  21 DAYS  25 DAYS  12500 $   -200    1/ 1/1972   3/31/1972                12.71    I
I                                                                                                           I
I   BH  BC   3   14 DAYS  14 DAYS  14 DAYS    000 $    000    1/15/1972   4/14/1972                12.71    I
I                                                                                                           I
I   BB  EC  32   90 DAYS  65 DAYS  90 DAYS  12000 $   -100    3/ 1/1972   5/30/1972                12.85    I
I                                                                                                           I
I   EC  FB  74    2 DAYS   2 DAYS   2 DAYS    300 $    000    3/ 3/1972   6/ 1/1972                12.85    I
I                                                                                                           I
I   AJ  FE  16  102 DAYS  90 DAYS 102 DAYS  10700 $   -100    2/ 9/1972   5/30/1972                15.71    I
I                                                                                                           I
I   FE  BI  25    4 DAYS   5 DAYS   5 DAYS   1000 $    000    2/13/1972   6/ 3/1972                15.71    I
I                                                                                                           I
I   BB  FH  51   56 DAYS  22 DAYS 222 DAYS   7200 $    000    1/26/1972   6/ 5/1972                18.57    I
I                                                                                                           I
I   BJ  FD  89    2 DAYS   2 DAYS   2 DAYS    000 $    000   12/ 5/1971   5/ 3/1972                21.28    I
I                                                                                                           I
I-----------------------------------------------------------------------------------------------------------I
I                                                                                                           I
I START.END.    ACTUEL LOWER TIME   UPPER ACTUEL  D(COST)/   EXPECTED     LATEST      SCHEDULE              I
I          CODE                                                                                             I
I EVENT EVENT     TIME  LIMIT   TIME LIMIT  COST   D(TIME)     DATE         DATE        DATE       SLACK    I
I                                                                                                           I
I-----------------------------------------------------------------------------------------------------------I
```

Figure 1—The starting situation of the project

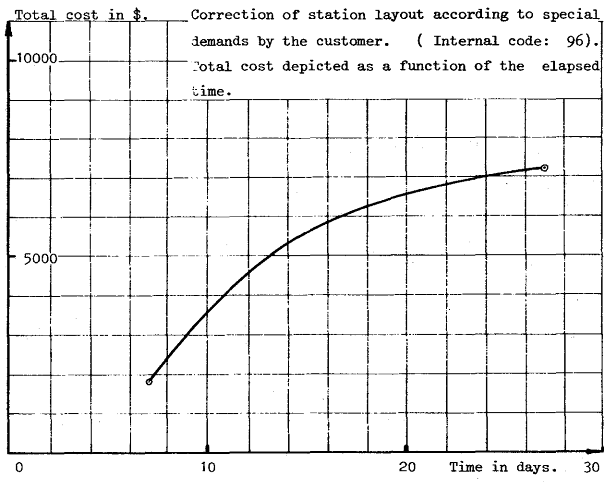
similar to the traditional aspect of throwing a die. But here we have weighted the die in such a manner, that the unforeseen occurrences depend on the size of the negative slack, i.e., upon the skill of the players.

HOW TO PLAY THE GAME

None of the participants need to have a special knowledge of programming. They are divided into groups each having a User's Manual containing the start information. This includes a description of the assumption for the start of the project (e.g., an approval to build the station from the board of preservation of natural beauty), a list of activities (e.g., fitting up 60 kV outdoor plant), an enlistment (Figure 1) regarding the starting situation of the project, where the activities as shown on the list are sorted after increasing slack. Furthermore, the cost data (Figure 2) which in matrices and graphs show the marginal cost of each activity as a function of time, is included. Finally, the rules for punching the data cards as well as a list of error messages are given.

The two periods of four hours fit well with the game. The first period is used to construct the network and to get it tested as indicated by the control routine as mentioned in a later section. The second period of four hours is used to carry out the decisions for the execution of the project, the consequences of which are simulated by the computer.

Two types of decisions are possible during the execution.


Correction of station layout according to special demands by the customer. (Internal code: 96). Total cost depicted as a function of the elapsed time.

Absolute cost matrix:

Days	7	9	11	13	15	17	19	21	23	25	27
Total cost	1800	3300	4500	5200	5700	6100	6400	6700	6900	7100	7200

Differential cost matrix:

To (days)		7	9	11	13	15	17	19	21	23	25	27
	7	0	1500	2700	3400	3900	4300	4600	4900	5100	5300	5400
F	9	-1500	0	1200	1900	2400	2800	3100	3400	3600	3800	3900
r	11	-2700	-1200	0	700	1200	1600	1900	2200	2400	2600	2700
o	13	-3400	-1900	-700	0	500	900	1200	1500	1700	1900	2000
m	15	-3900	-2400	-1200	-700	0	400	700	1000	1200	1400	1500
	17	-4300	-2800	-1600	-900	-400	0	300	600	800	1000	1100
d	19	-4600	-3100	-1900	-1200	-700	-300	0	300	500	700	800
a	21	-4900	-3400	-2200	-1500	-1000	-600	-300	0	200	400	500
y	23	-5100	-3600	-2400	-1700	-1200	-800	-500	-200	0	200	300
s	25	-5300	-3800	-2600	-1900	-1400	-1000	-700	-400	-200	0	100
	27	-5400	-3900	-2700	-2000	-1500	-1100	-800	-500	-300	-100	0

Figure 2—Cost data for one activity

A *trial* decision by which the participants can see the change in cost and the new slack for each activity as a result of suggested time alterations and length of the decision period. The project itself will remain in the old status.

A *final* decision by which the project is updated with the number of days given by the decision period. The status of the project is changed and there remains no possibility of changing the decision already made.

Each final decision, therefore, can be based upon an economical calculation involving the marginal cost for each activity carried out in a trial decision. A cost printout (Figure 3) will in both cases give the new status. No guess is necessary and the group can in fact plan and control the project.

Another feature of importance is the following. As described later, it is possible to change the network for the not finished part of the project during the game with the purpose of obtaining a more optimal network.

The computer program is equipped with test for input data and on the network, so that errors recorded can be corrected by the users without assistance from the instructor, just by making use of the corresponding error messages and the User's Manual. The topology cards and the time altering cards can optionally be introduced either over a terminal (1816 printer keyboard) or via the card reader.

The experience gained with the use of the game indicates that half a dozen final decisions are the educational optimal number.

A TYPICAL PROJECT

The project used for this presentation is a closed design project.[4] It is stated in terms of the engineering reality obtained by a simplification of the building of an open air transformer station for 60/10 kV on a turnkey contract with penalty for late delivery. Each element in the project is related to physical, realistic situations, because all data as well as unforeseen occurrences are selected from the pool of statistical information belonging to a big Danish company. The simplified project is large enough to give the students the impression that they are working with a genuine problem in the engineering design. The project is split up into 72 activities placed in an activity file.

Each item in the activity file contains start node, and termination node, code number, minimum, normal, and maximum duration, and upper and lower limits for the permissible duration caused by resource adjustment. Furthermore, in each of these items the most reasonable duration, two elements for identification of the cost function, and a verbal description of the activities are given.

```
I-------------------------------------------------------I-------------------------------------------------------I
I                                                       I                                                       I
I        GROUP   9   DECISION PERIOD     2              I                 FINAL      DECISION                   I
I                                                       I                                                       I
I-----------------I-----------------I-----------------I-----------------I-----------------I-----------------I
I                 I                 I                 I                 I                 I                 I
I    ACTIVITY     I    PREVIOUS     I    PREVIOUS     I   NEW    TIME   I   CHANGE   IN   I    NEW   COST   I
I                 I                 I                 I                 I                 I                 I
I     EVENTS      I      TIME       I      COST       I                 I      COST       I                 I
I                 I                 I                 I                 I                 I                 I
I-----------------I-----------------I-----------------I-----------------I-----------------I-----------------I
I                 I                 I                 I                 I                 I                 I
I      BBEE       I     205DAYS     I      91200 $    I     190DAYS     I       1300 $    I      92500 $    I
I                 I                 I                 I                 I                 I                 I
I      BBEF       I      91DAYS     I      20200 $    I     105DAYS     I      -1700 $    I      18500 $    I
I                 I                 I                 I                 I                 I                 I
I      BBEC       I      75DAYS     I      13800 $    I      90DAYS     I      -1800 $    I      12000 $    I
I                 I                 I                 I                 I                 I                 I
I      GAHC       I       9DAYS     I      24600 $    I      12DAYS     I      -3600 $    I      21000 $    I
I                 I                 I                 I                 I                 I                 I
I      HBHC       I      30DAYS     I      11800 $    I      37DAYS     I      -1500 $    I      10300 $    I
I                 I                 I                 I                 I                 I                 I
I      BCHC       I      50DAYS     I      41600 $    I      63DAYS     I      -4700 $    I      36900 $    I
I                 I                 I                 I                 I                 I                 I
I      BBBC       I      55DAYS     I     124400 $    I      60DAYS     I      -4900 $    I     119500 $    I
I                 I                 I                 I                 I                 I                 I
I      AJCF       I     121DAYS     I      14000 $    I     140DAYS     I      -1100 $    I      12900 $    I
I                 I                 I                 I                 I                 I                 I
I      AJCD       I     120DAYS     I      21400 $    I     150DAYS     I      -1100 $    I      20300 $    I
I                 I                 I                 I                 I                 I                 I
I      AJFF       I     165DAYS     I      13200 $    I     190DAYS     I      -1200 $    I      12000 $    I
I                 I                 I                 I                 I                 I                 I
I-----------------I-----------------I-----------------I-----------------I-----------------I-----------------I
I                                                       I                                                       I
I      TOTAL  CHANGE   IN   COST   IS  -20300 $         I     PREVIOUS  TOTAL    COST     WAS   915400 $        I
I                                                       I                                                       I
I---------------------------I---------------------------I---------------------------I---------------------------I
                            I                                                       I
                            I         NEW TOTAL     COST       IS  895100 $         I
                            I                                                       I
                            I-------------------------------------------------------I
```

Figure 3—Cost print-out according to final decision

A test file with one item per activity is used for the investigation of logical and correct build-up of the network. Each item in this test file contains the code numbers of all direct predecessors in the optimal network.

A text file, explaining the cause and effect of the unforeseen occurrences, is given.

The total duration, optimal cost, and the penalty fines, are specially codified.

Total exchange of data for a project with 72 activities and 22 unforeseen occurrences necessitates only the punching of $72+72+22\times6+1=277$ cards in order to establish the required data files. Thus, the earlier mentioned great flexibility is substituting one project with another.

CONSTRAINTS ON THE COMPUTER PROGRAM

The basic idea behind the construction of the computer program was that, by using networks with a number of activities smaller than 100, the requirement of available core storage is satisfied by almost any computer configuration. Furthermore, the computing technique used is kept at a non-sophisticated level in order to make it possible for any programmer to easily make changes. Of course, this does increase the computer time a little but, from the scope of the educational tool, this is of no importance at all.

The program, therefore, fulfills the following specifications:

a. Information, giving exact description of the specified project is placed on easily exchangeable data files on a disk.
b. The partners in the group must individually build up the relevant network from the given list of activities. Before the project is simulated on the computer, an analysis and control test are performed of the proposed network with regard to a logical and correct setup.
c. During the simulation of the project, the length of the decision intervals or the interval to the next date of followup, must be estimated by the user. This flexibility is introduced in order to make it possible for him to adjust his decision-making to the complexity of the subnetwork considered.
d. Delays, caused by unforeseen occurrences, are incorporated in the program as a function of the skill of the participants in order to simulate reality.
e. It is possible to introduce alterations in the network after each follow-up of the network. Obviously, such alterations will influence the remaining

period of the project. It should be added, that any alteration of this kind automatically results in a new test for a logical and correct setup.

THE STRUCTURE OF THE COMPUTER PROGRAM

A control system on a higher level establishes the necessary files and interconnects the eighteen subroutines by means of a conventional overlay structure.

The subroutines are grouped according to their various functions as follows:

a. A file part establishing the interconnection between the various teams and their corresponding data areas.
b. A control part reading the so-called topology cards, that is the cards describing the structure of the network. By means of these cards the program checks the network in order to evade loops, and multiple start nodes, and terminating nodes. Also, a node file and an arc file are formed. Finally, the network is checked to see that it is logically consistent and physically correct.
c. A cost part which, after registration of the time alteration cards, calculates the economical consequences of such changes.
d. An occurrence part calculates, based upon the node file and arc file, if those activities where the unforeseen occurrences may take place are finished within the decision period in question. Whether unforeseen occurrences actually shall take place, is decided by a so-called time criterion. This time criterion is based upon the idea that teams, who by bad planning have obtained too great negative slacks, should be punished. This part also includes the print routine for description of the unforeseen occurrences.
e. A time calculation part, which for each activity determines latest time of starting and earliest time of finishing. Furthermore, in this part a sorting according to slack and printing after increasing slack is carried out. In case of unfinished activities the list of sorted activities includes, in addition to start and finishing node markings, the following information: the actual time; the minimum and maximum times; the actual price; the differential price change (by 1 day's extension of the activity); the expected date; the latest date and the planned date; and the slack. For finished activities this list includes only, in addition to start and finishing node marking, the verbal description and the actual time for the termination.
f. A follow-up part that stores all relevant data for the respective teams and prepares a graphical description for comparison of the status of the teams, i.e., a time-cost graph. This can be plotted on request.

The computer program is coded in IBM 1130/1800 Basic FORTRAN IV. The utilized over-lay structure is arranged to keep the number of disk operations in combination with the handling of the files at a minimum. The subroutines are collected in main groups corresponding to full utilization of the core storage available. The present upper limit for the complexity of the project is set by the size of the data areas. Currently they are designed to handle up to 100 activities, and a maximum number of parallel paths not exceeding 400. Experience has shown that these limits are insignificant from an educational viewpoint. Comparing this computer program with commercial systems, it must be remembered that the above-mentioned parts: a, b (partly), d, and f (partly) are established primarily for educational purposes. That is, these parts are of no use in practical PERT planning programs. The turn-around time to verify optimal network is 5.5 min. and the time for an updating maximum 7.8 min. For non-optimal networks the turn-around time is considerably shorter.

CONCLUSION

In the past many brilliant professional PERT educational systems have been developed. But most of these, operating on a theoretical level, are highly generalized and seem to neglect to give the students an introduction to PERT and its applications except from the narrow viewpoint of purely mathematical algorithms. The acual planning and control of a given project must also be perceived as an integral part of PERT as a design tool. In the education one must incorporate heuristic elements in order to establish the vital link between mathematical models and reality. The computer-aided PERT game described in this paper is different. It handles realistic projects and takes into account unpredictable changes from the reality. Thus, it emphasizes that, although various aspects of engineering design may be reduced to formal disciplines, design will remain an art which can be mastered only by actual practice.

REFERENCES

1 J A RICHTER-NIELSEN
CAPERTSIM-1800

Master Thesis 1969 Electric Power Engineering Department Technical University of Denmark

2 J A RICHTER-NIELSEN
CAPERTSIM-1800 User's manual
Electric Power Engineering Department Technical University of Denmark 1969

3 J A RICHTER-NIELSEN
CAPERTSIM-1800 Operator's guide
Electric Power Engineering Department Technical University of Denmark 1970

4 O I FRANKSEN
Closed and open design projects in the education of engineers
IEEE Transactions on Power Apparatus and Systems No 3 March 1965

Interactive problem-solving—An experimental study of "lockout" effects

by B. W. BOEHM, M. J. SEVEN, and R. A. WATSON

The RAND Corporation
Santa Monica, California

THE NEED FOR QUANTITATIVE MAN-COMPUTER DATA

One danger inherent in computer system design and management is an ever-present temptation to consider computer system performance as an end in itself, rather than as a means to better serve people. Such "performance improvement" methods as universal use of one language, large blocking of data input and output, and intricately designed code and procedures can increase machine productivity. However, it costs users an abnormally high effort to achieve any results. On the other hand, text editors, extended debugging aids, and conversational programming systems tend to reduce user-time investments at the expense of machine efficiency.

In general, then, there is a tradeoff between machine efficiency and user time invested. Philosophical arguments will yield to factual analysis of this tradeoff only when the effects on both the humans and machines can be quantitatively measured and related to overall goals. In an attempt to contribute to the currently scant store of quantitative information on man-computer problem-solving processes,† and to evaluate available experimental techniques in the area, we designed and implemented an exploratory controlled experiment in man-computer problemsolving.‡

CHOICE OF EXPERIMENT

We structured this experiment to test Gold's hypothesis[2] that restricting one's access to the computer for a period of time *after* the presentation of current results ("lockout" period), might improve performance by inducing the user to concentrate more on problem-solving strategy than on tactics.

Figure 1 shows that the lockout requires the user to spend a certain amount of what is generally called "think time."

The general problem-solving situation required the subject to solve a geographical area servicing problem with the aid of JOSS, RAND'S interactive computer system.[3] Subjects were allowed two hours to solve the problem, but the problem was open-ended to the extent that a range of solutions existed. A protocol of each subject's performance was generated from automatic recordings within the JOSS system, written records kept by an observer, and audio tape recordings of the subject's vocalizations. The resulting data were analyzed using analysis of variance and regression techniques.

THE TEST PROBLEM

Each subject was given a map showing a grid of surface streets, two freeways, and contour lines that indicated the frequencies of emergencies per day per intersection throughout the area (Figure 2). Transit times between intersections were defined as two minutes on North-South surface streets, three minutes on East-West surface streets, and one minute on freeways. A time penalty of one minute was assessed for entering or leaving the freeway at any intersection.

The subject's task was to specify three surface intersections at which to locate three emergency hospitals, and to specify a set of decision rules regarding when and when not to use the freeways. His goal was to minimize the average response time per emergency for the entire area, taking into account the different accident densities. His solution was subject to the constraint that the maximum one-way response time to

* RAND consultant, Professor of Psychology, Harvey Mudd College.

† Sackman[1] has provided an excellent review of results to date.

‡ The Appendix describes some of the rationale leading to the structure of this experiment, beginning with an attempt to define a reasonably measurable and human-oriented computer system performance criterion called the "Productive Thought Ratio."

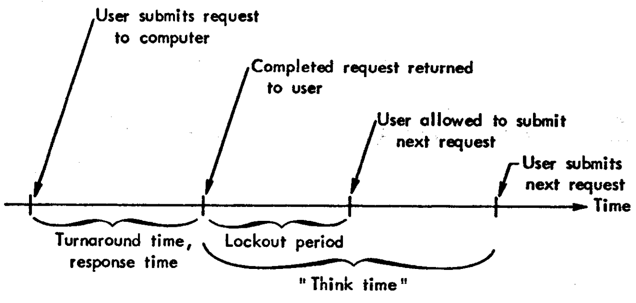

User submits request to computer
Completed request returned to user
User allowed to submit next request
User submits next request
Time
Turnaround time, response time
Lockout period
"Think time"

Figure 1—Sequence of events for submitting a trial solution

any given location be no more than 12 minutes. It was made clear that the number of ambulances was unlimited; scheduling and ambulance turn-around time were not factors.

The JOSS system was pre-programmed to provide the subject, on demand, with an evaluation of the effectiveness of his location and decision-rule inputs, and with certain other feedback relating to the problem. Hospital locations were specified in X, Y coordinates shown on the map. Variables used in the decision rule were specified so that the subject could refer to specific hospitals ($i = 1$, 2 or 3), hospital locations (x, y), or emergency locations (v, w) in terms understood by the special program. As a result of an evaluation computation ("Do part 1."), the program provided (1) the average response time per emergency, and (2) the maximum response time to any emergency. If requested, the program also provided various types of information matrices:

1. A matrix showing minimum response time to each intersection from any of the three hospitals ("Do part 210.");
2. Three individual matrices showing response times to each intersection from each of the three hospitals ("Do part 220.");
3. An individual matrix showing response times from each hospital specified ("Do part 221 for $i =$ —."). The use of the special program was illustrated with a reproduction of an actual JOSS record of three "trials" (Figure 3).

TREATMENT GROUPS

The primary experimental treatment was provided by programming the JOSS system to lock the subject out of the system for a specified length of time after each

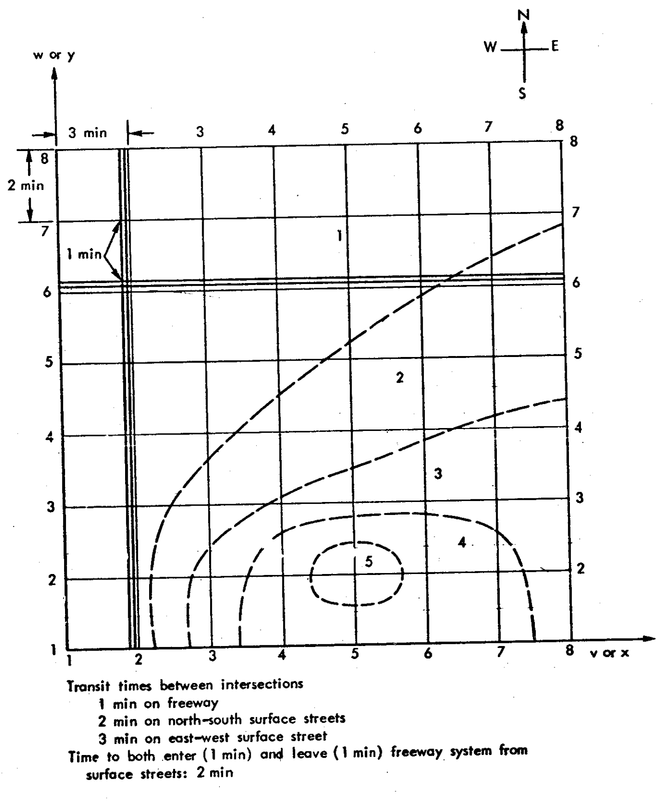
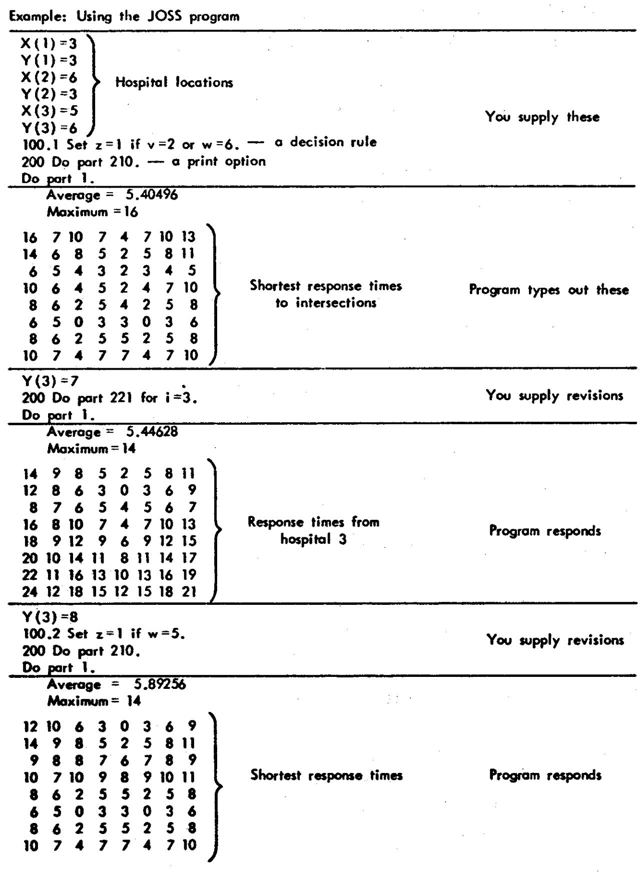

N
W
E
S
w or y
v or x
3 min
2 min
1 min
1
2
3
4
5
6
7
8
1
2
3
4
5
Transit times between intersections
1 min on freeway
2 min on north-south surface streets
3 min on east-west surface street
Time to both enter (1 min) and leave (1 min) freeway system from surface streets: 2 min

Figure 2—Problem map, on-line experiment



```
Example: Using the JOSS program
X(1)=3 ⎫
Y(1)=3 ⎪
X(2)=6 ⎬ Hospital locations
Y(2)=3 ⎪
X(3)=5 ⎪                                              You supply these
Y(3)=6 ⎭
100.1 Set z=1 if v=2 or w=6. — a decision rule
200 Do part 210. — a print option
Do part 1.
   Average =  5.40496
   Maximum = 16

16  7 10  7  4  7 10 13
14  6  8  5  2  5  8 11
 6  5  4  3  2  3  4  5
10  6  4  5  2  4  7 10    Shortest response times    Program types out these
 8  6  2  5  4  2  5  8      to intersections
 6  5  0  3  3  0  3  6
 8  6  2  5  5  2  5  8
10  7  4  7  7  4  7 10
Y(3)=7
200 Do part 221 for i=3.                              You supply revisions
Do part 1.
   Average =  5.44628
   Maximum = 14

14  9  8  5  2  5  8 11
12  8  6  3  0  3  6  9
 8  7  6  5  4  5  6  7
16  8 10  7  4  7 10 13    Response times from        Program responds
18  9 12  9  6  9 12 15      hospital 3
20 10 14 11  8 11 14 17
22 11 16 13 10 13 16 19
24 12 18 15 12 15 18 21
Y(3)=8
100.2 Set z=1 if w=5.                                 You supply revisions
200 Do part 210.
Do part 1.
   Average =  5.89256
   Maximum = 14

12 10  6  3  0  3  6  9
14  9  8  5  2  5  8 11
 9  8  8  7  6  7  8  9
10  7 10  9  8  9 10 11    Shortest response times    Program responds
 8  6  2  5  5  2  5  8
 6  5  0  3  3  0  3  6
 8  6  2  5  5  2  5  8
10  7  4  7  7  4  7 10
```

Figure 3—Sample JOSS printout

trial, i.e., after a current set of results had been presented to him. Lockout conditions were different for each of five groups of subjects, and included both fixed and variable intervals.

The subjects, primarily graduate students at RAND for the summer, were divided into the following five groups:

0—No lockout; free access to console;
5—Five-minute lockout period;
8—Eight-minute lockout period;
V—Variable lockout period (5-min mean);
c—Choice; subjects were instructed to "lock themselves out" as much as possible, but otherwise had free access to console.

On the basis of a questionnaire, subjects were ranked from 1 to 20 with respect to computing and operations research experience. The experimental groups were balanced in regard to experience.

BASIC RESULTS

The primary measure of a subject's performance was the minimum average emergency response time he could achieve during the two-hour period. For analysis purposes, this performance was transformed into a percentage of optimal performance. Figure 4 shows the resulting performance of each individual, organized with respect to experimental group and presented with group averages. In this case, the group with a moderate lockout period (5 min) performed better than both the group with free access (0 min) and the group with a relatively severe lockout period (8 min). The variable lockout group performed almost as well as Group 5, and the "choice" group almost as well as Group 0. (However, Group C achieved this performance with half as many computer trials as Group 0.)

The numbers next to each data point in Figure 4 indicate the subject's experience ranking, "1" being highest. It is evident that performance strongly correlates with experience. This comparison is highlighted in Figure 5, which plots each subject's performance rank versus his experience rank. The associated symbol identifies the subject's group. Most subjects fall quite close to the equivalence line bisecting the figure. However, the less experienced members of Groups 5 and V generally performed better than their experience might predict. The more experienced members of Group 8 and the less experienced members of Groups 0 and C generally performed worse than their experience might predict. Analysis of variance calculations indicates that lockout is significant at the 0.025 level, experience significant at the 0.005 level, and the interaction between lockout and experience significant at the 0.10 level.

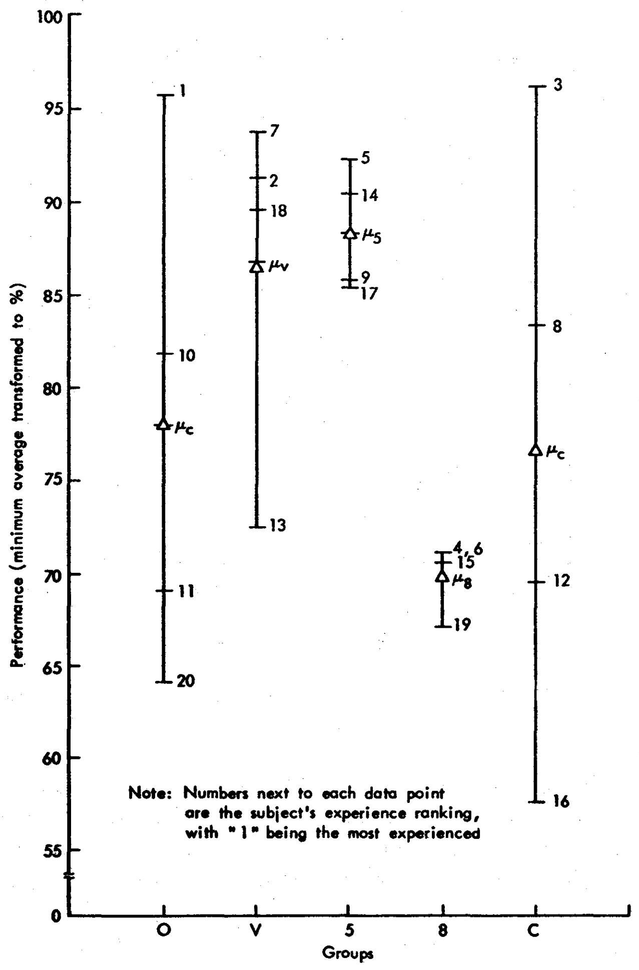


Figure 4—Quality-of-solution scores attained by the subjects

Over 40 other performance measures were collected and analyzed along with considerable anecdotal data of interest, detailed in Reference 4. Significant further results demonstrate that:

1. The subjects with free access (Group 0) average twice as much computer usage as groups with restricted access.
2. Group 0 subjects show no relative economies of their own time in attaining their high performance levels; however, Group 5 subjects do.
3. In general, subjects express dissatisfaction with restricted access, even in the groups with high performance.

TENTATIVE CONCLUSIONS

Perhaps the most impressive aspect of the experiment was the subjects' tremendous variability in problem-

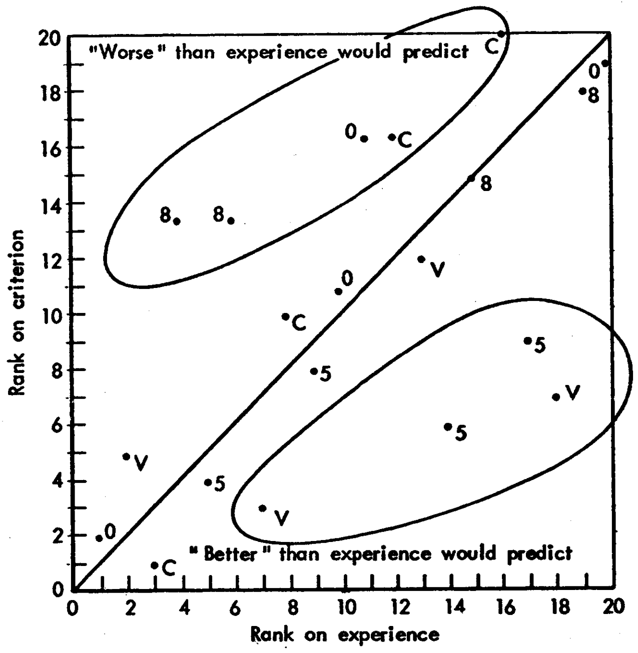


Figure 5—Rank on experience compared with rank on criterion measure

solving approaches. It is difficult to imagine anyone ever formulating a single model of man-computer problem solving that would fit even our small group of subjects, which included some whose performances were so irregular that they had to be dropped from the analysis. For example, one subject promptly began by dumping our JOSS control program; after two hours, his only result was a set of undebugged modifications of this program. Another preferred to work almost completely by hand, saying "he didn't trust computers," and never achieved a feasible solution.

However, with respect to the large majority of problem solvers who achieved feasible solutions in the experiment, the results of this small exploratory study raise some interesting questions regarding popular beliefs about man-machine problem solving. Our evidence suggests that, at least in this experimental context, users tend to become dissatisfied if mild restraint is placed on their free interaction with the computer. They also tend to problem solve more effectively, using less computer time and less of their own time in the process. Such shibboleths as "faster is better" and "more computer time means less human time" may at times serve the computer salesman more than the consumer. The results also cast doubt on the validity of user acceptance as a general index of system effectiveness. The user may want what inconveniences him least in the short run, or he may want what he has been led to believe he should want, but the general efficacy of such desires cannot be taken for granted.

Definitive answers to questions relating to the nature of relevant parameters of problem solving systems are of more than academic value. For example, under some circumstances, organizations under pressure to expand their hardware inventory to meet increased demand might find it far more productive to keep the system they have and introduce some form of constraint (e.g., an accounting system) that will encourage more judicious and creative use of the existing computational capabilities. However, without more information and better understanding, it would be a mistake to conclude that either approach is the "right" one. The only general conclusions that can be reached on the basis of the present work are that the relationships involved in man-machine problem solving are neither obvious nor simple, and that there is reason to believe that further investigation could have practical significance.

MAN-COMPUTER EXPERIMENTS—FUTURE PLANS

We are currently testing the same problem on another group of subjects, under lockout conditions 0 and 5, to determine whether the initial results are confirmed by a larger sample. Also, because earlier subjects indicated in their debriefings that a graphic display capability could have helped them, and since RAND has an interactive Conversational Programming System (CPS) working on both typewriter and graphic consoles, we are considering a modification of the current experiment to test the relative efficacy of typewriter and graphic terminals in this context.

However, before plunging into another experiment we feel it important to devote more thought to two fundamental questions:

1. The classification of problem characteristics and problem solving activities, at least in the neighborhood of our current study.
2. Determination of better measures of human problem solving experience, attitudes, and capability.

Without solid foundations in these areas, future studies will progress no further in operational utility than the one reported here: provocative, useful as a cautionary indicator, but hardly a predictor for any operational situation.

REFERENCES

1 H SACKMAN
Experimental investigation of user performance in time-shared computing systems: Retrospect, prospect, and the public interest
System Development Corporation SP-2846 1967

2 M M GOLD
Methodology for evaluating time-shared computer usage
PhD dissertation Massachusetts Institute of Technology 1967
3 C L BAKER
JOSS: Introduction to a helpful assistant
The RAND Corporation RM-5058-PR August 1966
4 M J SEVEN B W BOEHM R A WATSON
Problem-solving with an interactive computer: A study of user behavior
The RAND Corporation R-513-NASA In process

APPENDIX—TOWARD A PERFORMANCE CRITERION FOR MAN-COMPUTER SYSTEMS: THE PRODUCTIVE THOUGHT RATIO

The following approach guided our research efforts in the analysis of man-computer systems:

1. Formulate a performance criterion for man-computer systems that appears reasonably discriminating and measurable.
2. Investigate the implications of using this criterion operationally.
3. Identify the resulting key problems and experiment for insight into them.

FORMULATE A PERFORMANCE CRITERION

Current performance criteria for such computer systems as throughput, component utilization efficiency, and turnaround or response time, tend to concentrate on the servicing of individual computer run requests rather than on the project advancement for which a given run is being made. Computer systems optimized with respect to the above criteria tend to emphasize machine efficiency at the expense of such amenities as ease of learning, programming, debugging, or modifying programs, which tend to increase human efficiency.

Suppose, however, that one could characterize the computer support of various types of projects (e.g., an engineering research and development project) as time series of individual computer run requests (Figure 6), and that one could separate the time spent on the project into three activities, essentially mutually exclusive:

T_1: Time spent thinking about the project;
T_2: Time spent thinking about the programs supporting the project;
T_3: Time spent waiting for the computer to respond.†

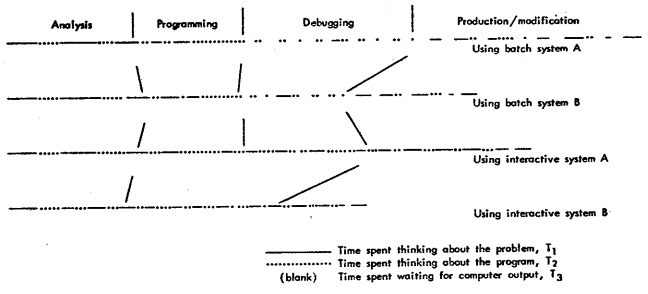

Analysis
Programming
Debugging
Production/modification
Using batch system A
Using batch system B
Using interactive system A
Using interactive system B
Time spent thinking about the problem, T_1
Time spent thinking about the program, T_2
(blank) Time spent waiting for computer output, T_3

Figure 6—Time series characterization of a computer-supported project

Then consider the following performance criterion, the *productive thought ratio* (P.T.R.):

$$\text{P.T.R.} = \frac{T_1}{T_1 + T_2 + T_3}.$$

A computing system that maximizes the P.T.R. (over some mix of projects) will not only try to increase machine efficiency (by decreasing T_3), but also human efficiency by decreasing T_2 through reducing the time spent learning languages, programming, debugging, and modifying programs).‡ Also, with the time series characterization of a project, the P.T.R. is a reasonably *measurable* quantity; it requires the currently available machine measurements plus an approximate breakdown of how people use their time.

INVESTIGATE OPERATIONAL IMPLICATIONS; IDENTIFY KEY PROBLEMS

Suppose the manager of a computer system received the following statement:

"This month our P.T.R. was 0.37. Last month it was 0.29."

What would this statement tell him?

1. *Nothing, unless he was sure the variation was not attributable to changes in workload.* To eliminate this difficulty, some means must be found to

† Such a characterization is appropriate for a computer service facility and the maintenance aspects of a real-time control system. It is less appropriate for the operating aspects of a real-time control system, which are better judged directly with respect to the objectives of the system.
‡ The P.T.R. is intended to function best in evaluating changes relative to an existing operation. Its current form does not exclude extreme cases that produce unwarrantedly good results. For example, not using a computer yields $T_2 = T_3 = 0$ and P.T.R. $=1$, an "optimal" solution. These could be fixed by adding more terms, but this would obscure the subsequent discussion of more fundamental difficulties.

normalize the P.T.R. with respect to workload. One possibility would be to measure it only with respect to standard project types (large event simulations, small scientific investigations, multitape data analyses, etc.), under the assumption that the projects within each type are relatively homogeneous. At present, no solid data are available to test this assumption, which suggests one potential research area: the collection of detailed case histories of several projects within one of the above standard project types, and their characterization and comparison in terms of such schemes as Figure 6.

2. *Very little, without some correlation between thinking time and insight.* This correlation can vary markedly for different systems, particularly in such areas as computer graphics.† However, some properly instrumented experiments in man-computer problemsolving could shed some light on the question. This became one of our experimental design considerations.
3. *Nothing, if based on bad measurements.* How capable and how motivated are people to separate their time accurately into categories T_1, T_2, and T_3? Our experimental observations and debriefing forms were structured to pick up such information. As one example of the results, we found that some of the subjects' estimates of time spent waiting for the computer to respond (T_3) were underestimated or overestimated by a factor of at least two, though on the average the agreement with observations was fairly close.
4. *Very little, if a significant number of users are productively using their computer wait time (T_3) on one project to advance another project (time-sharing themselves).* This avenue leads to a host of fundamental questions involving human thinking and problem solving processes, generally couched in such elusive terms as "concentration," "subconscious," "motivation," etc. At this early stage the most definitive statements possible are:
   - The P.T.R. is not a sufficiently delicate metric to illuminate this phenomenon, and at best its use must be restricted to "dedicated" activities.
   - To make any headway with the fundamental questions above, one needs a larger problem-solving data base. Our increased appreciation of this need strengthened the case for performing experiments to gather more data.
   - Such phenomena as "lockout" cannot be neatly fitted into the categories T_1, T_2, and T_3; our considerations of the possible effects of lockout led to the major hypotheses to be tested via controlled experiment.
5. *Probably about as much as can any other general man-computer performance criterion at this time.* The significance of the above discussion lies in its general applicability and most of these enumerated P.T.R. difficulties arise with alternative criteria that attempt to assess the computer's contribution to human performance over a wide spectrum of activities. As the spectrum of activities is narrowed, ways can be found around some of the difficulties (e.g., workload variation), but others will remain quite thorny (e.g., accuracy of measurements and value of results).† However, the potential payoffs of even partial insights in this area are sufficient to warrant an increased level of activity in gathering and analyzing man-computer performance data.

† In practice, of course, measures of effectiveness must also be balanced with measures of cost.

† Some care must also be taken to avoid criteria that over-emphasize the machine-like aspects of human performance (e.g., number of designs tried in a day).

TYMNET*—A terminal oriented communication network

by LA ROY TYMES

Tymshare, Inc.
Cupertino, California

INTRODUCTION

The past few years have seen many applications of the mini-computer in digital communications. They have been used as interfaces to larger computers. They have been used as terminal drivers and data multiplexors.[1] They have been used to connect computer centers for intercomputer communication.[2,3] TYMNET* is a communication net that encompasses all of these features at relatively low cost.

Although the net is very general purpose, it has been specifically oriented around the needs of the full duplex terminal in the ten to thirty character per second range. The terminal, connected to an acoustic coupler, interacts with a program in a timeshared computer on a character-by-character basis. The goal is to make this interaction as intimate as possible, even though the terminal and program are often thousands of miles apart.

THE HARDWARE

The net consists of Varian 620i's interconnected by 2400 and 4800 bit per second synchronous full duplex private leased lines. The 620i is a 16 bit machine with three working registers and a 1.8 microsecond cycle time. They are standard, off the shelf machines with 8K memory. Their only options are power failsafe and a primitive interrupt. The clocks from the synchronous modems interrupt the 620 for every bit to and from the synchronous modems. Some machines have a synchronous modem interface designed and built by Tymshare to assemble and disassemble 16 bit words, thus reducing the number of interrupts by a factor of 16.

The 620's are divided into two types, called base and remote (see Figure 1). The purpose of the base is to interface the net to the host, or timeshared computer. An assumption, which has proved to be valid, is that the base is usually up even if the host is down. Therefore, the base is useful as a node in the net even when it is not serving its host.

The remote drives the terminals. The typical remote is connected to up to 31 asynchronous full duplex modems. The data rate varies from 110 to 300 baud. Some remotes output to hard wired terminals at 600 and 1200 baud.

The remote can output on two wires and input on three wires on each modem for passing data and control information. On outputting a 16 bit word, the remote sets the voltage on 16 wires. On inputting a 16 bit word, the remote reads the voltages on 16 wires. The hardware understands the relationship between zeroes, ones, and RS232 voltage levels, and nothing more. In particular, it knows nothing about baud rate, character rate, or any other terminal characteristic. Serialization and deserialization of characters is done entirely with a software routine which is executed 1200 times a second. This simple hardware gives maximum reliability and flexibility at minimum cost.

When a user calls in to our system, he types a character to identify his terminal type. The remote analyzes this character to determine the baud rate, character rate, carriage return delay time, and other parameters. It then assigns two routines to this port, one for input and one for output (some terminals use a slower character rate for the keyboard than the printer). These routines handle all idiosyncracies of the terminal, including conversion to and from ASCII if the terminal is non-ASCII. Thus it is an easy matter to accommodate new terminal types as they become available.

If the remote has more than 8K of memory it may also have a printer, magnetic tape unit, and other equipment attached to it. The hardware interface is always the most primitive that will work.

* Trademark, Tymshare, Inc.

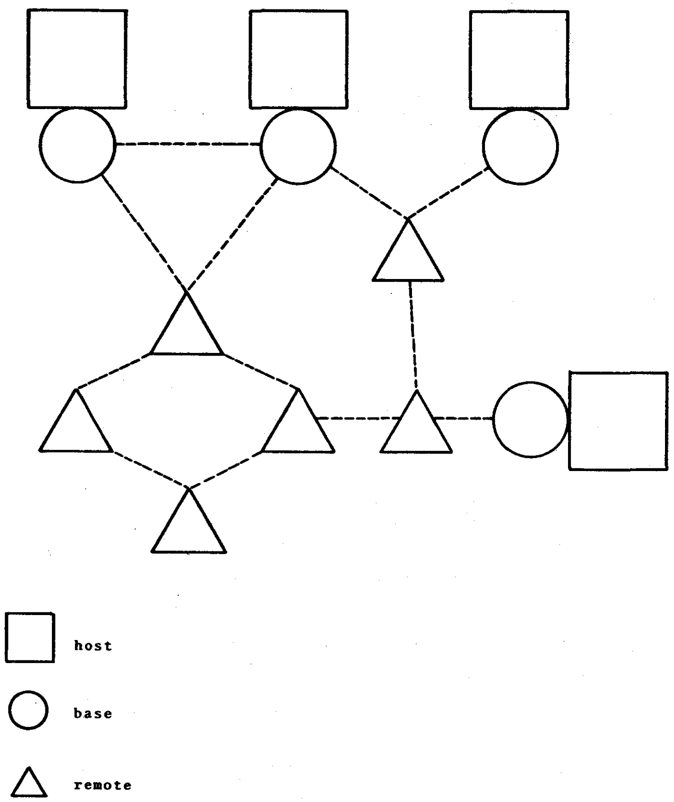


Figure 1—A small network showing the relationships between host, base, and remote

THE CIRCUIT

The nodes of the net are interconnected by full duplex synchronous private leased lines at 2400 and 4800 bits per second. The data is sent over these lines in the form of physical records. A physical record consists of a 16 bit header followed by several logical records followed by 32 bits of checksum. Errors are corrected by retransmission. When the error rate is nominal (less than 10 incorrect checksums per minute) about 240 data characters per second are passed on a 2400 bit per second line, or enough to handle 40 interactive terminals.

Inside each node there are a large number of character buffers, each one assigned to some routine which processes characters found in that buffer. Placing a character into a buffer is sufficient to insure that the appropriate routine will process that character at the appropriate time. In particular, some buffers are assigned to the physical record maker and placing a character into such a buffer will cause it to be assembled into a physical record with a particular virtual channel number.

Associated with each synchronous line is a permuter table (see Figure 2). This table is a list of pointers to pairs of buffers. The nth entry in the table corresponds to virtual channel n for that line. There is a matching permuter table in the node at the other end of that line.

We are now ready to trace the path of a character through the net. When the user strikes a key on a terminal, the terminal serializes the character and feeds it into an acoustic coupler. The coupler sends it over a telephone line to an asynchronous modem. The remote assembles the character and places it into the buffer assigned to virtual channel 2 and informs the physical record maker that something is ready for channel 2. The physical record maker looks up the second entry in the permuter table to find which buffer has the character. It then builds the record (possibly containing data for other channels as well). When this record arrives in the next node (the base in this example) it is torn down. Since the character came in on channel 2, the second entry in the permuter table for this line is checked to see where to put the character. In this case it goes into port 1 of the host.

I am now ready to define a circuit. A circuit is a full duplex character path through the net. It is described by entries in permuter tables. There are two buffers in each node per circuit, one buffer for each direction of character flow. Some readers will recognize this as a permutation switching network.[5, 6]

A word about the efficiency is in order. Each logical record begins with one character to specify size and another character to specify virtual channel number. If data characters are sent in to the host one at a time as rapidly as a person types, there are two characters of overhead for every data character, plus the header and checksums of the physical record.

Several observations are appropriate here. First, the remote normally handles echoing on full duplex terminals. Second, while there are hundreds of terminals in use at any given instant, there are only 15 or 20 host computers clustered into three computer centers. Third, although the capacity of the synchronous lines is symmetric, the data flow into the host com-

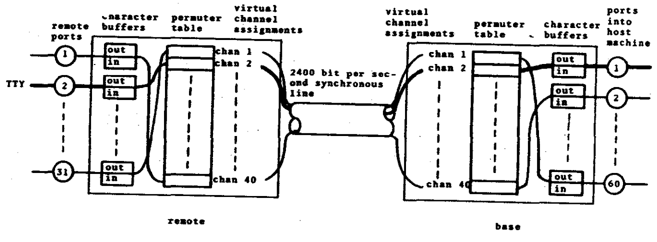


Figure 2—A circuit passing through two nodes between terminal and host. Two buffers are used in each node, one buffer for each direction of character flow. In this example, a teletype on remote port 2 uses virtual channel 2 to reach host port 1

puters is only a small fraction of the outbound data flow. Fourth, computer programs do not usually output characters one at a time: the mean is about 40. Thus, on the inbound stream, we are "inefficient", but it does not matter. On the outbound stream the overhead (which includes rate control and network control, see below) is about 20%.

Another problem is how does one control the rate at which characters flow. A computer program can easily output at 1000 characters per second, whereas a terminal may consume them at only 10 per second. The buffering capacity of the intermediate nodes is finite and will be exhausted unless backpressure can be generated to shut off the program.

A circuit can be low speed, meaning it can handle terminals up to 30 characters per second, or high speed, for 120 character per second links. Most circuits are low speed because they require less buffering. High speed circuits use high speed virtual channels, and low speed circuits use low speed channels. The distinction is noted by a bit in the permutation table.

Whenever node A sends characters to node B on channel *n*, it subtracts the number of characters sent from a counter associated with channel *n*. When that counter goes to zero, no more characters will be sent on that channel. Twice a second node B will send one bit per channel back to node A saying whether it has less than 32 characters (128 on a high speed channel) in the receiving buffer for that channel. If node B has less than 32 characters, node A will reset its counter to 32 (128 for high speed channels). This tends to put a vague upper bound on the number of characters a node will buffer for any given circuit. Thus, backpressure is cascaded back to the source.

The circuit may be regarded as a pair of pipes connecting two points. In each pipe the characters always flow in one direction, and they never get out of order. All interaction between terminal and program is through these pipes. All entities in these pipes are eight bit bytes, but they are not all data characters.

There are several special control characters to allow the program to control echoing modes and other properties of the circuit. All are encoded as eight bit bytes. Since a data character may be any eight bit byte, it could look like a control character. If it does, it is preceded in the pipes by an escape character which warns all appropriate machinery that the succeeding character is a data character, whichever it may look like.

Another special character is used to precede, or prefix, an eight bit parameter modifier. The parameter modifier is used to allow a user program to change one of 16 four bit fields describing the terminal in the remote. This allows the user program to alter such terminal characteristics as input baud rate, output baud rate, and the parameters in the formula to compute carriage return delay time. It can also control such things as whether an incoming line feed is echoed as a line feed or (more commonly) as a line feed, carriage return, and rubout.

The remote normally handles all echoing for full duplex terminals, but there are times when this is undesirable or impossible. For instance, if the terminal is listing output from the program and the user types ahead, the remote cannot echo the characters or they would be mixed up with the output text. (Everyone types ahead on full duplex terminals. One does not realize how natural and convenient this is until one switches to half duplex after using full duplex for several weeks.) The remote stops echoing and sends a special character to the base to say that the terminal is in deferred echo mode. All echoing is now done from the host by a strategy that guarantees that the characters will come out in the right order. It is undesirable to stay in deferred echo mode because the echoing is so slow. It may take over a second for a character to make a round trip through the net if there are many nodes involved.

Before the remote can return this terminal to immediate echo, it must make sure that doing so will not cause characters to be echoed out of order. That is, it must make sure that the host has caught up with the input character stream and echoed all echoable characters. This condition is met if both pipes are empty and the program is dismissed waiting for an input character.

To test for this state, the remote places a green ball in the inbound pipe. When the green ball reaches the base, it waits until the program is dismissed waiting for an input character and the character buffers in the host associated with that program are empty. It then enters the outbound pipe and returns to the remote, possibly following some data characters. If it reaches the end of the pipe before any more data characters were sent in to the base, the terminal returns to immediate echo mode. If more unechoed data characters have been sent in during this time, then it is not safe to return to immediate echo because character echoes may follow the green ball.

A green ball can obviously take an arbitrarily long time to complete this trip. Once the remote has sent in another unechoed data character, it knows that the information that the green ball would convey if it returned is obsolete. That is, the pipes are not necessarily clear. Before the remote can make another attempt to get the terminal out of deferred echo mode, the old green ball must be flushed out. The remote places a red ball in the inbound pipe. The red ball simply goes to the base and returns, destroying any green ball it encounters.

This same machinery is used to find out when it is time to unlock the keyboard of an I.B.M. 2741. This is interesting because it allows the remote to completely shield an ASCII, full duplex oriented host from the awkward operating characteristics of that unfortunate terminal.

When an outbound pipe is full of characters, it may take several seconds to type out. Sometimes a program wishes to clear this pipe quickly, as when it realizes that the output is not wanted by the user. To clear the pipe, the host releases a character gobbler. The character gobbler races through the pipe at top speed, gobbling all characters ahead of it. When it gets to the end of the pipe, it annihilates itself.

Even more potent than the character gobbler is the circuit zapper. When a circuit is no longer needed, its buffer pairs and permuter table entries must be released quickly and cleanly. There should be no left over characters wandering aimlessly about the net. When the user logs off and hangs up his phone, or when the host decides to release a circuit, a circuit zapper is released. As it moves through the circuit, it clears all buffer pairs and leaves a trail of null permuter table entries behind it. When a circuit is zapped, it is completely erased. There are no remnants of it to pollute the net.

THE SUPERVISOR

The supervisor is a program that runs under timesharing on an XDS 940. Its purpose is to build circuits, perform diagnostics, keep statistics, and handle all other matters of global importance to the network. There are normally several supervisors running at any moment to provide backup in case of failure, but only one is in control of the network. This one is called the supervisor in active mode, or Sam, for short. Sam has in its memory a detailed representation of the network. In particular, it has a copy of all the permuter tables.

Every node in the network has a leprechaun. The purpose of the leprechauns is to issue high priority diagnostics, pass login information to Sam, and obey all commands from Sam, such as commands to change permuter tables. Every link has one full duplex supervisory channel for leprechaun communication. In every node, one link is called upstream, or toward Sam. Some nodes also have a downstream link. Messages moving on these links go upstream or downstream. If a leprechaun wishes to send a message to Sam, it places the message in the upstream link and the message percolates up to Sam. If Sam wishes to send a message to a leprechaun, Sam sets up a downstream path and sends the message down that path. Thus the leprechaun is a very simple creature with no global knowledge of the net. In fact, it takes less than 300 words of code.

Circuit building is Sam's primary function, and can be illustrated by several examples. Suppose a user calls a remote and identifies his terminal. The next thing he types is usually his name and password. A leprechaun in the remote sends these characters to Sam. Any further characters typed by the user remain in a buffer in the remote until a circuit is built. Sam hashes the user's name into the MUD (master user directory) to find out, among other things, which host has this user's files. The user himself may not know this since an operator may have moved them the night before. Sam then checks to see if that host is up and has an available port. The next step is to select the nodes and links through which to construct the circuit. Since the representation of the network is a multilinked list of node descripters, and since each link is tagged as being in the direction of a particular host or not in that direction, this is a trivial process requiring less than a millisecond of compute time. The only optimization done is to avoid links which are heavily loaded or which have high error rates. An unused virtual channel on each link is assigned to this circuit.

The user's name is now passed on to the host. The host hashes the name into the LUD (local user directory) to find the user's file directory, billing account, and so on. The final step is to send commands to the leprechauns in all the nodes affected to make the appropriate entries in their permuter tables.

The circuit just described is called a normal circuit. It is the only kind most programs use. Some programs use an auxiliary circuit to connect to things other than terminals. Suppose a program in host A wants information from the files of host B. It makes a supervisor request to construct an auxiliary circuit between its port on host A and a port on host B. The port on host A is multiplexed between the normal circuit and the auxiliary circuit, but only the program and Sam know this. Neither host knows that this is happening. Host B thinks the auxiliary circuit is a normal circuit with a terminal at the other end. The program in host A can interact with the program in host B just as a user would. In particular, the two programs can exchange data. Since the data rate is limited to about 150 characters per second, it is not a good way to move large files, but that is not an important restriction. Of course, the program in host B can acquire still another auxiliary circuit to host C.

Another use of the auxiliary circuit is attaching a program to some peripheral device, like a printer. The program supplies the logical name for the device, which is unique for every device. Sam locates the device (its location may change from day to day) and makes the connection. A potential use of auxiliary circuits, not yet implemented, is remote dialout. The program supplies the telephone number. Sam selects a node with a remote dialout unit on it which is inside or close to a toll-free dialing area for that number and makes the connection.

Another type of circuit building occurs when a node or link goes down. Sam will reroute all affected circuits for which an alternate route exists.

A critical situation is created when Sam goes down. Without Sam, no new circuits can be built and no new users can be accommodated. A new Sam must be created quickly.

There are normally many supervisors running besides the supervisor in active mode. They receive a message from Sam at least once a minute. Should these messages stop, they automatically begin to take over the net.

The network takeover involves three things. First, one supervisor must be chosen to be the new Sam. Second, this supervisor must construct an up-to-date representation of the net in its own memory. Third, it must win the allegiance of all the leprechauns.

The supervisor runs in a host, and the host is connected to a base. The supervisor wins the allegiance of the leprechaun in that base by causing the upstream direction to point into the host. Once that is done, the supervisor can find out all it wants to know about the base by asking the leprechaun. It then commands the leprechaun to send a takeover message down all of its supervisory channels. When a takeover message arrives in a node from a particular link, that link becomes the upstream direction. The leprechaun identifies itself on the new upstream link and the supervisor proceeds to find out what it needs to know to build its internal network representation. The supervisor continues node by node, level by level, until there are no more unknown links. When this is done, the upstream direction has been defined for all leprechauns, the internal network representation is complete, and the supervisor becomes the new Sam.

In the process of taking over the net, a supervisor creates a steadily growing sphere of influence. Should this sphere of influence intersect that of another supervisor, a leprechaun in the intersection of the two spheres switches allegiance. Just before switching allegiance, it sends a message in the old upstream direction saying which supervisor is taking over. Thus, when supervisor A steals a node from supervisor B, supervisor B discovers that supervisor A is trying to take over the net.

The supervisors are arranged in a pecking order. If an inferior supervisor steals a node from a superior supervisor, the superior supervisor will take it right back. If a superior supervisor takes a node from an inferior supervisor, the inferior supervisor becomes quiescent.

Several points are worth stressing here. First, a leprechaun has no global knowledge. This means that the software in a given node can be treated as an independent module which obeys certain conventions. This enormously simplifies the debugging and minimizes the danger of a bug in one node clobbering another node.

Second, the supervisor has no *a priori* knowledge of the net when it begins to take over. This makes it convenient to make alterations to the net. The supervisor simply adapts itself to whatever configuration is presented to it.

Third, the process is entirely automatic. The consequences of such a complex process being dependent on human operators are too horrible to contemplate.

Fourth, all global information is available in one single spot. This greatly facilitates diagnostics, record keeping, and debugging.

Fifth, the most complex routines, those of the supervisor, run under timesharing. The advantages of debugging under timesharing are difficult to exaggerate.

THE FUTURE

Technology of the type described in this paper is in a rapid state of flux. The price-performance ratio of minicomputers is improving, terminals are proliferating, and commercial timesharing is becoming a big business. Therefore, the threat of obsolescence becomes a major design consideration. Before one can intelligently discuss the threat of obsolescence, two general questions must be answered. First, what are the long-range objectives of the network? Second, in what directions is technology most likely to change?

Much research is currently being done on high speed graphics terminals, sophisticated text handling terminals, and on the problems of moving large masses of data quickly. These projects are glamorous. I concede that they are useful. But they are expensive, and therefore of limited use. I feel that, given the current market and technology, the greatest social and commercial potential lay with the low cost keyboard terminal.

I would like to make the timeshared computer available to anyone who wants it. In order to do this, the cost must be low enough so that everyone can afford it. Thus, although the network is very general purpose, its performance has been optimized around the needs of the simple low speed terminal with an attempt to serve the greatest number with the fewest dollars.

In what directions is technology moving? First of all, the variety of terminals is increasing. Since all terminal drivers are implemented entirely in software, it is easy to design a new one and make it available to all nodes of the net on short notice.

Second, the output character rates of "low cost terminals" are increasing. Keyboard CRT terminals come down in price every year and hard copy terminals are getting faster. I predict that in two or three years an acoustic coupler that operates at 1200 baud in one direction and 150 baud in the other will be in common use. Our algorithm for serializing and deserializing characters in software is well suited to outputting at 1200 baud and inputting at a much slower rate.

Third, the cost of moving data over long distance phone lines will come down, but at a slow rate compared to other aspects of the technology. 9600 bits per second is about it for voice grade lines, and although widespread availability of digital (rather than analogue) "voice grade lines" may be just around the corner on the telephone companies' time scale, it is an eternity away on the timesharing industry's time scale. This is already the dominating cost of the net, and the net is designed to optimize this resource.

Fourth, the speed of the minicomputer will continue to increase dramatically. In a system so totally committed to a software solution to all problems, the raw speed of the computer is obviously crucial. The recently available Varian 620f costs less than the 620i, yet it is more than twice as fast. Still faster computers will be available when we need them. Since the special hardware is so primitive, and since the software in any node is only about 4000 instructions, it is easy to switch to another computer. Since any node can be treated as an independent module following well defined conventions, it is easy to phase in new hardware and software (possibly not well debugged) one node at a time.

Finally, the future is guaranteed to have some surprises. One can never be absolutely certain how the future will turn out, and the only hedge against uncertainty is flexibility. Quite possibly the Tymnet of 1984 will be quite different from the Tymnet of today, but it is hoped that the evolution will be relatively painless.

REFERENCES

1 H B BURNER R MILLION O W RICHARD J S SOBOLEWSKI
The use of a small computer as a terminal controller for a large computing system
Proceedings of AFIPS SJCC 1969

2 L ROBERTS
Computer network development to achieve resource sharing
Proceedings of AFIPS SJCC 1970

3 S CARR S CROCKER V CERF
Host—Host communications protocol in the ARPA network
Proceedings of AFIPS SJCC 1970

4 K THURBER
Programmable indexing networks
Proceedings of AFIPS SJCC 1970

5 A WAKSMAN
A permutation network
Journal of the ACM Vol 15 pp 159-163 January 1968

Implementation of an interactive conference system

by THOMAS W. HALL

Language and Systems Development, Inc.
Silver Spring, Maryland

INTRODUCTION

This paper discusses a specific type of decision making through the use of a timesharing computer facility. The technique is variously called "conferencing" or "Delphi conferencing."* In conferencing, a computer would serve as a data collection and routing device which enables a geographically scattered group of experts on some subject (the conferees) to conduct remotely those discussions and/or referendums that might occur at a conventional, face-to-face conference. A computerized conference system must then handle the mechanics of running such a remote conference. Conference systems have been studied recently and advantages of this type of system have been discussed:** for instance, the remote accessibility itself; the possible anonymity of the respondents as a control over personality factors; and the capability of participating in the conference at one's convenience rather than at a precise time. The purpose of this paper is to present aspects of implementing a conference system. These aspects are of three types: functional (conferee oriented); control (conference chairman oriented); and implementation (program requirements to provide control and functional capabilities). Following this discussion of these three aspects, some features of an actual conference system implementation are reviewed.

FUNCTIONAL ASPECTS

The functional aspects of a conference system consist of what the user of the system can actually do. The user of a conference system—denoted a "respondent"—requires:

1. The ability to access a data base available to all the respondents. This data base is a representation of the activities of a conference—consisting principally of the texts of the things being presented for consideration by the respondents.
2. The ability to contribute to the activities of the conference in prescribed ways—expressing opinions, proposing new topics for consideration, and recording quantitative judgments on topics, e.g., by voting *yes* or *no* on a proposal.

In moving from these requirements to actually implementing a conference system, we see that what distinguishes an implementation of a conference system from a user's point of view is then:

1. How accurately the system's data base structure can mimic the conference situation the user needs; and
2. What the limits on his use of the system are—exactly what he can and cannot do.

From this point, it is possible to proceed to implement an almost endless variety of conference systems. What follows in this paper is a description of a system that has been implemented. Access to a prototype of this system was made available to a group of respondents—the subject being the conference itself. As a result of this experiment, the following functional system was designed and implemented.

Access

To participate in a conference, a respondent must first contact the computer. This normally entails using a remote device such as a Teletype(R) to call the phone number tying into the computer's timesharing system.

* The term "Delphi" refers generally to non-face-to-face interactions. The users involved in the exercise described in (2) correctly applied the name "Delphi Conference" to the exercise and this appellation has survived.
** See "Delphi Conferencing" (TM 125) by Dr. Murray Turoff of Systems Evaluation Division, Office of Emergency Preparedness, Washington, D.C.

Next, the user has to provide the identification and accounting information the monitor system needs to allow him to sign on. Then the user has to indicate that he wishes to use the conference system rather than some other program. (The accessing procedure up to this point has been known to intimidate some first-time computer users, but it is just a fixed sequence of things to do ritualistically.) When the conference system is finally reached it takes control from then on. The conference system asks for a short code word which the respondent has been given by whoever arranged the conference. This code first of all allows the conference system to tell which conference the user is participating in—the system is capable of conducting several conferences at once, each with its own data base. The code also uniquely identifies the respondent within the conference so that his activities can be accounted for separately from all other respondents. Finally, this unique identification allows the system to give different privileges to different users. A given respondent may have any or all of the following privileges:

1. viewing the items, messages and votes (see the following pages for definition of these quantities;)
2. voting on items;
3. adding items or adding messages.

Interaction

When the conferencing system accepts the user's code, it offers the user his choice of seeing an abstract of the subject of the conference, or of using a tutorial program explaining the system, or of proceeding to the heart of the interaction. When he has proceeded to the main interaction, he is given the choice of:

1. viewing summary information;
2. viewing items or messages;
3. voting on items;
4. adding items or messages (or amending those he has already added.)

Items, messages, and votes are the basic types of data handled by the conference:

1. *Items* are blocks of concise textual data which are to be presented to the respondents for their consideration and evaluation. Four types of items are recognized:

   a. Proposals—proposed actions or policies of which the respondents are asked to judge the desirability;
   b. Comments—discussion points which may be evaluated by the respondents for their importance;
   c. Facts—quantitative points of information which may be of interest in the discussion—respondents may evaluate facts on their pertinence to the subject of the conference;
   d. Estimates—requests for numeric estimates from the respondents—again the respondents may vote on the relevance of such estimates.

As noted, proposals, comments, facts, and estimates are evaluated by respondents according to the primary criteria of desirability, importance, pertinence, and relevance, respectively. Additionally, items can be graded by the respondents on secondary criteria—for example: agreement with a comment, feasibility of a proposal, confidence in an estimate, or impact of a fact. Items can optionally be associated with one another, linking together items on the same subject.

2. *Messages* are blocks of textual data which are not voted on by respondents and may be used for such purposes as pure discussion or to make comments about or clarifications of items. Messages may optionally be linked or associated with items, also.
3. *Votes* are the responses of the conference users to the items of the conference. Votes on the primary and secondary criteria are recorded on a scale of 1 to 5, with the exact vote distribution being made available to the respondents only after fifty percent of the vote is in and only after the respondent who requests to see the votes has himself already voted. User responses to estimate items are also part of the voting; the number, average, and standard deviations of the estimates being recorded, along with the number of estimates above the mean—a rough measure of skewness. As a result of the respondents' votes, items are segregated according to those which have been accepted, those which have been rejected, and those which are still pending.

The process of adding items and evaluating those which other respondents have added constitutes the essential user interaction.

CONTROL ASPECTS

The control aspects of a conference system consist of the overall control mechanisms for a conference. In the line of controlling a conference, the following capabilities must be examined:

1. organizing a conference;
2. exercising dynamic control over the proceedings of the conference;
3. examining the results of the conference;
4. disbanding the conference at its conclusion.

Again in our particular implementation, these capabilities appear in concrete form. Instead of identifying himself to the conference system as a regular user, one can take the role of a conference chairman or "monitor." Obviously, the effort of finding people to participate in the conference and the task of guiding the conference proceedings along a coherent path still tax the human resources of the conference monitor. The conference system only takes care of the mechanics.

Setting up a conference

A user first creates the conference of which he is to be the monitor. In doing so, he establishes a monitor code or "key" which is required before he or anyone else can act as the monitor for that particular conference. Creating a conference entails:

1. establishing a list of respondents—supplying code words for their access to the system and indicating which permissions (votes, adding items, etc.,) are accorded to each respondent;
2. providing the abstract of the conference's subject;
3. possibly providing an initial set of items for the consideration of the respondents.

Modifying the conference

Occasionally during the life of a conference, the monitor may have to intervene in one of the following ways:

1. to alter or completely purge inappropriate items or messages or to rearrange the association links between items into a more correlated structure;
2. to change the list of respondents—adding new ones or deleting old ones;
3. to communicate important information to the respondents in the form of a "monitor message" which is presented to each respondent the next time he uses the system.

Analyzing the conference

In addition to the vote totals made available to the respondents, the monitor of a conference can view the individual voting records of the respondents and can summarize the voting of various groups of respondents.

Deleting the conference

At the conclusion of a conference, the monitor directs the conference system to eliminate the conference from the computer system—reclaiming whatever storage the conference utilized.

IMPLEMENTATION ASPECTS

So far, this paper has discussed what a particular conference system does. This is a major part of implementing the abstract idea of a conference system—the designing part. Now, we turn to the actual realization of the design—making a working system to run a conference in the manner previously described. This part of the job—which is the part usually thought of as implementation—involves several areas of general concern which can be considered independently of a particular system.

Hardware and monitor systems

The nature of a conferencing situation as described up to this point makes evident some initial requirements.

1. *Communications*

To serve its intended purpose, the conference system must be convenient to geographically scattered users. Optimally, the system should accept interaction with the user from whatever remote device he finds most accessible. In practical terms this is best accomplished by such means as a dial-up Teletype arrangement as used by most commercial timesharing services. Where possible, higher speed devices including graphics display terminals, should be accommodated by the system for the user's convenience.

2. *Operation under a Timesharing System*

At any one time, the conference system may be in use by no respondents or by several. In any case, the vast majority of the computer's time would be spent idle if it had to run only this one system. This problem and many others are solved by writing the conference system as a subsystem of a standard timesharing system. This subjects the user to variations in response time due to totally unrelated uses of the timesharing computer, but this disadvantage is insignificant compared to the advantages. With a standard timesharing system, the desired communications capabilities should be already included; the job of dividing time between respondents becomes the monitor system's task instead of the conference system's; and the power and flexibility of the timesharing system's software become available.

Required features

Having made the decision to operate the conference system under a timesharing monitor, one must require the monitor system to allow all the operations absolutely essential to the running of a conference.

1. *Common Access*

First of all, the respondents must be able to run the program or programs constituting the conference system. If possible, the conference system might be placed among the standard components of the monitor system to make access to it as easy and natural as possible. Ideally, the conference system should be a re-entrant program so that several respondents could be served by the single copy of the conferencing program. Second, the monitor system must allow the concurrent use of a common area of mass storage (a file) by several respondents. Access to this area must be *direct* in the sense that what a user writes there can immediately be read by another user—i.e., all users agree as to what is in the file at any given time. This essentially requires unbuffered input/output to be used. In addition to providing common access to a file, the monitor system must also be able to limit it as noted in the next point.

2. *Exclusive Access*

When two or more user programs in a timesharing situation independently desire to change the contents of a common file, some means must be used to insure that their attempted changes do not interfere with one another. This is done in practice by guaranteeing one user program at a time exlcusive access to the area he wishes to change. Three techniques are available for this purpose: first, a program might arrange to raise its priority sufficiently to guarantee that it can get done what it wants to get done without interruptions; second, the program might request the monitor system to temporarily grant it exclusive use of the entire file; and, third, the program might ask the monitor to grant it exclusive use of just the portion of the file that had to be modified. Any or all of these features may be available for use in a given timesharing system.

Programming Systems

In actually programming a conference system, one must choose a language and programming system which offers access to all the required features mentioned previously. It is of no use if the monitor system offers exclusive access to common files if the language chosen for the conference system cannot make use of this feature. It is sufficient to choose a language which can communicate with the presumably omnipotent assembly language. It is not completely unreasonable to write the conference system entirely in the assembly language, but use of a higher level language suggests itself for several reasons:

1. *Availablity*

Oddly enough, many timesharing systems offer the remote user no access to assembly language programming. The philosophy behind this seems to be twofold: first, the timesharing user is not seen as requiring any of the features available only in assembly language; and, second, access to all the features open to assembly language programs may be hazardous to the stability of the system. Conversely, higher level languages such as BASIC and FORTRAN are made commonly available.

2. *Programming Ease*

Unless the programmer(s) assigned to code a conferencing system are of the extremely rare breed of systems programmers who prefer assembly language coding, the time required for implementing a conference system in an appropriate higher level language should be considerably less. Also, mistakes in higher level language coding are apt to be less troublesome—either because fewer mistakes are made or because they are more easily found and corrected with the help of the diagnostic aids accompanying more advanced programming systems.

3. *Use of Packaged Features*

In using a high level language, one can often make use of powerful features built into the language which would be difficult to rival in assembly language. The most important of these are the input/output operations which are often the most difficult part of assembly language level programming to master. Also, the higher level language should offer capabilities for manipulating string data and for data formatting chores.

4. *Efficiency*

Clearly, the execution of a conference system carefully written in assembly language would be more efficient in terms of computer usage than that of one written in a higher level language. However, this would appear to be of negligible interest for two reasons: first, the saving in effort of implementing a system in a higher level language should counterbalance the execution differences; and, second, the nature of a conference system is such as to make it entirely input/output oriented or "I/O bound," waiting for the user's input nearly continuously. This means that the response time for a user—which is the most important feature of such an interactive system—will not be appreciably affected by the inefficiencies in computing due to the higher level language. It also means that the dollar cost due to use of the computer's processor will probably be smaller than the cost of calling up the computer in the first place.

FEATURES OF AN ACTUAL IMPLEMENTATION

To explore the requirements for implementing a conference system further and to show how the required features come into use in practice, the techniques used in an actual implementation of a conference system* should be examined.

Choice of computers

All the features necessary for implementation of a conference system are readily available on the UNIVAC 1108** system operating under the standard UNIVAC monitor system, Exec 8. The important features of the Exec 8 system for this purpose are: its timesharing capabilities—allowing remote users access to the full resources of the system, including the assembly language; and its flexible file handling capabilities.

Programming language

On the UNIVAC 1108 used, an extended version of the BASIC language, XBASIC,*** was available for use. The theoretical justifications for writing a user conference system in a higher level language, or a user programming system centered around such a language, have already been stated. Writing the conference system in XBASIC made available the powerful features of the XBASIC language, such as string, vector, and data file manipulation capabilities. Even more significantly, using the XBASIC system made writing the conference system much easier. In fact, the system was developed entirely from remote terminals—a rather remarkable occurrence: using one remote user oriented system (XBASIC) to develop another (the conference system). One final consideration in using XBASIC was that XBASIC is not a completely static system—if new features were seen to be necessary for such tasks as writing a conference system, they could have been added to the language readily to produce a better programming language at the same time. As a result of the conference system project, a re-entrant version of XBASIC specifically streamlined for running other user systems was created.

Data structures

1. *Files*

The data base for each conference (there can be up to 26 active simultaneously) is stored on two random access mass storage files. Names for these files are generated by the conference system and all control of these files is done internally by the system without the user's having to know anything about Exec 8 file handling. This internal allocation, use and release of files under program control without bothering the user with facility assignment or "data definition" control

* Implementation of the conference system described was done by the author for Language and Systems Development, Inc., 8121 Georgia Avenue, Silver Spring, Maryland, 20910. The work was partially supported by the Department of the Army, Corps of Engineers (Contract DACA31-70-C-0088).

** UNIVAC is a registered trademark of the Sperry Rand Corporation.

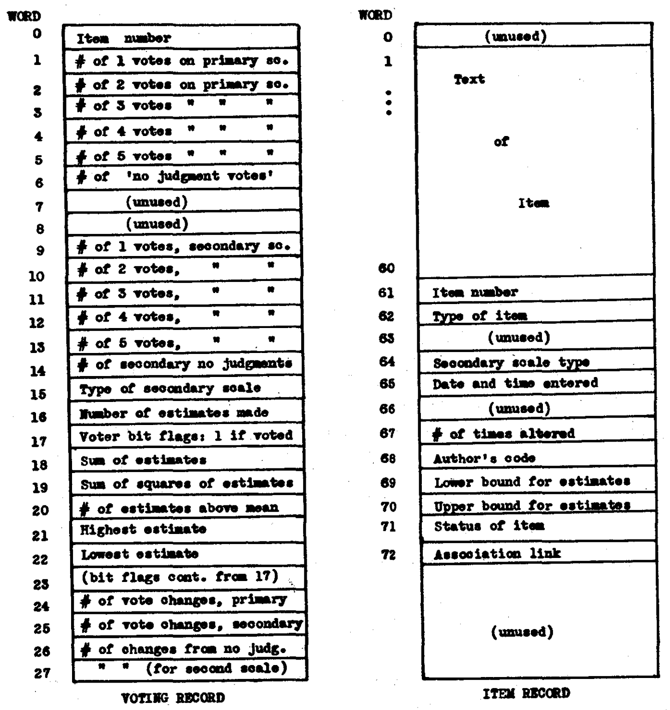
*** XBASIC is a proprietary processor developed by Language and Systems Development, Inc. See "XBASIC for the Univac 1100 Series Computers," Language and Systems Development, Inc. (1969)

cards is extremely desirable in a user oriented system. Two files are used by the conference to conserve storage. Since Exec 8 offers dynamically expanding files, the conference system only uses the space in these files as it needs to, using them as two pushdown lists. The two files are automatically placed by the conference system on the best (fastest) type of mass storage available at the time—ranging from 4.3 millisecond average access time for the fastest to 92 milliseconds average time for the slowest.

2. *Data Items*

The storage used for the conference system's files is actually only randomly accessible down to 28 word blocks. This presents the challenge of fitting the required data efficiently into convenient multiples of 28 word blocks. In most cases, this proved fairly easy. The primary simplifying assumption permitting the use of fixed size blocks of storage is that textual items have a maximum length (360 characters.) This is acutally a requirement imposed in the design of the system to force brevity on the respondents, not a requirement imposed by the implementing programmer.

(For sample data block formats, see Figure 1.)



WORD	VOTING RECORD
0	Item number
1	# of 1 votes on primary sc.
2	# of 2 votes on primary sc.
3	# of 3 votes " " "
4	# of 4 votes " " "
5	# of 5 votes " " "
6	# of 'no judgment votes'
7	(unused)
8	(unused)
9	# of 1 votes, secondary sc.
10	# of 2 votes, " "
11	# of 3 votes, " "
12	# of 4 votes, " "
13	# of 5 votes, " "
14	# of secondary no judgments
15	Type of secondary scale
16	Number of estimates made
17	Voter bit flags: 1 if voted
18	Sum of estimates
19	Sum of squares of estimates
20	# of estimates above mean
21	Highest estimate
22	Lowest estimate
23	(bit flags cont. from 17)
24	# of vote changes, primary
25	# of vote changes, secondary
26	# of changes from no judg.
27	" " (for second scale)

WORD	ITEM RECORD
0	(unused)
1 ⋮ 60	Text of Item
61	Item number
62	Type of item
63	(unused)
64	Secondary scale type
65	Date and time entered
66	(unused)
67	# of times altered
68	Author's code
69	Lower bound for estimates
70	Upper bound for estimates
71	Status of item
72	Association link
	(unused)

Figure 1—Sample data block formats

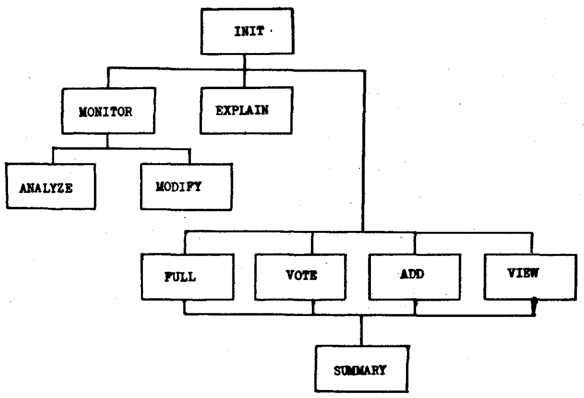


Figure 2. Program Divisions

Legend:

INIT	-- controls access to the system
EXPLAIN	-- provides an explanation of the system
MONITOR	-- supervises activities of the conference monitor
ANALYZE	-- provides reports on the conference proceedings
MODIFY	-- allows the monitor to modify the conference
FULL	-- provides the main user interaction for full respondents
VOTE	-- user interaction for ones who can only vote and view items
ADD	-- user interaction for those who can only add or view items
VIEW	-- user interaction for those who can only watch
SUMMARY	-- provides summary information to the users

Figure 2—Program divisions

3. *Program Structures*

All together, the conference system requires somewhat more than 2000 lines of XBASIC program to function. However, only segments of the system are actually required by a user at any one time. To keep the respondents from tying up core storage with unused portions of the system, there are actually several programs comprising the system. When the user requests a feature of the system implemented by a program other than the one he is currently executing, the current program deactivates itself and calls in the new program (program "chaining"). The various programs are organized so as to minimize the number of chaining operations required. As an additional feature of the system, a respondent who is not entitled to vote is chained to an interaction program which does not contain code to perform voting—saving some amount of memory space.

(For organization of the program chains, see Figure 2.)

Control mechanisms

1. *User Directives*

The user directs the conference system by making choices at various junctures in the logic of the system.

The system indicates what the available choices are (or can be asked to skip the explanations for experienced users) and then requests the user's decision in the form of a numeric choice. The system does recognize such things as "YES" and "NO" choices; but for the more complicated multiple-choice decisions, making numeric choices is the best method in the long run—being faster to type with less chance of error than attempting to use keywords. Input is taken by the system in string form and decoded by the program so that it can recognize errors and act on them without subjecting the user to the monitor system's error messages about such things as improper numeric formats.

2. *Common File Management*

The careful control of access to the common data file of a conference is extremely important in the implementation of the conference system. The problem is complicated by the way items are stored in the files. To keep an upper bound on the amount of file storage used by a conference, only the fifty most recent items are kept available. The items are stored in the file in a circular manner—the fifty-first item is written into the area formerly used by the first item. The conversion from external numbers of items to their internal positions in the file is handled fairly easily with modular arithmetic. The trouble comes when the user asks for the first item as it is being outdated and replaced by the fifty-first. Other conflicts arise with two users attempting to update the vote totals at one time or two users attempting to add an item at the same time.

As indicated in the section on Functional Aspects, several methods are available for resolving conflicts in use of common files. In the Exec 8 system, one can either ask for exclusive rights to a file—if someone else is using it, the request is rejected and must be repeated; or one can ask for exclusive use of a small block of the file, "locking" it—if someone else attempts to reference this block, his request is automatically delayed until the lock is removed. This second method is by far the better—it offers much finer control over the file's use; and it turns out to be over thirty times faster to use, due to the way Exec 8 operates. The feature of locking and unlocking portions of the file is used to direct access to the common file via a method which could be called "key block" control. For example: to add an item, a user's conference program must lock the key block necessary for this function—namely the directory block containing a pointer to the next available place in the file. The user then adds his item, updates the directory block and unlocks it. During this time, other users can be accessing other portions of the file freely.

Obviously, the more key blocks are locked and the longer they are locked, the slower the system will respond to other users. Several devices are used to circumvent this:

a. The functions of key blocks are distinct so that, for example, a user adding an item only locks out other users attempting to do the same thing.
b. All preparations for changing the file are made before the locking is done; then the file is changed as rapidly as possible and unlocked. The principal rule here is to never have a key block locked while waiting for input from a respondent.
c. Control of key blocks is not required by all operations which just read information.

Note that (c) implies that the problem of attempting to read an item which has just been outdated has not yet been resolved. This problem is handled by having each item or similar type of record explicitly contain the external item number to which it corresponds. When the system reads an item, it checks to be sure that the item number in the data read matches the one it was expecting—if the check fails, the system knows the desired item has been outdated and tells the user so. Happily, this checking also protects the conference system against failures of the computer system. If the computer stops after an item has been changed but before the changes are made in the key directory blocks, the check for outdated items should discover the discrepancy.

EXAMPLES OF INTERACTION

The following pages are copies of actual sessions using the conference system. In these examples, the information in lower case letters or following question marks represents input supplied by the user. All of the details of the interaction and the meanings of some of the choices offered by the system are not discussed in this paper. The examples are presented only to give the reader the flavor of the interaction.

The first example covers the setting up of a conference (in this case with only one respondent—the creator of the conference). The second explores the entering of an item into the conference, voting on the item, and seeing the results of the voting. The final example shows the monitor having the results of the conference printed (the results being very sparse in this case).

The remote device used for the interaction was a Teletype Model 37. The cooperation of the Teletype Corporation in adapting this device for use with the computer system was greatly appreciated.

```
DELPHI CONFERENCE SYSTEM AT  141830  ON  123170
PLEASE TYPE YOUR CODE? monitor




DO YOU WISH TO:
  SET UP A CONFERENCE     (1)
  DELETE A CONFERENCE     (2)
  MODIFY A CONFERENCE     (3)
  ANALYZE A CONFERENCE    (4)
  RETURN TO MAIN SYSTEM   (+)
  TERMINATE               (-)
MONITOR CHOICE:? 1
BY WHAT CODE SHOULD YOU BE KNOWN (5 SYMBOLS)? twh
ALL YOUR CODES SHOULD BEGIN WITH THE LETTER A
YOUR CODE IS AUTOMATICALLY ATWH
TYPE CODES FOR OTHER PARTICIPANTS.
WHEN ASKED 'VERIFIED?', HIT ONLY RETURN KEY IF OK, ANY SYMBOL IF NOT.
WHEN ASKED 'CODE:?' AND YOU HAVE NO MORE CODES, HIT JUST THE RETURN KEY.
BUT FIRST, SUPPLY INFORMATION FOR YOURSELF:
WHAT CLASSIFICATION CODE? 11
LIST PERMISSIONS: VOTE(1),ADD ITEM(2),ADD MESSAGE(3):? 1,2,3
HIT RETURN IF FULL EXPLANATION IS TO BE GIVEN WHEN THIS
CODE FIRST LOGS IN. TYPE ANY SYMBOL IF NOT.? no
TYPE ANY TWO LINES OF INFO. FOR FUTURE REFERNCE:
? this is alphabetic information to identify this respondent.
?
VERIFY: CODE= ATWH PERMISSIONS=
 1  2  3
CLASSIFICATION= 11
THIS IS ALPHABETIC INFORMATION TO IDENTIFY THIS RESPONDENT.

VERIFIED?
CODE:?
IS THE 'WAIT' CHOICE ALLOWED. RETURN=YES, ANY SYMBOL=NO? no
THE SYSTEM WILL KEEP A BACKUP TAPE ASSOCIATED WITH THIS CONFERENCE.
SUPPLY A TAPE REEL NAME TO BE USED FOR THIS PURPOSE. IF YOU DO NOT
HAVE A TAPE ALREADY, HIT JUST THE RETURN KEY AND THIS SYSTEM
WILL REQUEST ONE FROM THE OPERATOR. TAPE NUMBER:? none
TYPE FOUR LINES OF SUBJECT DEFINITION.
FOR THE FIFTH LINE, TYPE YOUR NAME AND HOW TO REACH YOU.
? this is a demonstart___ration conference.
? it has no real subject.
?
?
?   this would be the monitor's name and phone numbe_.
THIS IS A DEMONSTRATION CONFERENCE.
IT HAS NO REAL SUBJECT.
   THIS WOULD BE THE MONITOR'S NAME AND PHONE NUMBER.
VERIFIED?
DO YOU WANT A 'MONITOR MESSAGE' NOW.
HIT RETURN IF YES, ANY SYMBOL IF NO.? no
CONFERENCE SUCCESSFULLY CREATED.
MONITOR CHOICE:? +
DELPHI CONFERENCE SYSTEM AT 145603 ON 123170
PLEASE TYPE YOUR CODE? atwh
DO YOU WISH
   AN EXPLANATION     (1)
   SUBJECT DEFINITION (2)
   LONG FORM          (3)
   SHORT FORM         (4)
CHOICE? 3




DO YOU WISH TO:
  VIEW SUMMARIES              (1)
  VIEW ITEMS                  (2)
  VIEW MESSAGES               (3)
  VIEW VOTES                  (4)
  VIEW AND VOTE ON ITEMS      (5)
  VOTE                        (6)
  ADD AN ITEM OR MESSAGE      (7)
  MODIFY AN ITEM              (8)
  WAIT                        (9)

MODE CHOICE:? 7
ITEM OR MESSAGE MUST FIT IN SIX LINES OF 60 CHARACTERS.
HIT RETURN KEY WHEN EACH LINE IS COMPLETED AND WAIT
FOR THE QUESTION MARK BEFORE BEGINNING NEW LINES.
YOU MUST SUPPLY SIX LINES EVEN IF THEY ARE PUT IN
BLANK BY HITTING THE RETURN KEY AT THE BEGINNING OF THE LINE.
 ADD ITEM OR MESSAGE IN NEXT SIX LINES. END OF LINE IS HERE   *
? this is a test item for demonstration purposes only.
? treat this item as an estimate. when asked for your
? estimate, supply an arbitrary number within the given
? limits.
?
?

THIS IS A TEST ITEM FOR DEMONSTRATION PURPOSES ONLY.
TREAT THIS ITEM AS AN ESTIMATE. WHEN ASKED FOR YOUR
ESTIMATE, SUPPLY AN ARBITRARY NUMBER WITHIN THE GIVEN
LIMITS.


IS THE ABOVE CORRECT AS STATED:
YES(1), NO(2), OR NULLIFY(+)
CONTINUE CHOICE:? yes
INDICATE ASSOCIATION WITH EXISTING ITEM (NOT
MESSAGE) BY SUPPLYING EXISTING ITEM NUMBER.
IF NO ASSOCIATION, ENTER ZERO(0).
ASSOCIATION CHOICE? 0
INDICATE TYPE:
  PROPOSAL  (1)
  COMMENT   (2)
  FACT      (3)
  ESTIMATE  (4)
  MESSAGE   (5)
TYPE CHOICE:? 4
```

```
WHAT IS THE LOWER BOUND FOR ESTIMATES? 0
WHAT IS THE UPPER BOUND? 100
INDICATE SECONDARY EVALUATION SCALE:
  NO SECOND SCALE   (0)
  CONFIDENCE        (1)
  AGREEMENT         (2)
  FEASIBILITY       (3)
  IMPACT            (4)
  PROBABILITY       (5)
  ARBITRARY         (6)
SCALE CHOICE:? 0
TYPE: E SEC. SCALE: NONE ITEM >0
LB: 0  UB: 100
ARE YOU SATISFIED WITH THE ABOVE CHOICES:
YES(1), NO(2), OR NULLIFY(+).
CONTINUE CHOICE:? 1
ITEM OR MESSAGE ENTERED: 123170 AT 150122
3E:
THIS IS A TEST ITEM FOR DEMONSTRATION PURPOSES ONLY.
TREAT THIS ITEM AS AN ESTIMATE. WHEN ASKED FOR YOUR
ESTIMATE, SUPPLY AN ARBITRARY NUMBER WITHIN THE GIVEN
LIMITS.
0<3
DO YOU WISH TO ADD OR MODIFY ANOTHER ITEM OR MESSAGE
YES(1) OR NO (2).
CONTINUE CHOICE:? 2
DO YOU WISH TO:
  VIEW SUMMARIES             (1)
  VIEW ITEMS                 (2)
  VIEW MESSAGES              (3)
  VIEW VOTES                 (4)
  VIEW AND VOTE ON ITEMS     (5)
  VOTE                       (6)
  ADD AN ITEM OR MESSAGE     (7)
  MODIFY AN ITEM             (8)
  WAIT                       (9)

MODE CHOICE:? 5
DO YOU WISH ITEMS PRESENTED BY:
  LIST ORDER             (1)
  SINGLY BY NUMBER       (2)
  ASSOCIATIONS           (3)
  THOSE NEW OR MODIFIED  (4)
  THOSE ACCEPTED         (5)
  THOSE SIGNIFICANT      (6)
  THOSE PENDING          (7)
  THOSE INSIGNIFICANT    (8)
  THOSE REJECTED         (9)
ORDER CHOICE:? 2
ITEM:? 3
```

```
3E:
THIS IS A TEST ITEM FOR DEMONSTRATION PURPOSES ONLY.
TREAT THIS ITEM AS AN ESTIMATE. WHEN ASKED FOR YOUR
ESTIMATE, SUPPLY AN ARBITRARY NUMBER WITHIN THE GIVEN
LIMITS.
0<3
YOU HAVE NOT YET VOTED ON THIS ITEM.
PER: LAST VOTE: 0 PRESENT CHOICE:? 3
ESTIMATE BETWEEN 0 AND 100
NO LAST CHOICE.  PRESENT CHOICE:? 63
ITEM:? 3

3E:
THIS IS A TEST ITEM FOR DEMONSTRATION PURPOSES ONLY.
TREAT THIS ITEM AS AN ESTIMATE. WHEN ASKED FOR YOUR
ESTIMATE, SUPPLY AN ARBITRARY NUMBER WITHIN THE GIVEN
LIMITS.
0<3
CODE: (1) (2) (3) (4) (5) (6)   AVE
 PER:  0   0   1   0   0   0    3.0
    N:   1  A:      63.00  AA:   0
   SD:        .00  LE:      63.00  HE:      63.00
PER: LAST VOTE: 3 PRESENT CHOICE:? 1
ESTIMATE BETWEEN 0 AND 100
LAST CHOICE: 63 PRESENT CHOICE:? 65
(NOTE: THAT VOTE CHANGES THE ITEM'S STATUS.)
ITEM:? +
DO YOU WISH TO:
  VIEW SUMMARIES               (1)
  VIEW ITEMS                   (2)
  VIEW MESSAGES                (3)
  VIEW VOTES                   (4)
  VIEW AND VOTE ON ITEMS       (5)
  VOTE                         (6)
  ADD AN ITEM OR MESSAGE       (7)
  MODIFY AN ITEM               (8)
  WAIT                         (9)

MODE CHOICE:? 1


          ACTIVITY SUMMARY

THERE ARE NO MESSAGES.
ITEMS:      1 TO 3
  TYPES: P:  0 C:  1 F:  0 E:  2
 STATUS: A:  1 S:  0 P:  2 I:  0 R:  0 PURGED  0
  ACTIVE VOTERS                        1
  ACTIVE VIEWERS                       0
  TOTAL LOGINS                         7
  TOTAL VOTES                          1
  TOTAL VOTE CHANGES                   2
  VOTE CHANGES FROM NO JUDGEMENT       0
  NO JUDGEMENTS                        0
DO YOU WISH:
   ASSOCIATION MAP       (1)
   ITEM SUMMARY          (2)
   YOUR VOTE SUMMARY     (3)
   RETURN TO MODE CHOICE(+)
SUMMARY CHOICE:? -
```

```
DELPHI CONFERENCE SYSTEM AT 151230 ON 123170
PLEASE TYPE YOUR CODE? monitor




DO YOU WISH TO:
  SET UP A CONFERENCE     (1)
  DELETE A CONFERENCE     (2)
  MODIFY A CONFERENCE     (3)
  ANALYZE A CONFERENCE    (4)
  RETURN TO MAIN SYSTEM   (+)
  TERMINATE               (-)
MONITOR CHOICE:? 4
WHAT IS YOUR MONITOR CODE? atwh


ANALYSIS  OF  DELPHI  CONFERENCE  NUMBER 1
DO YOU WISH INFORMATION ABOUT EACH RESPONDENT TO BE PRINTED.
YES(1) OR NO(2):? 1
WHAT ARE THE LIMITS ON FIRST DIGIT OF CLASSIFICATION CODE:
TYPE  LB,UB:? 1,9
WHAT ARE THE LIMITS FOR THE SECOND DIGIT:? 1,9


CODE:  ATWH   -- VOTING RESPONDENT
INFORMATION SUPPLIED BY MONITOR:
THIS IS ALPHABETIC INFORMATION TO IDENTIFY THIS RESPONDENT.
NUMBER OF LOGINS=             7
PERMISSIONS =                 7 (PACKED FORMAT)

ITEM NUMBER 1 VOTING RECORD:
HAS NOT VOTED ON THIS ITEM.

ITEM NUMBER 2 VOTING RECORD:
HAS NOT VOTED ON THIS ITEM.

ITEM NUMBER 3 VOTING RECORD:
LAST VOTE ON SCALE 1 =        1
LAST VOTE ON SCALE 2 =        0
NO. VOTE CHANGES ON PES =     2
NO. VOTE CHANGES ON SES =     0
NUMBER OF CHANGES FROM NJ =   0
LAST ESTIMATE =               65


TOTAL AVERAGES FOR THIS GROUP:
NUMBER OF CODES COUNTED =     1
TOTAL LOGINS =                7
VOTE CHANGES ON SES =         2
VOTE CHANGES ON PES =         0
VOTE CHANGES FROM NJ =        0

ITEM  AVE. ON PES   AVE. ON SES
   3         1.00   NO SES

 ******** END OF SUMMARY **********


 TIME :   1.522
```

ACKNOWLEDGMENTS

This paper is an outgrowth of work done on the subject of conference systems by Dr. Murray Turoff of the Systems Evaluation Division, Office of Emergency Preparedness, Washington, D.C. The author worked with Dr. Turoff in this area during the year of 1970. The design of the user's interaction with the conference system cited in the paper is due to Dr. Turoff—the expansion of the design into a complete conference system and the implementation of this system being the author's concern.

The cooperation of the author's associates at Language and Systems Development, Inc., should also be acknowledged—their ideas on systems programming being, as always, useful.

Who are the users?—An analysis of computer use in a university computer center*

by EARL HUNT, GEORGE DIEHR and DAVID GARNATZ

University of Washington
Seattle, Washington

INTRODUCTION

This is a study of how the users of the University of Washington computing center exercise its machinery. Our hope is to make an undramatic but useful contribution to knowledge. In a simpler day the distinction was made between "scientific" and "business" computing. Undoubtedly this contrast is still useful for many purposes, but finer distinctions are needed. We shall present statistics showing that, within a community which contains not a single "business" user, there are distinct groups with quite different machine requirements. Of course, nobody who is aware of modern computing would seriously dispute this. Our contribution is to provide statistics on the relative size of the different groups. We also offer this report as an example of methodology. The usefulness of our numbers to another center will depend upon the extent to which the other center is like ours. The ways in which we acquired and analyzed our statistics would be useful more generally.

From the viewpoint of the Computer Center, a knowledge of user characteristics is important in planning. In the particular center we studied, and others like it, there will probably be no major change in the types of computing done over the next five years (unless qualitatively different equipment capabilities are provided), but there will be a steady increase in the number of users. The characteristics of this increasing population must be known in order to anticipate bottlenecks and to plan for orderly expansion. Users also need to know something about themselves. Time is expensive, so computer use must be estimated as accurately as possible in budget preparation. In the days before multiprogramming, one simply rented the entire computer configuration for a few seconds, even if only half of it was used. Today charges are based upon use of memory size, processor time, and peripherals. To make accurate estimates of his needs, the user must ask "What resources do people like me actually utilize?" Consider the problem of the instructor trying to estimate the cost of a course in programming. What he knows is that he will have n students, k problems, and that the problems will take an average of m runs to solve. These runs vary greatly as the students progress from incompetence with control cards to an ability to write infinite loops. To estimate the cost of computing, the instructor needs statistics about how student jobs perform. The research scientist who has not yet settled on his batch of production programs (and who may never find them) is in a similar situation. He knows how many people he has on his project and knows how often they submit programs. He also knows that the programs vary greatly as he and his associates go through cycles of planning, debugging, production modification, and reprogramming. To estimate his budget he needs *applicable* averages.

As our final justification, we point to an application of user statistics within Computer Science. The use of models to predict system performance has become increasingly popular in system evaluation. Basically, the idea is to view a computing configuration as a job shop servicing jobs drawn at random from a population of users, and then to analyze a model of such a service. In order to make the model anything more than an exercise in mathematics, however, one must show a correspondence between it and reality. Here we present some statistics which can be appealed to in justifying a model of the user.

THE UNIVERSITY AND THE CENTER

Some words about the setting of our study are in order. The University of Washington is a large state

* This research was supported by the Institutional Research Fund of the University of Washington, Seattle, Washington.

university with about 33,000 students, 20 percent of them in graduate or professional schools, and a faculty of roughly 2,500. The University computer center provides general support for this community. Specialized research computing capabilities needed for process control or real time applications are provided by dedicated installations scattered throughout the campus. We did not study these. The University's administrative data processing is done on a dedicated Burroughs B5500 computer, and hence is also not included in this study.

The center's "scientific" computer is a Control Data 6400 system with 65,000 sixty bit words. It is used in batch mode under control of the SCOPE 3 operating system. Systems and library applications programs reside on a 132 M character CDC 6638 disk, which is also available for user temporary files. Every time that a job requires service from the operating system an appropriate message is recorded on the DAYFILE, a log maintained by the SCOPE system. To obtain our data we sampled several copies of the DAYFILE, recording the following information.

1. Job identification: The code used in job identification distinguishes between graduates, undergraduates, and faculty, and between jobs associated with classwork and jobs associated with research projects. The technique of financial control in the system discourages the use of class numbers for research jobs and vice versa.
2. Central processor time used
3. Peripheral processor time used
4. Priority of job at time it is run (0-low priority to 7-high priority)
5. Number of tape drives charged for
6. Charges assigned
7. Whether the job is a FORTRAN or non-FORTRAN
8. Number of lines printed
9. Number of cards read
10. Amount of central memory used, expressed as a percentage of 32 K words.

Three different statistical techniques were used. One was a simple summary of the statistical characteristics of each of the nine measurements of the aggregate sample, obtained by plotting a histogram and preparing tables of measures of the central tendencies and dispersion statistics, using BMD01D program to do this. The correlations between the different measures were computed using the BMD03M program (Dixon, 1965). This program provided correlation matrices and a factor analysis using the variance maximization criterion (Harman, 1960) to define orthogonal factors. Finally, a cluster analysis (Diehr, 1969) was performed to see if the jobs analyzed fell into groups of similar jobs. The cluster analysis algorithm used grouped the observations into a fixed number of groups called *clusters*, such that the sum of squared distances from observations to their cluster means was minimized. The algorithm will be described in detail in a moment.

TABLE I—Descriptive Statistics for 1588 Jobs Submitted to CDC 6400

Measure	Mean	Standard Deviation
Cards read	224	495
Lines printed	760	1260
CPU time (sec.)	11.0	41
PPU time (sec.)	11.9	35
Central memory	55.8	25.4
Tape drives charged	.28	.55
Cost to user	1.44	4.10
Percent jobs using Fortran	.54	.50

DESCRIPTIVE STATISTICS RESULTS

Table I presents descriptive statistics for 1588 jobs selected from first shifts.* We shall discuss this sample extensively. Similar analysis of second shift data and data from a different time of the year produced very similar results. Therefore, virtually all of our remarks will be concerned with an analysis of these jobs.

Whether Table I presents a true or false picture of the user community depends on the purpose for which the examination is conducted. It shows what sort of use is made of computing by the "average" user. This hypothetical individual submits what most people intuitively familiar with the center would consider a medium-sized job, reading about 200 cards, printing 700 to 800 lines (about eight pages plus system output), and using around eleven seconds each of cpu and ppu time. Slightly more than half of the jobs execute a Fortran compilation. Like the man with 2.4 children, the average user is not the typical user! Frequency plots of the variables CARDS READ, LINES PRINTED, CP TIME, PP TIME, and COST showed that the distributions were positively skewed with means in regions of very low density, suggesting that (a) mean values were not good descriptors of the popu-

* In obtaining the 1588 teaching and research jobs we also encountered on DAYFILE logs a record of 364 miscellaneous jobs. These included jobs generated by the computer center itself, administrative work for some reason not done on the B5500, and an occasional commercial user. Because this group of jobs was so heterogeneous it was not further included in the analysis.

lation and (b) that the observations were exponentially distributed. If the second conclusion had been correct, logarithmic transformations of the indicated variables would have produced symmetric distributions. In fact, they did not. This is illustrated in Figure 1, which is a frequency histogram for the *logarithm* of CP time. (The other four variables listed above were similarly distributed, while MEMORY USE was symmetric originally and TAPE DRIVES CHARGED and FORTRAN use are discrete.) Both the mean value of the transformed CP time and the logarithmic value of the mean of the untransformed time are shown. It can be seen that neither figure is an accurate descriptor. The frequency distributions were positively skewed even after the transformation and, in some cases, appeared to be bimodal. This strongly suggests that instead of regarding jobs as being generated by a single process, the jobs should be thought of as being a mixture of two or more populations which *individually* might be satisfactorily characterized by standard descriptors of central tendency and dispersion.

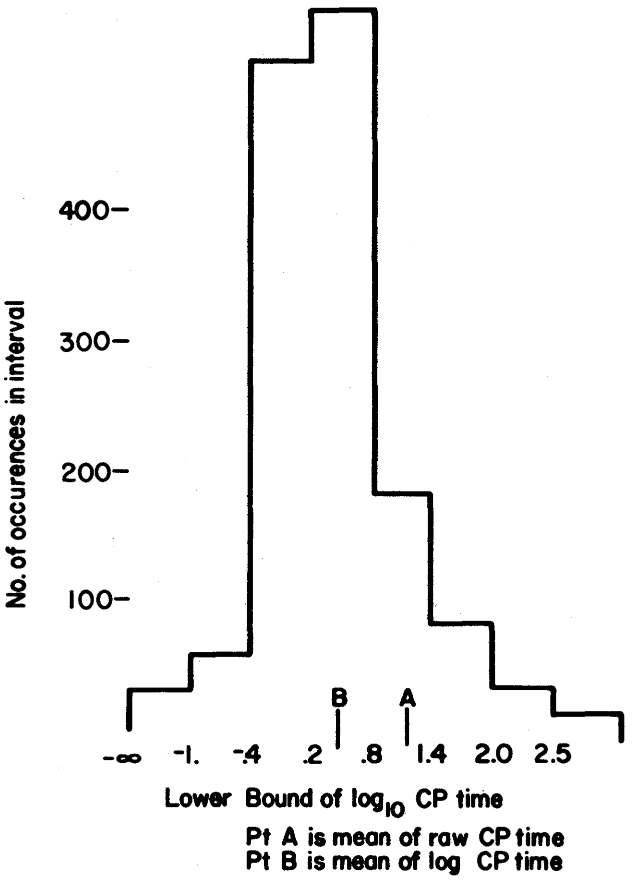


Figure 1—Frequency histogram of $\log_{10}$ CP time

TABLE II—Mean Values of Each Measurement, for Total Sample, Research, and Instructional Job Numbers

Variable	Overall Mean	Research Job Mean	Instructional Job Mean
Cards read	224	490	95
Lines printed	760	1430	442
CPU time	11.0	26	3.8
PPU time	11.9	22	7.1
Central memory	55.8	66.0	51.0
Tape drives	.28	.4	.22
Cost	1.44	3.40	.48
Percent jobs using Fortran	.54	.73	.44
Priority of run	.016	.04	.004
Number of jobs	1588	527	1061

To investigate this hypothesis, we first divided the sample into two groups, jobs associated with research projects and jobs associated with instruction. It was immediately clear that this was, indeed, a reasonable distinction. Table II shows the means on each measure for the sample as a whole and by subgroups. On the average, the difference between subgroup *means* exceeds one standard deviation about the sample mean, thus clearly supporting the hypothesis that there are two distinct subgroups.

One is tempted to say, "Of course, why bother to measure such an obvious thing?" We would expect to find differences between instructional and research work, although our intuition is not very good at predicting the fine detail of these differences. We also found, however, that this simple division is not enough—averages do not describe the typical research or instructional job either! Examination of the histograms within classes based on the research-instruction distinction again showed distributions similar to Figure 1. We therefore eschewed our intuition and turned to an "automatic" method of dividing jobs into homogeneous groups, using cluster analysis.

CLUSTER ANALYSIS RESULTS

The purpose of a cluster analysis is to group observations into k subclasses such that, in some sense, the differences between members of the same class is small relative to the differences between members of different classes. The particular cluster analysis technique we used regards each observation as a point in n dimensional Euclidean space. Observations are assigned to a predetermined number of groups (*clusters*) in such a way that the sum of squares of the distances of points to their cluster mean point is minimized. Thus the cluster analysis is bound to produce groups for which

TABLE III—Mean Values for Measures—Two Clusters Compared to Research and Instructional Jobs

Variable	Group			
	Cluster 1	Instruction	Cluster 2	Research
Log cards read	3.9	3.8	5.4	5.6
Log lines printed	5.6	5.5	6.3	6.5
Log CP time	—.09	—.5	1.9	2.3
Log PP time	1.6	1.5	2.4	2.5
Memory use	51	48	67	70
Tape drives	.22	.19	.40	.44
Log cost	−2.0	−2.1	.53	.50
Percent Fortran use	.44	.44	.74	.72

central tendency measures are reasonable descriptors, while the standard deviation within a cluster indicates how much variation there is about the mean point. The algorithm begins with all observations in a single cluster around the population mean point. A second cluster is initiated whose first member is the observation furthest away from the center point. An iteration phase follows, in which each observation is assigned to one or the other cluster by making the choice which minimizes the sum of squares about cluster points. The cluster mean point is adjusted as the observation is grouped. The iteration is continued until no further adjustments are made. A new cluster is then initiated by choosing as its first member that observation which is furthest from the mean point of the cluster to which it is now assigned. The iteration is then repeated. The entire process is continued until the predetermined number of clusters is obtained.

While a stable partition represents a local minimum by the sum of squares criterion, it is not necessarily a global minimum. Extensive experimentation with this algorithm in comparison to several other clustering methods has indicated that it consistently finds good clusters (Diehr, 1969). Our only reservation is that because a minimum variance criterion is being used, one wants to avoid situations in which the means and variances of the partitions are correlated. Fortunately, this can be achieved by using logarithmic transformations of highly skewed variables (in this case CARDS READ, LINES PRINTED, CP TIME, PP TIME, and COST). Accordingly these variables were included after a logarithmic transformation. The variables MEMORY USE, TAPE USE, and FORTRAN USE were included but not transformed.

If the research-instructional distinction is a valid one, then a clustering into two classes should recreate it. This is, indeed, what happens. Table III shows the means and standard deviations for two clusters, compared to the breakdown of jobs by research or instructional sources. Table IV shows a cross classification of jobs both by their origin and the cluster into which they fall. Almost 90 percent of the instructional jobs fall into the first cluster, while about 75 percent of the research jobs fall into the second cluster.

While this confirms our faith in the research-instruction distinction, it still leaves us with too gross an analysis. Clusterings into from two to six groups provided a significant insight into the data. Let us describe the results of these successive clusterings briefly.

Three groups: The data was partitioned into small, medium, and large resource use groups. The small job group is largely classwork jobs, the large usage group largely research jobs, and the medium usage group made up of half research-half classwork jobs. There is no indication of sub-populations which have heavy I/O use but light processor use (i.e., no "scientific business" breakdown).

Four groups: The data was partitioned into two groups of jobs with small resource use; differentiated only by use or non-use of the FORTRAN compiler. The other two groups were jobs with medium to large system resource use and "aborted" jobs. The medium-large usage group is similar to the medium-large usage group found for two clusters. The group of aborted jobs tends to be small in terms of I/O requirements, and had virtually no CP use.

Five groups: This clustering separated a group of large jobs using tape drives from the four groups described above.

Six groups: This is perhaps the most interesting clustering. Two levels of system resource use were uncovered, with three types of jobs within each level. There were three types of small jobs; 408 FORTRAN and 472 non-FORTRAN jobs, and 89 aborted jobs. The small job groups were primarily instructional, and included jobs using a BASIC interpreter. The aborted jobs were almost all terminated due to control card errors. It is interesting to note that such errors apparently occur on about 5 percent of the jobs submitted.

The medium to large job groups included 181 medium-sized jobs using tape drives, 293 medium to large jobs which did not use tapes, and 148 very large jobs.

TABLE IV—Cross Classification of Jobs by Cluster and Administrative Source

		Administrative Source	
		Instruction	Research
Cluster	1	884	144
	2	187	373

We feel that the most interesting contrasts are between (a) the population statistics, (b) the statistics for the two-cluster (research-instruction) partition, and (c) the finer data of the six group clustering. Figure 2 is a graphic summary of what one sees if jobs are regarded as coming from one, two, or six populations. In this figure each cluster is represented as a rectangle. The following information is coded in the figure:

1. The area of the rectangle drawn for the group is proportional to the number of jobs within it.
2. The shading indicates the number of research jobs—i.e., a completely shaded rectangle would represent a group containing only research jobs, while an unshaded rectangle would represent a group of instructional jobs.
3. The horizontal axis shows the average number of standard deviations between a group mean and the population mean on each of the resource variables. Thus the "partition" consisting of all 1588 jobs has its rectangle centered at 0.0 on the horizontal axis, while clusters containing large resource use jobs are centered to the right of this point, and those containing small jobs are centered to the left.
4. The vertical axis indicates the number of groups (1, 2, and 6) on which the partition is based and, within the region for a given number of groups, the fraction of FORTRAN jobs. Thus one can determine that the 1028 "small" jobs in the two groups clustering contained approximately 45 percent FORTRAN jobs, while the "medium-large" jobs were 75 percent FORTRAN by examining the vertical position of the appropriate rectangles.

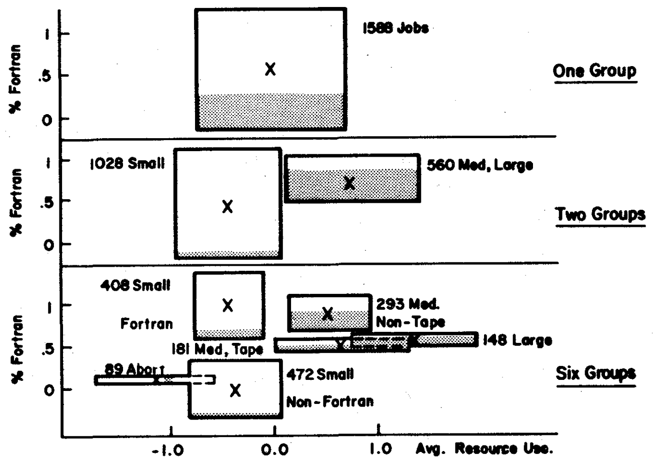


Figure 2—Graphic summary of six cluster result—see text for explanation of code

TABLE V—Correlations Between Variables Based on 1588 Cases

Variable	1	2	3	4	5	6	7
1. Log cards read	1.00	.42	.51	.39	.36	.02	.62
2. Log lines printed		1.00	.46	.39	.22	.10	.50
3. Log CP time			1.00	.53	.46	.16	.75
4. Log PP time				1.00	.18	.41	.71
5. Memory use					1.00	.10	.43
6. Tape drives						1.00	.31
7. Log cost							1.00

5. The length of the rectangle indicates the average variation on the system use variables, with 0.8 std. dev. used as a basis. Thus, it is evident that for six groups the "small-non-FORTRAN" jobs had a slightly greater variation on the average than the "small-FORTRAN" jobs. The length of the rectangles also shows that the "med-non-tape" jobs are better defined than either the "med-tape" jobs or the "large" jobs.

CORRELATION ANALYSIS

The cluster and descriptive analyses dealt with the relations between jobs. Another way to analyze our statistics is to look at the relationship between variables. The table of correlation coefficients for all variables was computed and factor analyzed. The analysis was performed separately for the different classes of user and for all cases together. Since there was no substantial difference in either the correlation or factor matrixes, only the overall picture will be discussed.

Before performing the correlation analysis a certain amount of data editing was done. The distinction between FORTRAN and non-FORTRAN jobs and the priority measures were dropped, and a logarithmic transformation was performed on all other variables. The logarithmic transformation was used because all variables were either exponentially distributed or had a number of cases with extreme values. High or low correlation coefficients based on untransformed data, then, might be produced by only a few cases. The use of the logarithmic transformation greatly reduces the chance of this occurring.

The correlations between the variables are shown in Table V. The table of untransformed variables presents substantially the same appearance except that the extreme values are somewhat higher. The picture of correlations is not immediately clear. It becomes so, however, when one looks at Table VI, which shows the

TABLE VI—Factor Loadings for Variables on First 3 Factors

	Factor		
Variable	1	2	3
Log cards read	.72	.35	.12
Log lines printed	.65	.14	.45
Log CP time	.84	.13	−.07
Log PP time	.76	−.40	.18
Memory use	.55	.34	−.71
Tape drives	.35	−.83	−.26
Log cost	.92	−.05	.02
Cumulative % variance	50	66	77

factor loadings for each of the variables on each possible factor*. Only the first three factors will be discussed. These account for better than 75 percent of the total variance. The first, and by far the largest (50 percent) factor can be thought of as a "standard job" factor. It accounts for half or more of the variance in cards read, lines printed, and central and peripheral processor time. Our interpretation is that this factor is produced by the correlated variation in the measures used by most jobs. The second factor is essentially a "tape request" factor (note the high loading of "requests"), and reflects a difference between jobs that do or do not use tapes. The third factor has its heaviest loading on memory use. It reflects variation in memory use by some jobs that lie outside of the normal spectrum of computing (i.e., outside the range covered by factor 1). This is probably caused by (a) jobs that have control card errors and hence use little memory and (b) a few research jobs that utilize memory heavily.

* Since factor analysis may not be familiar to all readers, we shall explain a way to interpret its results. For further details see Harman (1960) or Morrison (1965).

Suppose each job were plotted as a point in 7 dimensional space. Since most measures are exponential, conversion to a logarithmic scale ensures that the swarm of points will be roughly a hyperellipse. The factors can be thought of as the axes of the hyperellipse. The first factor is the major axis, the second factor the next longest axis, etc. The 'percent variance extracted' by each factor is the percent of variance in distances from the centroid of the ellipse associated with projections on the factor in question. The loading of a variable on a factor can be interpreted in the following way. Suppose each point is plotted on a chart of variables against factor. Note that these will not generally be orthogonal axes. The square of the loading of the variable on the factor is the fraction of variance in the variable associated with variance in the factor. Alternately, the loading can be thought of as the correlation between the variable and a hypothetical pure test of the factor.

In general, the factor analysis supports the other statistics we have gathered. An interesting point is the low loading for memory use on factor 1, which indicates that most jobs have a uniform memory requirement. This could be quite important in designing memory allocation algorithms in multi-programming systems.

SPECIFIC QUESTIONS

The statistical analysis raised a number of non-statistical questions about jobs, and particularly about jobs that were not typical of their administrative category. To answer these a special cross tabulation program was written, modeled after a more extensive information retrieval system designed by Finke (1970) and used to sort jobs in various ways. Some of the specific questions and their answers were as follows:

Q.1. How does the use of FORTRAN or BASIC affect instructional jobs?

A. BASIC jobs use less processor time than FORTRAN but, on the average, much more memory than the average for *instructional* jobs. Research jobs virtually never use BASIC except for relatively small jobs.

Q.2. What percent of memory is used by the "average" job?

A. Better than half the teaching jobs use less than 16K words. The comparable "break even" point for research jobs is 24K. Twelve percent of the research jobs use more than 32K words, while less than two percent of instructional jobs do. Furthermore, most of the long instructional jobs are generated by a few individuals (i.e., are multiple jobs with the same user I.D.).

Q.3. How many runs are compiler runs of any sort? What compilers were used?

A. About two-thirds of the jobs call for at least one compilation. In 1588 runs the FORTRAN compiler was called 849 times, BASIC 249 times, SNOBOL once, SIMSCRIPT 13 times, the COMPASS assembler <u>twice</u> (by the same job number) and COBOL and ALGOL never. (Excellent COBOL and ALGOL systems are available on the University's B5500, so this may be misleading.) Only three center supported "packages" were used: the BMD statistical programs, the SMIS package, and a SORT-MERGE system, for a total of 69 runs. One wonders two things: how much effort are users devoting to duplicating library programs and how much effort should a computer center devote to maintaining such programs?

CONCLUSIONS

Our specific conclusions are that the University of Washington Computer Center users create jobs that fall into four groups, some with important subgroups, producing the six groups graphed in Figure 2. The four major groups are aborted jobs, small jobs (with subgroups FORTRAN and non-FORTRAN), medium-sized jobs (with subgroups tape and non-tape jobs), and large jobs. Small jobs are primarily due to classroom work, middle and large jobs are associated with research work. The principal ways in which jobs differ from each other is in the amount of processor time used and the amount of input. These statistics, which are not terribly startling, are of direct use to the University of Washington and are of indirect use to any institution which is willing to assume it is like Washington.

Should our analysis be used generally even though our particular results are not general? To answer this, we will point out two courses of action which are available to the University of Washington now that it has these statistics, but which might not have been available (or at least, would have been available only by trusting the Computer Center Director's intuition) without the analysis.

At most universities computer use of education is supported by intramural funds, while a substantial part of the research computing support is extramural. Understandably, granting agencies (notably the United States Government) insist that the same algorithm be used to allot charges to all users. The argument is that the cost of a computation should be determined by the nature of the computation and not by who makes it. While seemingly fair, this can be frustrating to an institution which wishes to encourage educational use of computing, but needs to capture all funds that are available for research computing. More generally, there are a number of situations in which a computer center may wish to encourage or discourage certain classes of user, while still retaining the principle that the same charge will be levied for the same service. The solution proposed is to establish a charging algorithm which is sensitive to the varying characteristics of jobs from different user sources. For example, if the University of Washington were to place a very low charge for the first 200 cards read and the first 10 seconds each of CP and PP time, and charge considerably for system utilization beyond these limits, then the educational users would pay proportionately less and the research users proportionately more of the total bill. Charges would still be non-discriminatory in the sense that identical jobs receive identical bills. Note also, how our statistical analysis dictates the type of charging algorithm. From the correlational analysis we know that the only way of differentially effecting user charges is to manipulate the number of cards read and the processor time charges. From the descriptive statistics and the cluster analysis we can predict how a given manipulation will affect different sections of the user community.

Very much the same reasoning can be used in planning for new equipment acquisition. Obviously equipment additions aid in computing either because they facilitate the running of all jobs equally (in which case the aim is to increase throughput uniformly) or because they aid in processing of certain types of jobs. The computer center director rightly looks at equipment in terms of how it affects bottlenecks in his throughput or in his capability to do certain types of computation. From the University administrators' view, however money into the computing center is a means toward the end of achieving some educational or scholarly goal, such as increased production of engineering B.S.'s or support of a Geophysics research program. We can use a statistical analysis of user characteristics to reconcile these points of view. Taking an obvious example from our data, if the University of Washington decides to put x dollars into support of computing for education, the money should not be spent buying tape drives. To take a more subtle case, suppose we were faced with a choice of obtaining a medium-sized computer or expanding the CDC 6400 system to a CDC 6500 or CDC 6600 computer. The appropriate course of action might be determined by the purpose for which the money is intended, to facilitate educational or research use. Without these statistics, we do not see how the management goals of the institution and the technical goals of the Computer Center can be coordinated.

Our results also are of interest to two groups of people outside of our own institution; those interested in research on computing systems and those involved in selling computers to universities. We feel that we have clearly shown that a simple model of a single process for generating statistical characteristics of user jobs—such as those assumed to estimate the performance of system algorithms—is not appropriate. A model of a university computing community must be based on sub-models for quite different populations. The business-scientific distinction is decidedly not appropriate.

We close with some remarks on the methods we have used. Two of our techniques, descriptive statistics and correlational analysis, are conventional statistical methods. Indeed, the programs we used, BMD03M and BMD01D, are part of the most widely supported applications package in programming! There is no reason why everyone with a computer of any size could not perform these analyses on his job stream. We feel, however, that the clearest picture of our users was obtained by the less conventional cluster analysis. We recommend that this technique be used more widely to analyze computer use. We hope it will aid in identifying the characteristics of existential, rather than postulated, computer users.

REFERENCES

1 W DIXON ed
Biomedical computer programs
Health Sciences Computing Facility UCLA 1965

2 H HARMAN
Modern factor analysis
U Chicago Press 1960

3 D F Morrison
Multivariate Statistical Methods. New York: McGraw-Hill, 1965

4 G DIEHR
An investigation of computational algorithms for aggregation problems
Western Management Science Institute UCLA Working Paper 155 1969

5 J FINKE
A users guide to the operation of an information storage and retrieval system on the B5500 Computer
Technical Report No. 70–1–3
Computer Science Univ of Washington

An initial operational problem oriented medical record system—For storage, manipulation and retrieval of medical data*

by JAN R. SCHULTZ, STEPHEN V. CANTRILL and KEITH G. MORGAN

University of Vermont
Burlington, Vermont

INTRODUCTION

The ultimate role of the computer in the delivery of health services has yet to be defined. There may be profound implications in terms of quality of medical care, efficiency, economics of care, and medical research. Final judgments as to advisability and economic feasibility await the implementation of prototype total medical information systems and further technical developments directed toward lowering the high cost of currently developing systems. Development of less expensive hardware and real-time application of the present hardware and software must go on in parallel. We have been involved in the latter, and an experimental time-shared medical information system has been developed for storing and retrieving the total medical record, including both the narrative and the numeric data. This development has integrated the Problem Oriented Medical Record, a means of organizing medical data around a patient's problems, with a touch sensitive cathode ray tube terminal that allows structured input (with additional keyboard entry capability) by directly interfaced medical users (in particular the physician and the nurse).

A total of 85 general medical patient records have been kept on the system as of December, 1970. The system handles all aspects of medical record keeping—from the Past Medical History and Systems Review collected directly from the patient to complete Progress Notes and flowsheets, all recorded in a problem-oriented manner. It allows the direct inputting of data by the information originator and the retrieval of data in various medically relevant forms.

THE PHILOSOPHICAL BASIS FOR THE SYSTEM

The system is based on a medical philosophy requiring data in the medical record to be *problem oriented* and not *source oriented*.[1,2] Data are collected, filed and identified in a problem oriented record with respect to a given *problem* and not to the *source* of the data as in the traditional source oriented record.

The problem oriented medical record requires a systematic approach to treatment of the patient. This systematic approach is defined by the four phases of medical action: data base collection, problem formulation, plan definition and follow-up. A brief explanation of the four phases will outline the basic requirements for this system (see Figure 1):

```
.....................          .....................
:I. ESTABLISHMENT OF:          :II. FORMULATION OF  :
:   A DATA BASE:    :          :    ALL PROBLEMS:   :
:- - - - - - - - - -:          :- - - - - - - - - - :
: HISTORY.          :  --->    : PROBLEM LIST.      :
: PHYSICAL EXAM.    :          :                    :
: ADM. LAB. WORK.   :          :                    :
:...................:          :....................:
                              /
                             /
.....................       /  .....................
:III. PLANS FOR EACH:<-/       :IV. FOLLOW-UP ON EACH:
:     PROBLEM:      :          :    PROBLEM:        :
:- - - - - - - - - -:          :- - - - - - - - - - :
: COLLECTION OF     :  --->    : PROGRESS NOTES.    :
:    FURTHER DATA.  :          :                    :
: TREATMENT.        :          : TITLED & NUMBERED  :
: EDUCATION OF PAT. :          :  BY PROBLEM.       :
:...................:          :....................
```

Figure 1

* This research was done under a research grant from the Department of Health, Education, and Welfare, Health Services and Mental Health Administration, National Center for Health Services Research and Development, PHS 1 R18 HS 00175-01, entitled "The Automation of a Problem Oriented Medical Record," Co-Principal Investigators Lawrence L. Weed, M. D. and Jan R. Schultz.

During the first phase of medical action the patient's complete data base is collected. This includes a branching questionnaire Past Medical History and Systems Review taken directly by the patient, a Physical Examination entered by the physician and other medical personnel, the Present Illness structured from choices (to be discussed later) and entered by the physician, and certain admission laboratory orders generated by the physician.

After a complete data base is collected the physician studies it and formulates a list of *all* the patient's problems. This is the *second phase* of medical action. The *problem list* includes medical, social, psychiatric and demographic problems. Each problem is defined at the level the physician understands it. A problem can be a "diagnostic entity," a physiologic finding, a physical finding, an abnormal laboratory finding, or a symptom. The problem list is a dynamic index to all the patient's plans and progress notes since it can be used to follow the course of the problem(s).

After a complete problem list is formulated, the physician must define an initial *plan for each problem.* This is the *third phase* of medical action. The plans are divided into plans for more information, plans for treatment and contingency plans. With plans for more information it is possible to: (1) rule out different problems by ordering certain tests or procedures, (2) get laboratory tests for management, and (3) get more data base information. Under plans for treatment: a drug, diet, activity, procedure, or patient education can be prescribed. Contingency plans are possible future plans to be carried out if certain contingencies are satisfied.

The *fourth phase* of medical action is writing *progress notes* for each problem. The progress notes for each problem are divided into: Symptomatic, Objective (laboratory, x-ray and other reports), Treatment Given, Assessment and Follow-up Plans (similar in content to the Initial Plans) sections. The progress notes allow medical personnel to act as a guidance system and follow the course of each problem, collecting more data base, reformulating and updating problems and respecifying the plans, each action dependent upon the course of the patient's problems.

The Problem Oriented Medical Record has been used in paper form for the past fourteen years. It has proved a working record system on paper and was demonstrated practical long before computerization was ever considered. Its dynamic structure, non-source orientation and medically relevant labeling of all data, however, facilitates computerization. Computerization augments its medical capabilities by making it possible to retrieve all data on one problem in sequence and by allowing data to be organized separate from its source in the record. This ability is recognized as having significant medical implications,[3,4] for it allows the physician to follow the course of a problem in parallel with the patient's other problems or as a separate problem (i.e., retrieving information chronologically vs. retrieving all data on one problem). The computer enables rapid audit of all the patients with similar problems as well as the ability to audit a physician's logic and thoroughness on one specific patient, and will allow the development of research files.

As previously written:

> "*We should not assess a physician's effectiveness by the amount of time he spends with patients or the sophistication of his specialized techniques. Rather we should judge him on the completeness and accuracy of the data base he creates as he starts his work, the speed and economy with which he obtains patient data, the adequacy of his formulation of all the problems, the intelligence he demonstrates as he carefully treats and follows each problem, and the total quantity of acceptable care he is able to deliver.*"[1]

Our experience with this system indicates that computerization does facilitate such an assessment.

At the time this project began, and the system was specified, other operating systems (of which we had knowledge) were few.[5,6] After an analysis of these systems our group decided that we would try to build the necessary medical application programs using as a basis system software developed by another group. To quote R. W. Hamming;

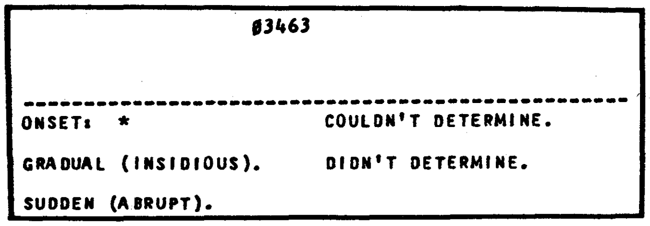
> "*Indeed, one of my major complaints about the computer field is that whereas Newton could say, 'If I have seen a little further than others it is because I have stood on the shoulders of giants,' I am forced to say, 'Today we stand on each other's feet.' Perhaps the central problem we face in all of computer science is how we are to get to the situation where we build on top of the work of others rather than redoing so much of it in a trivially different way. Science is supposed to be cumulative, not almost endless duplication of the same kind of things.*"[7]

This would serve two purposes: It would divide the total work task naturally into more manageable units, and it would force our group to learn completely the system we were to build upon, thus allowing the basic system software to be utilized as a tool in the accomplishment of our medical goals. We built upon the system being developed by Medical Systems Research Laboratory of Control Data Corporation. The hardware developments of Dr. Robert Masters (the Digiscribe) and the software developments of Mr. Harlan Fretheim (the Executive, the Human Interface Program and SETRAN) have been fundamental to our own progress.

THE BASIC SYSTEM SOFTWARE DEVELOPED BY MEDICAL SYSTEMS RESEARCH LABORATORY OF CONTROL DATA CORPORATION

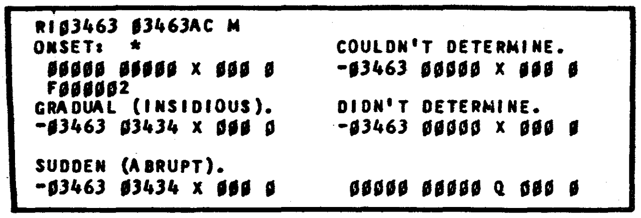
Directly interfacing busy medical personnel to the computer system required the development of an effective facile interface. The Digiscribe terminal with its associated software is such an interface. Displayed on the cathode ray tube is an array of choices from which the user can make a selection by touching the screen with his finger. The user's selection is input to the computer system in the form of a character so that for each of twenty different positions on the cathode ray tube screen a different character is generated. A system program[8] accepts as input the user selection and on the basis of it appropriate branching takes place and new information is displayed. Included in the information used by the program that interprets the selection (the Human Interface Program) is a push-down list of frame numbers waiting to be displayed, branching information and certain internal parameters (not seen by the user at the terminal) which can be associated with each choice displayed on the screen. In addition to text, branching information and internal parameters, a program call[9] can be associated with any selection. This allows a certain amount of open endedness and provides the means for calling and executing application programs.

The selections made by the medical user at the terminal are concatenated by the Human Interface Program to form "paragraphs" of information. The *paragraph* is the basic unit of information generated in the system. (The Storage and Retrieval programs manipulate paragraphs of information.) Associated with each paragraph is the Selection Parameter List which includes for each choice made by the user the frame number, the choice number within that frame, and any internal parameters associated with the choice. The internal parameters can be used to code selections so programs can interpret compact codes rather than alphanumeric data. The internal parameters are identified by a single letter (e.g., "F" type internal parameters specify format codes which will be explained in detail below). Using the Selection Parameter List our pro-



```
                    ø3463


---------------------------------------------------
ONSET:  *                       COULDN'T DETERMINE.

GRADUAL (INSIDIOUS).            DIDN'T DETERMINE.

SUDDEN (ABRUPT).
```

DISPLAY IN 'HIP'



```
RIø3463 ø3463AC M
ONSET:  *                        COULDN'T DETERMINE.
 øøøøø øøøøø X øøø ø             -ø3463 øøøøø X øøø ø
 Føøøøø2
GRADUAL (INSIDIOUS).             DIDN'T DETERMINE.
-ø3463 ø3434 X øøø ø             -ø3463 øøøøø X øøø ø

SUDDEN (ABRUPT).
-ø3463 ø3434 X øøø ø              øøøøø øøøøø Q øøø ø
```

SAME DISPLAY IN 'SETRAN'

Figure 2

grams can analyze the input data without having to search the generated English text.

The user generated paragraphs and associated Selection Parameter Lists, are the coupling mechanism between the Human Interface Program and our application programs which store, retrieve, and manipulate the patient records and other files.

A language was developed as part of the basic system called SETRAN (Selection Element TRANslator)[10] which makes possible the programming of the branching logic displays and alteration of already entered displays using the keyboard on the terminal. See Figure 2 for an example of a frame as displayed in the Human Interface Program and in SETRAN. It has been possible to train physicians and other personnel in this language and development of new displays has proceeded without a computer person acting as an intermediary. Allowing medical personnel, with almost no computer science training a direct means of developing and altering basic system displays is fundamental during the development phase of systems such as these. The massive number of such displays required for such a system (currently over 16,000 displays have been developed by the PROMIS group) and the necessity for a tight feedback-loop during the developmental phase of the displays require such tools. Once systems such as this are beyond the developmental phase, access to such programs must be carefully controlled.

The operating system supports multiple terminals

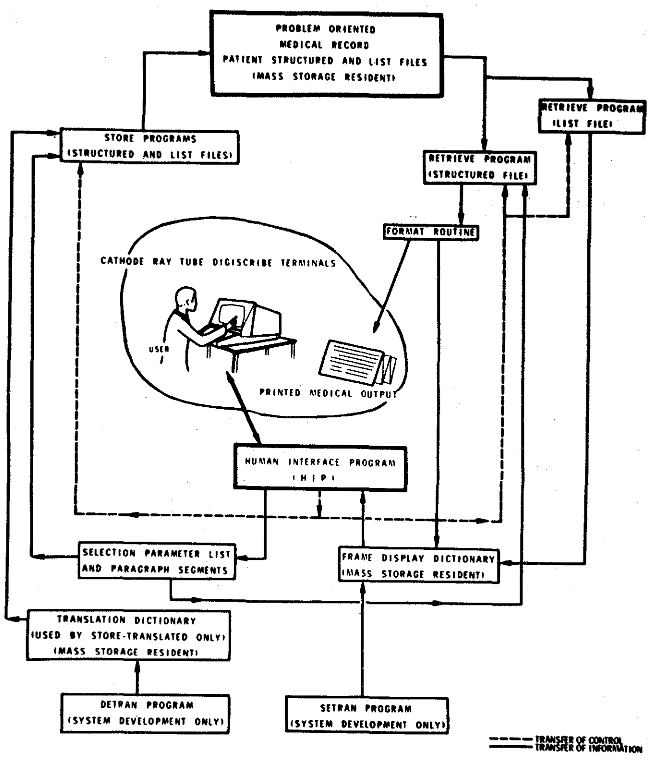

PROBLEM ORIENTED MEDICAL RECORD PATIENT STRUCTURED AND LIST FILES (MASS STORAGE RESIDENT)
STORE PROGRAMS (STRUCTURED AND LIST FILES)
RETRIEVE PROGRAM (LIST FILE)
RETRIEVE PROGRAM (STRUCTURED FILE)
FORMAT ROUTINE
CATHODE RAY TUBE DIGISCRIBE TERMINALS
USER
PRINTED MEDICAL OUTPUT
HUMAN INTERFACE PROGRAM (HIP)
SELECTION PARAMETER LIST AND PARAGRAPH SEGMENTS
FRAME DISPLAY DICTIONARY (MASS STORAGE RESIDENT)
TRANSLATION DICTIONARY (USED BY STORE-TRANSLATED ONLY) (MASS STORAGE RESIDENT)
DETRAN PROGRAM (SYSTEM DEVELOPMENT ONLY)
SETRAN PROGRAM (SYSTEM DEVELOPMENT ONLY)
TRANSFER OF CONTROL
TRANSFER OF INFORMATION

Figure 3—EXPLANATORY LEGEND

The "HUMAN INTERFACE PROGRAM (HIP)" displays medical content from the "FRAME DISPLAY DICTIONARY" to the user in a branching logic fashion. Most of the entries in the "FRAME DISPLAY DICTIONARY" were created by system development personnel at some time in the past by using the "SETRAN PROGRAM."

As the user makes a series of choices at the terminal, "HIP" creates a "SELECTION PARAMETER LIST AND PARAGRAPH SEGMENTS" which represent the history of choices made by the user.

At appropriate times in the series of displays seen by the user, dependent upon pathway, "HIP" calls the "STORE PROGRAMS" and the "RETRIEVE PROGRAMS."

The "STORE PROGRAMS," by processing the "SELECTION PARAMETER LIST AND PARAGRAPH SEGMENTS" for the specific user, can update the mass storage resident "PROBLEM ORIENTED MEDICAL RECORD PATIENT STRUCTURED AND LIST FILES."

If the "STORE TRANSLATED" module of the "STORE PROGRAMS" is used (e.g., for processing patient histories), reference will be made to the "TRANSLATION DICTIONARY" which was created in the past by system development personnel using the "DETRAN PROGRAM."

The "RETRIEVE PROGRAM (STRUCTURED FILE)" processes the user's retrieval request as specified in his associated "SELECTION PARAMETER LIST AND PARAGRAPH SEGMENTS" retrieving any specified part of a patient's medical record.

Output by the "FORMAT ROUTINE" from the "RETRIEVAL PROGRAM (STRUCTURED FILE)" can be either to "PRINTED MEDICAL OUTPUT" (hardcopy) or directly to the user's cathode ray tube terminal via the "FRAME DISPLAY DICTIONARY" and "HIP."

The "RETRIEVE PROGRAM (LIST FILE)" may be called in the process of the user making choices at the terminal. It produces output directly to the user's terminal.

All of the above processes take place unknown to the user but are a direct response to his choices at the terminal in either storing in or retrieving information from the Computerized Problem Oriented Medical Record.

allowing a rapid response to user interaction (a user familiar with the frame content can make selections faster than one per second) and supports application programs operating in a multi-level, multi-programming mode.

The frames, application programs, medical files and station associated variables are disk resident. Most selections made by a user require four disk accesses (in the current version of the system) and the variables associated with a station are core resident only while the selection for that station is being processed.

There are four main classes of application programs. The two highest level classes are interactive with the user and require immediate execution. The Selectible Element Translator is of this type and allows the on-line entering and changing of frame content and branching (by appropriate personnel, not all users).

The second level class of programs is executed while the user is at the terminal, but these programs may take longer to run than the ones executed immediately. An example of this type is the program which retrieves from the patient records to the cathode ray tube terminal.

The lowest level of application programs is executed sequentially by priority level in the background after a user has signed off the terminal. These include the programs which store into the patient records and retrieve patient records to the line printer.

AN APPROACH TO THE COMPUTERIZATION OF THE MEDICAL RECORD

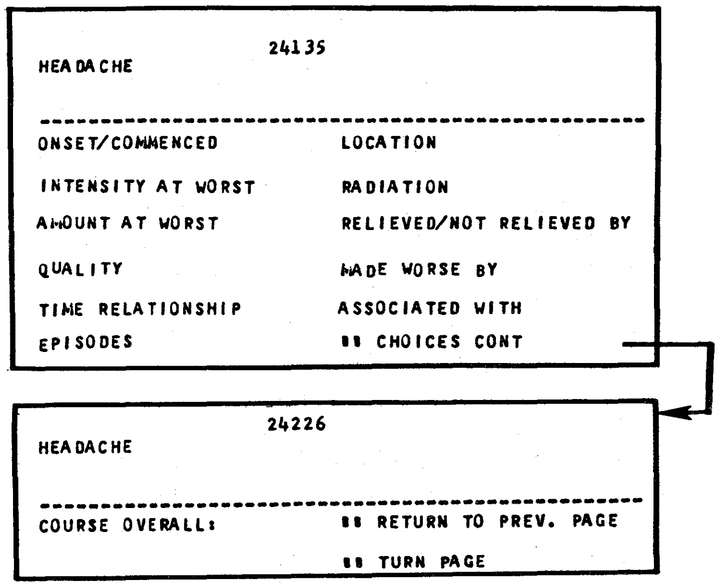
Our approach to the computerization of the medical record involves many elements. The Problem Oriented Medical Record represents the medical "foundation" upon which the total system rests. The Human Interface Program, the Selection Element TRANslator, the system executive and the hardware "drivers" represent the computer software "foundation." The other elements to our approach will be discussed in this section.

The information generated by having the medical personnel (or the patient himself) go through the branching displays is English narrative. Although the number of selections presented on each display is small (averaging 8 selections) the number of paths through these selections is quite large. There are currently over 16,000 displays in the system. Approximately 12,000 of these are branching displays, the remainder are solely for information, e.g., before any drug can be ordered a sequence of displays requests the physician to: (1) CHECK THE PROBLEM LIST FOR: followed by a list of problems (2) SIDE EFFECTS TO WATCH FOR: (3) DRUG AND TEST INTERACTIONS: (4) TEST INTERFERENCE: (5) USUAL DOSAGE: followed by optional (6) MECHANISM OF ACTION: (7) METABOLISM AND EXCRETION:.

Structured branching logic displays allow the medical user to operate from a body of knowledge broader than can be kept in his own memory. This body of knowledge as represented in the library of displays[11,12] is capable of being updated in an organized and systematic way, so that it can always reflect the most current and sophisticated medical thinking. The system is dynamic and since data can be typed into individual patient's records supplementing the structured displays, it would be possible to analyze this typed in data to update the library of displays (if the typed in data indicate a deficiency in the branching displays and not in the physician using the displays). Once systems similar to the one described here are in daily operation, the organized and systematic updating of the library of displays will have to be centralized in an organization with this sole responsibility and authority.

The generation of English is the result of a user making selections from structured, branching-logic displays. These selections must then be transformed by a program into an internal form for storage in a patient's file. The program that does the transformation of the selections into an internal form is generalized; it is independent of the specific content contained in the selections. It stores data in an internal form, independent of the output devices ultimately used to display the data. The retrieval routines that allow the stored data to be manipulated and displayed in various forms will also be discussed in more detail. (See Figure 3).

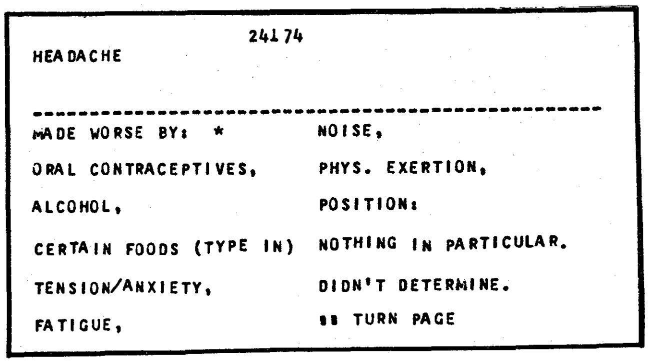
The displays necessary for the generation of each section of the Problem Oriented Medical Record are specified by a "meta-structure" for each section. The "meta-structure" specifies the branching logic of the content displays.[1] There are structured approaches to Present Illness, Problem Lists, Drug Sequences, and Progress Notes. For example, the Present Illness meta-structure would include, for each body system, a list of the symptoms particular to that system and for each symptom, a list of its characteristics. In the Psychiatry system, for example, if Headache were selected from the frame of the list of possible symptoms, the physician would be asked to describe characteristics of this headache:



```
                          24135
HEADACHE

------------------------------------------------------
ONSET/COMMENCED             LOCATION

INTENSITY AT WORST          RADIATION

AMOUNT AT WORST             RELIEVED/NOT RELIEVED BY

QUALITY                     MADE WORSE BY

TIME RELATIONSHIP           ASSOCIATED WITH

EPISODES                    ** CHOICES CONT
```

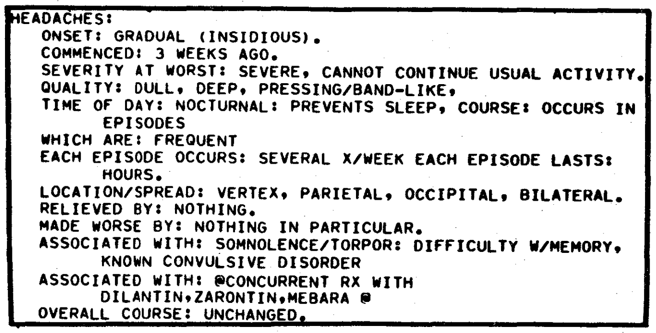
```
                          24226
HEADACHE

------------------------------------------------------
COURSE OVERALL:             ** RETURN TO PREV. PAGE

                            ** TURN PAGE
```

If "MADE WORSE BY" were chosen by the physician, a frame containing the following selections appears:



```
                          24174
HEADACHE

----------------------------------------------------
MADE WORSE BY: *            NOISE,

ORAL CONTRACEPTIVES,        PHYS. EXERTION,

ALCOHOL,                    POSITION:

CERTAIN FOODS (TYPE IN)     NOTHING IN PARTICULAR.

TENSION/ANXIETY,            DIDN'T DETERMINE.

FATIGUE,                    ** TURN PAGE
```

This technique allows a complete English narrative description of HEADACHE to be generated:



```
HEADACHES:
   ONSET: GRADUAL (INSIDIOUS).
   COMMENCED: 3 WEEKS AGO.
   SEVERITY AT WORST: SEVERE, CANNOT CONTINUE USUAL ACTIVITY.
   QUALITY: DULL, DEEP, PRESSING/BAND-LIKE,
   TIME OF DAY: NOCTURNAL: PREVENTS SLEEP, COURSE: OCCURS IN
         EPISODES
   WHICH ARE: FREQUENT
   EACH EPISODE OCCURS: SEVERAL X/WEEK EACH EPISODE LASTS:
         HOURS.
   LOCATION/SPREAD: VERTEX, PARIETAL, OCCIPITAL, BILATERAL.
   RELIEVED BY: NOTHING.
   MADE WORSE BY: NOTHING IN PARTICULAR.
   ASSOCIATED WITH: SOMNOLENCE/TORPOR: DIFFICULTY W/MEMORY,
         KNOWN CONVULSIVE DISORDER
   ASSOCIATED WITH: @CONCURRENT RX WITH
         DILANTIN,ZARONTIN,MEBARA @
   OVERALL COURSE: UNCHANGED.
```

All programs that interpret input data can assume a standard form and structure guaranteed by how the branching logic displays are programmed. Contained in the Selection Parameter List for each paragraph are the internal parameters which define the type of data and what the program should do with it. The programs also receive information which further describes the paragraph as a unit. This information was defined by selections as the user went through the displays. Associated with each paragraph is a paragraph label which further describes the paragraph on two levels: Information Type 1 (IT1) defines the major section in a record to which this paragraph belongs (e.g., Physical Examination, Progress Notes, Problem List, Past Medical History and Systems Review, etc.); Information Type 2 (IT2) defines the subsection within the major section (e.g., Skin Examination in the Physical Examination, Symptomatically in the Progress Notes, etc.). All paragraphs which contain problem oriented data have associated with the label the applicable problem number. Also associated with the paragraph label is information indicating whether this paragraph contains either narrative or numeric information. The date, time, user and patient numbers are also associated. This data determines where the paragraph is to be stored in a specific patient's record.

Structured displays and the internal parameters linked to selections (and collected in the Selection Parameter List) both imply a closed system. The closed system enforces the organized entry of information in a well-defined syntax. Via the structured displays, the user is aware of data relationships that normally are imbedded in the data interpreting programs. The data is so entered that it has an inherent structure—not to be found when data is entered free form. This structure holds even if the data entered via selections from the structured displays are supplemented by typed-in information. Much simpler data interpreting programs are required for such structured data with the associated Selection Parameter List then for purely free formed input.

In addition, the user of structured displays can operate on recognition rather than recall. He has available, at the time he needs it, the organized knowledge of his profession. This knowledge can be systematically updated with a thoroughness that is impossible on an individual basis.

In summary, the necessary elements in our approach to a computerized system to store and retrieve Problem Oriented Medical Records include a medically relevant organization of the data in the medical record, an effective interface between the medical user and the computer system, a means of structuring the medical content material on frames using meta-structures, programs to transform selections into a manipulatable internal form, programs to retrieve the stored data in various forms, and a "closed" system.

FILE STRUCTURES FOR MEDICAL RECORDS

Our files may be characterized in terms of (1) maintenance and (2) structure. In terms of maintenance: Will this file handle information that can be both inserted and deleted (purgable) or only inserted (non-purgable)? An example of a file into which information will only be inserted and never purged is the patient's problem-oriented medical record. A list of all the patients on a ward is an example of a file that will be added to and deleted from on a regular basis as patients are admitted and discharged from the ward. In terms of structure: We define a file of homogeneous elements (a file which contains a specific subset of the IT1, IT2's) as a list file; e.g., the list of patients on a specific ward, or the list of problems on a specific patient, or the list of current drug/diet/activities for a specific patient. A file of heterogeneous elements (all IT1, IT2's) we call a structured file, for example, the patient's Problem Oriented Medical Record.

There are then a total of four types of files that we envision possible within the system: (1) a list file that cannot be purged; files of this type will ultimately be used for research retrieval capabilities; (2) a list file that can be purged; (3) a structured file that cannot be purged, and (4) a structured file that can be purged and which may in the future be used for the most current progress note. This progress note could be purgable since it might be possible, for example, to condense many vital sign values to one value and a range. For the current implementation of the system it has been necessary to utilize list files that are purgable and structured files that cannot be purged. Further expansion and sophistication of the current system will require the other two types of files to be developed and utilized. The two file types currently used will be described in more detail.

NON-PURGABLE STRUCTURED FILES—THE PATIENT'S PROBLEM ORIENTED MEDICAL RECORD

An individual patient's file requires a structure that facilitates the storage and retrieval of its data while minimizing the number of mass storage accesses. This file (a non-purgable file with heterogeneous elements) consists of a "Table of Contents" and a variable number of "Items." The Table of Contents is an index to all the data in an individual patient's file and is addressable as a function of the patient's system I.D. number. The Table of Contents contains a variable length list of Item Pointers (see Figure 4). Each Item Pointer includes Information Type 1, the storage date and time, and the Item number containing the paragraph (5). If the Information Type 1 is problem oriented then there is also an array of bits that represent the presence of that problem number in the Item. This feature allows the rapid sequential retrieval of all the data on one problem by indicating whether the Item contains paragraphs on the problem number.

The Item is the depository of the narrative (e.g., Present Illness) and numeric (e.g., Blood Pressure) paragraphs. It contains the paragraphs generated by the user at the terminal and transformed by the STORE program into the internal form. (A description of this internal form is given later in this paper.) For each paragraph in the Item there is a corresponding *paragraph pointer* also in the Item. This paragraph pointer contains Information Type 1 and 2, problem number, date, time, user number and the relative address of the paragraph in this Item.

To access any specific paragraph of information in an Item a search through the paragraph pointers, which are sorted and linked together, is required. No searching of actual narrative data is necessary for the retrieval of any narrative paragraph contained in an individual Item, as the paragraph pointers completely define the contents of each narrative paragraph at the level the medical user needs to differentiate the data.

This type of file structure allows the retrieval of specific paragraphs of data in a minimum number of mass storage accesses (two accesses), if we assume the Table of Contents does not overflow. Since the Table of Contents (core resident in one access) contains Item Pointers for each Information Type in the patient's file, searching it gives the Item number(s) of the requested data. The necessary Item(s) can then be brought into core in one access per Item.

INTERNAL REPRESENTATION OF NARRATIVE AND NUMERIC DATA WITHIN THE PATIENT'S PROBLEM-ORIENTED MEDICAL RECORD

Each paragraph generated by the user to be stored into a patient's record can contain either narrative or numeric information. The STORE program and appropriate subroutines are used to transform each paragraph into a standard internal form which consists of

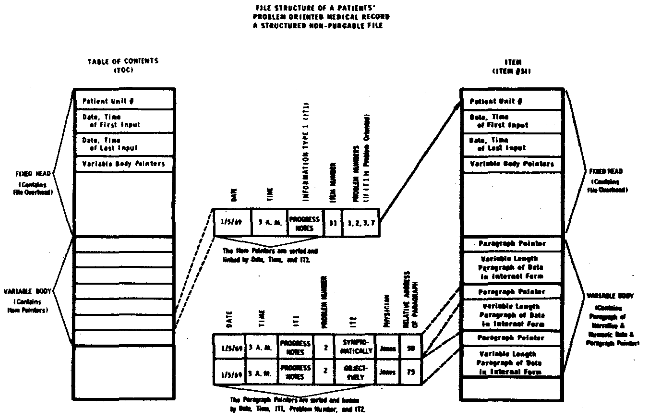

FILE STRUCTURE OF A PATIENTS' PROBLEM ORIENTED MEDICAL RECORD A STRUCTURED NON-PURGABLE FILE
TABLE OF CONTENTS (TOC)
ITEM (ITEM #31)
Patient Unit #
Date, Time of First Input
Date, Time of Last Input
Variable Body Pointers
FIXED HEAD (Contains File Overhead)
VARIABLE BODY (Contains Item Pointers)
DATE
TIME
INFORMATION TYPE 1 (IT1)
ITEM NUMBER
PROBLEM NUMBERS (If IT1 is Problem Oriented)
1/5/69
3 A.M.
PROGRESS NOTES
31
1, 2, 3, 7
The Item Pointers are sorted and linked by Date, Time, and IT2.
IT1
PROBLEM NUMBER
IT2
PHYSICIAN
RELATIVE ADDRESS OF PARAGRAPH
2
SYMPTOMATICALLY
OBJECTIVELY
Jones
50
75
The Paragraph Pointers are sorted and linked by Date, Time, IT1, Problem Number, and IT2.
Paragraph Pointer
Variable Length Paragraph of Data in Internal Form
VARIABLE BODY (Contains Paragraph of Narrative & Numeric Data & Paragraph Pointers)

Figure 4

strings of eight bit characters. The paragraph, for narrative data, represents the smallest unit of information the physician or other medical personnel can ever request on an individual patient. For numeric data, the numeric block—which specifies one numeric value—is necessary, as the physician requires time ordered graphs of various physiologic parameters and thus the retrieval of individual numeric blocks is required. It is possible to store a variable number of numeric blocks in one paragraph. In both cases the data are accessible at the level the medical personnel are most accustomed to working with it. This internal representation allows the rearrangement of data for various retrieval requirements that are impossible with a manual paper record system.

If the paragraph contains narrative data, a STORE routine interprets "F" type internal parameters in the Selection Parameter List as format codes. These format codes are associated with certain selections. They are used to define the internal form in which the data will be stored. This internal form in turn defines the output format of the paragraph and specifies the relationship between the selection with the "F" internal parameter and those selections following. The selection with the "F" internal parameter is treated as a "title" and the selections until the next "F" internal parameter are "data" (See diagram below). Specifically, the format code defines the indentation level of the title (level 0 is the least indentation, level 3 is the greatest amount) and whether there are carriage returns before and/or after the title. With this information it is possible to output the narrative using an interpretive output subroutine called FORMAT. The format codes and the internal form are output device independent.

Since SETRAN (Selection Element TRANslator) allows internal parameters to be associated with any choice, the individual who writes the frame can specify or change the output format. Such flexibility is an important feature in a system like ours as it allows the individual writing the frame content material to specify, at the same time, the output format of the information. Changes of output format are also greatly facilitated as they only require a rewriting, using SETRAN, of the "F" internal parameters on the frame. (Any information previously generated from these frames is not reformatted.)

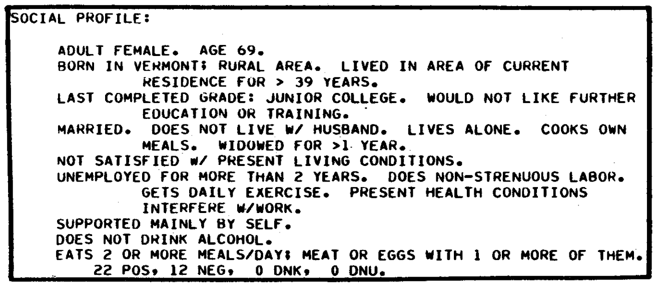
Internally a paragraph of narrative data is of the following form:

```
(Format Code) (Title Narrative) (Data Narrative)...
(Format Code) (Title Narrative) (Data Narrative)...
(Format Code) (Title Narrative) (Data Narrative)
(Terminating Format Code)
```

For example:

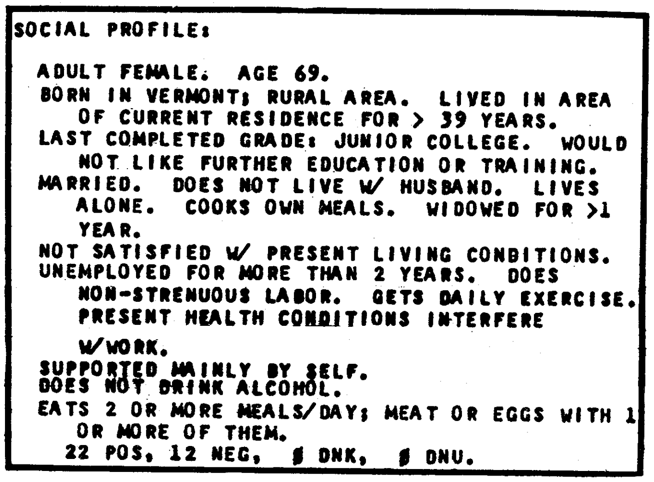
```
(CR, LEVEL 0, CR) (SOCIAL PROFILE:)
(    LEVEL 1    ) (ADULT FEMALE. AGE 69.)
(    LEVEL 1    ) (BORN IN VERMONT;
                     RURAL AREA. LIVED IN
                     AREA OF CURRENT
                     RESIDENCE FOR >39
                     YEARS.)
(    LEVEL 1    ) (LAST COMPLETED
                     GRADE: JUNIOR
                     COLLEGE. WOULD NOT
                     LIKE FURTHER
                     EDUCATION OR
                     TRAINING.)
(    LEVEL 1    ) (MARRIED. DOES NOT
                     LIVE W/HUSBAND.
                     LIVES ALONE. COOKS
                     OWN MEALS. WIDOWED
                     FOR >1 YEAR.)
(    LEVEL 1    ) (NOT SATISFIED
                     W/PRESENT LIVING
                     CONDITIONS.)
(    LEVEL 1    ) (UNEMPLOYED FOR
                     MORE THAN 2 YEARS.
                     DOES NON-STRENUOUS
                     LABOR. GETS DAILY
                     EXERCISE. PRESENT
                     HEALTH CONDITIONS
                     INTERFERE W/WORK.)
(    LEVEL 1    ) (SUPPORTED MAINLY BY
                     SELF.)
(    LEVEL 1    ) (DOES NOT DRINK
                     ALCOHOL.)
(    LEVEL 1    ) (EATS 2 OR MORE
                     MEALS/DAY; MEAT OR
                     EGGS WITH 1 OR MORE
                     OF THEM.)
(    LEVEL 2    ) (22 POS, 12 NEG, 0 DNK,
                     0 DNU.)
(TERMINATION CODE)
```

A printout on the line printer of this paragraph would result in the following:



```
SOCIAL PROFILE:

     ADULT FEMALE.  AGE 69.
     BORN IN VERMONT; RURAL AREA.  LIVED IN AREA OF CURRENT
            RESIDENCE FOR > 39 YEARS.
     LAST COMPLETED GRADE: JUNIOR COLLEGE.  WOULD NOT LIKE FURTHER
            EDUCATION OR TRAINING.
     MARRIED.  DOES NOT LIVE W/ HUSBAND.  LIVES ALONE.  COOKS OWN
            MEALS.  WIDOWED FOR >1 YEAR.
     NOT SATISFIED W/ PRESENT LIVING CONDITIONS.
     UNEMPLOYED FOR MORE THAN 2 YEARS.  DOES NON-STRENUOUS LABOR.
            GETS DAILY EXERCISE.  PRESENT HEALTH CONDITIONS
            INTERFERE W/WORK.
     SUPPORTED MAINLY BY SELF.
     DOES NOT DRINK ALCOHOL.
     EATS 2 OR MORE MEALS/DAY; MEAT OR EGGS WITH 1 OR MORE OF THEM.
          22 POS, 12 NEG,  0 DNK,  0 DNU.
```

On the cathode ray tube it appears as:



```
SOCIAL PROFILE:

  ADULT FEMALE.  AGE 69.
  BORN IN VERMONT; RURAL AREA.  LIVED IN AREA
     OF CURRENT RESIDENCE FOR > 39 YEARS.
  LAST COMPLETED GRADE: JUNIOR COLLEGE.  WOULD
     NOT LIKE FURTHER EDUCATION OR TRAINING.
  MARRIED.  DOES NOT LIVE W/ HUSBAND.  LIVES
     ALONE.  COOKS OWN MEALS.  WIDOWED FOR >1
     YEAR.
  NOT SATISFIED W/ PRESENT LIVING CONDITIONS.
  UNEMPLOYED FOR MORE THAN 2 YEARS.  DOES
     NON-STRENUOUS LABOR.  GETS DAILY EXERCISE.
     PRESENT HEALTH CONDITIONS INTERFERE
     W/WORK.
  SUPPORTED MAINLY BY SELF.
  DOES NOT DRINK ALCOHOL.
  EATS 2 OR MORE MEALS/DAY; MEAT OR EGGS WITH 1
     OR MORE OF THEM.
    22 POS, 12 NEG,  Ø DNK,  Ø DNU.
```

The title and data narrative are both of variable length.

If the paragraph contains numeric data, then the STORE-NUMERIC routine interprets "N" type internal parameters in the Selection Parameter List as numeric codes. The numeric codes associate specific medical data with the internal structure of the numeric blocks. The numeric blocks each contain numeric values or objective text; this could represent a blood pressure, a clinical chemistry value, or any other type of objective information that must be manipulated internally in the system. The numeric codes are used to associate a *type code* (i.e., a number that represents the type of data contained in the block and is the means of identifying all numeric data within the system), a time, a date, a title, a numeric value, and objective text with the numeric block structure. For example, a temperature numeric block could contain: type code = 30; time = 14:35; date = Feb. 23, 1970; title = TEMP; numeric value = 38; number descriptor = C. Medical personnel writing frames can define and change the "N" type internal parameters using SETRAN. (Previously stored numeric blocks are not affected.)

The overall philosophy upon which the system was built required enough flexibility to allow the medical user,* after a minimal training period on the system, to change all system variables specifying the output format (and thus the internal form of the data) and those internal parameters that depend upon medical knowledge of various physiological parameters. Medical personnel associated with our project can directly develop many of the system content frames and can change the specification of how the data will be stored internally by changing internal parameters entered with the frame content material. Such changes require no modifications to our programs.

PURGABLE LIST FILES—PATIENT, WARD AND MESSAGE FILES

There are two different types of purgable list files—the message files and the intermediate selectable list files.

To facilitate the closed nature of the system it must be possible to present previously entered dynamic data to the user on displays for selection (e.g., a list of a patient's current active problems). The *intermediate selectable list files* perform this function. This ability to display lists for selection was necessary for us to develop because the basic system software (Control Data supplied) allows only for the creation of static displays (via the Selection Element TRANslator) from the keyboard and no ability to dynamically display various lists for selection. Because of time requirements in the displaying of lists, the data to be displayed come from this "intermediate" file rather than from scanning the entire patient's file each time a list is to be created for a user.

The files are directly accessible on the basis of either the patient's system I.D. number (patient list files) or ward number (ward list files). For each patient there are two classes of file entries: patient problem list entries and patient order entries. For each ward, the ward file consists of entries of the patients currently on that ward with additional information such as the status of each of their problems.

Two examples follow: To enter data on a patient, the patient must be identified. This could be done by having the user key-in certain identification information which then would be scanned, verified and used to access the patient's record. The identification is more easily achieved, if the user knows the ward on which the patient is staying, by allowing the user to select the desired patient from a dynamic list of patients on a given ward. The selection, constructed from information in the patient's record (including his name, age, sex, and unit number) has associated with it internal parameters that define this patient's system I. D. number. This allows the STORE or the retrieval programs to directly address the patient's Table of Contents. Another example, which has been very convenient for the nurses in reporting the administration of a drug on a patient, is a list of the current drugs that the patient is receiving.

The "message" file is not a pure "selectable" file since its entries are not used as part of the displays in the system. Its contents are copies of all the additions to the patient order files. In the future these could be sorted and printed out at the proper location in the hospital (e.g., laboratory, pharmacy, x-ray). However, this processing of the message files has not currently been implemented.

THE STORE PROGRAM FOR THE PATIENT'S STRUCTURED FILE

The STORE program stores all narrative and numeric data into the patient's problem oriented medical record (structured non-purgable file). It is executed after the user at the terminal has confirmed as valid all generated paragraphs. No data are stored until after this final verification procedure. The STORE program receives as input the paragraphs and their associated Selection Parameter Lists. A Paragraph Index is built for all the paragraphs input to the program. Each paragraph's identifying data: Information Type 1 and 2, problem number, storage mode, date, time, user number, and patient number are put in a Paragraph Index Element. After the Paragraph Index is built, it is sorted by patient number, date, time, Information Type 1, problem number, Information Type 2. This represents the order that the paragraphs are stored in the patient's file (for normal retrieval).

For each Paragraph Index Element the storage mode

* "Medical user" refers to medical personnel in the PROMIS Laboratory contributing to system development and not to the physician on the ward who does not need to know any of this material to function adequately and who in fact would never be allowed access to SETRAN.

defines which of these STORE routines is executed:

STORE-DIRECT is executed if the paragraph contains the narrative to be stored in the record. (This is the narrative shown on the top of the display as selections are made.) STORE-DIRECT interprets the "F" internal parameters in the Selection Parameter List as format codes and combines them with the selections to form the narrative data in its internal form.

STORE-NUMERIC is executed if the paragraph contains numeric blocks to be stored into the Item. STORE-NUMERIC interprets the "N" parameters in the Selection Parameter List along with the narrative in the paragraph to build the numeric blocks in their internal form.

STORE-TRANSLATED is executed if the paragraph is the result of a questionnaire. The paragraph does not contain narrative selections but the Selection Parameter List contains a record of the selections made on each frame. These paragraphs are formed when "YES" or "NO" questions are answered and the response must be translated into English narrative. The Selection Parameter List is interpreted and an "S" internal parameter associated with any selection signals a dictionary look-up using the frame and the choice number to define the dictionary element. The dictionary elements are concatenated according to rules defined in the dictionary and the resultant data are stored in the internal form for narrative data. Used in conjunction with STORE-TRANSLATED is a programming language similar to the Selection Element TRANslator. This program, the Dictionary Element TRANslator, DETRAN, is used to define the narrative to be associated with any choice, the rules to specify the concatenation of titles with subsequent dictionary elements, and the format codes necessary to specify the output format. Using STORE-TRANSLATED and Dictionary Element TRANslator it has been possible to give a patient a questionnaire in Spanish and have the narrative output in English.

THE STORE PROGRAM FOR THE LIST FILES

The STORE LIST program checks all newly input paragraphs to the patient's record and determines if they should be used in updating the various intermediate selectable list files for that patient, the ward which he is currently on, or the message file.

This program (in its usual mode) takes as input, the paragraphs just stored into a patient's record. This includes both narrative (e.g., problem statement) and numeric (e.g., order) paragraphs. From information in these paragraphs, the program may add, delete or alter entries in the appropriate (patient, ward, and message) list files. For example, if a new order is written for a patient, that order's "text" along with the order problem number, the type code, the frequency and the number of times for administration, will comprise an entry which will be added to that patient's intermediate selectable list file. The entries added to the file are sorted, using one or more elements of the entry as sort-keys, depending upon the entry type (e.g., problem list entry, order entry, ward entry, etc.).

Although this program is usually called by the STORE program, it can be called independently of the STORE program, too. For example, a single patient's intermediate selectable list files can be rebuilt by completely scanning all entries in the patient's record.

RETRIEVAL PROGRAMS—RETRIEVAL OF DATA FROM THE PATIENT'S STRUCTURED FILE

The retrieval programs working on structured files are of two types: The first type retireves both narrative and numeric data in the form of a narrative report; the second forms a time ordered "flowsheet" of various physiological parameters, clinical chemistry results and drugs administered. Both are strictly for the retrieval of data on a single patient.

Each retrieval program can display the retrieved information on either the cathode ray tube terminal or the high speed printer. Input to the retrieval programs is a paragraph which represents the retrieval request, that is, the complete specification of the data to be retrieved. Included with this paragraph is a string of internal parameters in the Selection Parameter List. The user is not aware of these parameters, he need only select the patient and the sections of the record desired (for example, Progress Notes on all active problems, History and Systems Review, or complete record grouped by major sections, i.e., at IT1 level). The retrieval program interprets the parameter string (which is well formed due to the structure of the retrieval frames).

Because each user must identify himself when he signs on, it is possible to allow him access to only certain displays in the system. Using this approach it is possible to limit an individual's access to information within the system by allowing him to formulate only certain retrieval requests.

A retrieval may require one or more retrieval cycles depending on the number of major record sections (IT1's) included in the request and the degree of grouping required in each major section. For each retrieval cycle required, the retrieval routine scans the Item pointers in the patient's Table of Contents to determine which Items contain paragraphs satisfying this retrieval cycle. The Items are then brought into core in the order specified by the applicable Item pointers in the Table of Contents. For each Item the paragraph pointers are scanned, and for each paragraph pointer satisfying the current retrieval cycle request, the FORMAT routine is called to output the paragraph. The address of this paragraph is given to the FORMAT routine along with certain control information requested by FORMAT. The FORMAT routine interprets the paragraph looking for format codes and outputs it, continuing until terminated by the FORMAT termination code, then returning to the retrieval program. Once control is returned from FORMAT, the retrieval routine searches the Item for the next proper paragraph pointer and continues feeding FORMAT until the list of paragraph pointers is exhausted. The retrieval program returns to the Item list, continuing until the Item list is exhausted.

A flowsheet is a time ordered table of multiple medical parameters. Sound interpretation of data involving clinical findings, vital signs, laboratory values, medications, and intakes and outputs requires organization of the data to clearly reveal temporal relationships and clarify the inter-relationships of crucial data. A user requests a flowsheet by selecting the patient, the medical parameters to be included on the flowsheet and the output device (printer or cathode ray tube). (See the flowsheet included in the annotated record.)

RETRIEVAL OF DATA FROM THE INTERMEDIATE SELECTABLE LIST FILES

Although technically a retrieval from the Intermediate Selectable List Files, the creation and presentation of Selectable Lists for the user is done in the context of his storing (or retrieving) other information to (or from) the patient's record. For example, to write a Progress Note about a specific problem on the patient's Problem List, the user must specify on which problem he is entering data. This is done by showing him, in display form, the list of the patient's Current Problems and having him select the proper one. It should be noted that all information in the Selectable List was previously generated and stored by a user.

Input to this program includes the number (type) of Selectable List the user is to see. This number points to an entry in a table which then drives the creation of the frames in the display dictionary containing proper contents from the appropriate Intermediate Selectable List File. The user is then automatically shown the first display containing the list of elements. If more than one display is necessary, the additional displays are linked to the first display. The selection of the appropriate element in the list is then made under the Human Interface Program.

The complex of the Intermediate Selectable List Files with Store List and the subsequent creation of Selectable Lists allows information previously entered into the system to govern the storage and retrieval of other patient information, facilitating a closed system.

AN ANNOTATED EXAMPLE OF AN ACTUAL RECORD GENERATED ON THE SYSTEM

The following is an actual record from one of the patients on the computerized ward. This is a complete "cycled" record; i.e., data that have been added to a section are output chronologically within that section (e.g., page 6 includes the PMH & SR additions entered to the G.U./Renal and Neurology sections by the physician after reading the history). This printed output serves as the "paper" chart and is kept in the chart rack where the traditional paper record was kept. In this way a back-up record is always available, and attendings or consultants can utilize this paper record as well as the cathode ray tube terminal. This printed output is never written on and a new copy (or any updates) is printed daily.

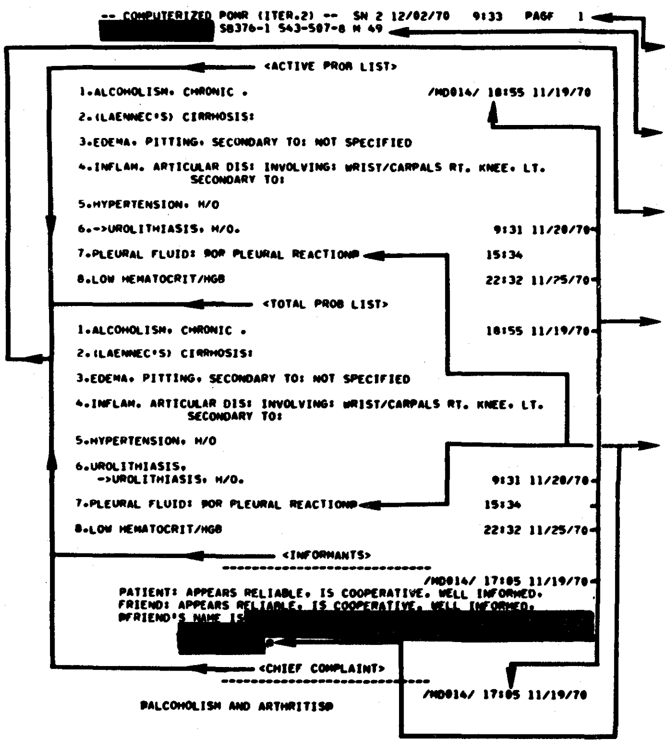
The annotations associated with each page will help explain how the record is constructed, its relationship to the data as they are stored in the patient's structured non-purgable file, and the user's relationship to various aspects of the data as additional information is added to the record. In the annotations, the following abbreviations are used in specifying the different storage modes (SM):

SM = D	Store Directly from selections or from keyboard.
SM = T	Store Translated by a dictionary lookup based upon the frame number and the choice number.
SM = N	Store Numeric from selections in an internal form which allows multiple numeric blocks within one paragraph (may include typed in information).

The purpose of duplication of the first page of the case is to show the layout and then the content on the same page. It would be helpful to refer to the Explanatory Legend for Figure 3 before proceeding. The blacked out spaces throughout the case are names which have been covered to protect the confidentiality of the patient.

ANNOTATED EXAMPLES OF THE SELECTABLE LIST FILES ASSOCIATED WITH THE PRECEDING RECORD

The following pages are copies as they appear on the cathode ray tube screen of the Selectable List on the same patient whose record has just been presented.



```
-- COMPUTERIZED POMR (ITER.2) --  SN 2 12/02/70   9:33   PAGF   1
[redacted] S8376-1 543-507-8 M 49

                    <ACTIVE PROB LIST>
 1.ALCOHOLISM. CHRONIC .                      /MD014/ 18:55 11/19/70

 2.(LAENNEC'S) CIRRHOSIS:

 3.EDEMA. PITTING. SECONDARY TO: NOT SPECIFIED

 4.INFLAM. ARTICULAR DIS: INVOLVING: WRIST/CARPALS RT. KNEE. LT.
              SECONDARY TO:

 5.HYPERTENSION. H/O

 6.->UROLITHIASIS. H/O.                               9:31 11/20/70

 7.PLEURAL FLUID: @OR PLEURAL REACTION@               15:34

 8.LOW HEMATOCRIT/HGB                                22:32 11/25/70
                    <TOTAL PROB LIST>
 1.ALCOHOLISM. CHRONIC .                             18:55 11/19/70

 2.(LAENNEC'S) CIRRHOSIS:

 3.EDEMA. PITTING. SECONDARY TO: NOT SPECIFIED

 4.INFLAM. ARTICULAR DIS: INVOLVING: WRIST/CARPALS RT. KNEE. LT.
              SECONDARY TO:

 5.HYPERTENSION. H/O

 6.UROLITHIASIS.
    ->UROLITHIASIS. H/O.                              9:31 11/20/70

 7.PLEURAL FLUID: @OR PLEURAL REACTION@               15:34

 8.LOW HEMATOCRIT/HGB                                22:32 11/25/70
                      <INFORMANTS>
              ----------------------------
                                              /MD014/ 17:05 11/19/70
     PATIENT: APPEARS RELIABLE. IS COOPERATIVE. WELL INFORMED.
     FRIEND: APPEARS RELIABLE. IS COOPERATIVE. WELL INFORMED.
     @FRIEND'S NAME IS [redacted]

                    <CHIEF COMPLAINT>
              ----------------------------
                                              /MD014/ 17:05 11/19/70
        @ALCOHOLISM AND ARTHRITIS@
```

GENERAL CHARACTERISTICS OF EACH PAGE

1. The retrieval program outputs this header on the top of each printed page. SN2 is the station # from which the retrieval was requested. The date and time (12/2/70 9:33) is the date and time of this retrieval. The page # is sequential within each retrieval.

2. This identification data is contained in the patient's Table of Contents and is part of the header on each page to uniquely identify the output. Note that the name has been blocked out.

3. The centered titles are output whenever a new section (Information Type One--IT1) is output. They are not stored in the record but are supplied by the retrieval program.

4. The subtitles are output whenever a new subsection (Information Type Two--IT2) is output. (See page 2 of record.)

5. The user number, time, and date are displayed at the start of the paragraph or paragraphs to which they apply, whenever they change.

   NOTE: When the user number, the time or the date do not change, they are not output. The date, time, and user number are positioned on the right to avoid distractions from the medical content.

6. All typed in data entered from the keyboard are denoted by the "@" signs surrounding the data.

```
-- [redacted] POMR (ITER.2) --  SN 4 12/10/70   10:58   PAGE   1
[redacted] SRHOLD2 543-507-8 M 49

                    <ACTIVE PROB LIST>
1.ALCOHOLISM. CHRONIC .                      /MD014/ 18:55 11/19/70

2.(LAENNEC'S) CIRRHOSIS:

3.EDEMA. PITTING. SECONDARY TO: NOT SPECIFIED

4.INFLAM. ARTICULAR DIS: INVOLVING: WRIST/CARPALS RT. KNEE. LT.
             SECONDARY TO:

5.HYPERTENSION. H/O

6.->UROLITHIASIS. H/O.                              9:31 11/20/70

7.PLEURAL FLUID: @OR PLEURAL REACTION@              15:34

8.LOW HEMATOCRIT/HGB                                22:32 11/25/70
                    <TOTAL PROB LIST>
1.ALCOHOLISM. CHRONIC .                             18:55 11/19/70

2.(LAENNEC'S) CIRRHOSIS:

3.EDEMA. PITTING. SECONDARY TO: NOT SPECIFIED

4.INFLAM. ARTICULAR DIS: INVOLVING: WRIST/CARPALS RT. KNEE. LT.
             SECONDARY TO:

5.HYPERTENSION. H/O

6.UROLITHIASIS.
   ->UROLITHIASIS. H/O.
                                                    9:31 11/20/70

7.PLEURAL FLUID: @OR PLEURAL REACTION@              15:34

8.LOW HEMATOCRIT/HGB                                22:32 11/25/70
                      <INFORMANTS>
             ----------------------------
                                             /MD014/ 17:05 11/19/70
   PATIENT: APPEARS RELIABLE. IS COOPERATIVE. WELL INFORMED.
   FRIEND: APPEARS RELIABLE. IS COOPERATIVE. WELL INFORMED.
   @FRIEND'S NAME@ IS [redacted]

                    <CHIEF COMPLAINT>
             ----------------------------
                                             /MD014/ 17:05 11/19/70
   @ALCOHOLISM AND ARTHRITIS@
```

NOTE: All material printed here (exclusive of certain titles) was retrieved by the RETRIEVE program from the patient's record. All of this information was stored into the patient's record by the STORE program.

NOTE: The "ACTIVE PROBLEM LIST" is a subset of the "TOTAL PROBLEM LIST", consisting of the last statement of each of the patient's active problems.
For the problem list, the SM=D, IT1=prob. list. IT2=body system under which problem is defined (this cannot be determined from the printout). The IT2 is known to the system, being associated with each problem when displayed on the list of the patient's problems by the VISUAL LIST program before the user writes a problem oriented note. The IT2 is used to determine the group of tests and symptoms from which the user can choose in writing an INITIAL PLAN or PROGRESS NOTE. For example, if the user selected "Low hematocrit/HGB" writing a progress note, then the system would automatically display the HEMATOPOETIC progress note displays.

NOTE: The "TOTAL PROBLEM LIST" is the "ACTIVE PROBLEM LIST" plus all problems that have been resolved/inactivated or combined. Here the complete history of the problem can be seen, c.f. problem #6. The "-->" on the "ACTIVE" list indicates the problem was restated (updated); on the "TOTAL" list, we see the original definition of the problem as well.

NOTE: All sections, "INFORMANTS" through "PHYSICAL DATA BASE BY SYSTEM" are all part of the DATA BASE part of the record (Box 1 in the 4 phases of medical action) and are not problem oriented.

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  10:58   PAGE   2
[redacted]  SBHOLD2 543-507-8 M 49

                    <PATIENT PROFILE>
                  ----------------------------
                                          /MD014/ 18:55 11/19/70
     @PT. IS A 49 YEAR OLD WHITE WIDOWED MACHINIST WHO HAS
          ALLEGEDLY BEEN OUT OF WORK SINCE MID-SUMMER. ALTHOUGH
          HE CLAIMS TO HAVE BEEN OUT OF WORK ONLY FOR THE PAST
          TWO WEEKS. HE HAS BEEN AN ALCOHOLIC SINCE HIS TEENS.
          CHRONICALLY. BUT RECENTLY (OVER THE PAST FEW MONTHS)
          HAS ALLEGEDLY INCREASED HIS INTAKE CONSIDERABLY. FOR
          THE PAST 2-4 WEEKS HE HAS BEEN DOING MOST OF HIS
          DRINKING AT HOME. SINCE THE HARTENDERS IN HIS AREA
          HAVE RECOGNIZED THE SERIOUSNESS OF HIS PROBLEM AND
          HAVE REFUSED TO SERVE HIM. ACCORDING TO HIS FRIEND. HE
          HAS BEEN SPENDING A GREAT DEAL OF TIME AT HOME IN BED
          AND HAS HAD. FROM TIME TO TIME SOME STRANGE IDEAS. FOR
          EXAMPLE. HE AT ONE TIME STATED THAT HE WAS GOING TO
          FIND HIS WIFE (DEAD FOR ABOUT TWO YEARS). ALSO HE
          CLAIMED TO HAVE HAD AN APPLICATION IN AT WEEKS SCHOOL.
          SOMETHING WHICH IS ALLEGEDLY FALSE.@
```

NOTE: This entire "PATIENT PROFILE" section was typed in by the physician using the "KEY" (type-in) program. This section of the record is one of the few areas which is not structured, allowing only typed-in information.

```
                  <PRES ILL - NEW PROB>
                  ----------------------------
SX:
```

This is a subtitle.

```
                                          /MD014/ 18:55 11/19/70
    DRINKING PROBLEM
       ONSET: GRADUAL (INSIDIOUS).
       COMMENCED: 30 YEARS AGO.
       SEVERITY AT WORST: SEVERE.
       AMOUNT AT WORST: @"4-5 SHOTS AND A FEW BOTTLES OF BEER" .
            FRIEND RELATES THAT HE DRINKS UP TO 3-4 SIX PACKS PER
            DAY.@ COURSE HAS BEEN CONTINUOUS SINCE ONSET.
       ASS.W/ @DENIES HAVING HAD DT'S OR CONVULSIONS. DENIES
            BLEEDING PROBLEM.@
       OVERALL COURSE: GETTING WORSE.
       PATIENT'S ATTITUDE: DOES NOT UNDERSTAND. IS INDIFFERENT. IS
            UNREALISTIC. DOES NOT ACCEPT STATEMENT OF PROBLEM.
    JOINT PAIN
       ONSET: SUDDEN (ABRUPT).
       COMMENCED: 1 WEEKS AGO.
       ANTECEDENT TO ONSET: @CLAIMS BITTEN BY SPIDER ON RIGHT
            WRIST@ NOT REQUIRING RX.
       AMOUNT AT WORST: CAN'T CONT. USUAL ACT: GOES TO BED. STAYS
            HOME FROM WORK: CAN'T DO PHYSICAL WORK.
       QUALITY: A DULL ACHE CONTINUOUS SINCE ONSET:
       LOCATION: MULTIPLE JOINTS ASSYMMETRICAL: WRIST (S). RIGHT.
            KNEE (S). LEFT.
       RELIEVED BY: NOTHING.
       MADE WORSE BY: NOTHING IN PARTICULAR.
       ASSOCIATED WITH: COULDN'T DETERMINE.
       OVERALL COURSE: UNCHANGED.
       PATIENT'S ATTITUDE: DOES NOT UNDERSTAND. ACCEPTS STATEMENT
            OF PROBLEM.
    EDEMA:
       ONSET: GRADUAL (INSIDIOUS).
```

This demonstrates the meta-structure approach in handling symptoms.

NOTE: Example of indentation levels of FORMAT routine as used by RETRIEVE to the printer: "SX" is at level 0; "DRINKING PROBLEM" at level 1; "ONSET", "COMMENCED", etc. at level 2. These codes are stored with the paragraphs of data by the STORE program since they will be needed every time that this data is retrieved from the record.

NOTE: Because of the specified mode of retrieval ("cycled"), each body system contains all HISTORY & LAB DATA BASE material entered under that body system in a cumulative manner, regardless of the time of entry or the aspect of the HISTORY & LAB DATA BASE (e.g. LAB order, patient HX). This ability of the RETRIEVE program allows an integrative association of subject-related information which is entered in a temporally unrelated manner. This is facilitated by the STORE program storing the information in such a way that this and other associations (e.g. flowsheets output by RETRIEVE-FLOWSHEET) are possible. This section of the record is "cycled" on each body system and may contain patient entered HISTORY (SM=T),physician entered HISTORY (SM=D) and/or physician entered LAB orders and reports (SM=N). This demonstrates that one body system under this ITl can contain all modes of storage (see GENITO-URINARY/RENAL, below).

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  10:58   PAGE   3
[redacted]  SBHOLD2 543-507-8 M 49

       COMMENCED: 1 WEEKS AGO.
       SEVERITY AT WORST: MODERATE. QUALITY: PITTING.
       DISTRIBUTION: LOCALIZED.
       LOCATION: FEET ONLY. BILATERAL. COURSE HAS BEEN CONTINUOUS
            SINCE ONSET.
       RELIEVED BY: NOTHING.
       NOT RELIEVED BY: ANYTHING.
       MADE WORSE BY: NOTHING IN PARTICULAR.
       ASSOCIATED WITH: COULDN'T DETERMINE.
       OVERALL COURSE: SUBSIDING.
       PATIENT'S ATTITUDE: DOES NOT UNDERSTAND. ACCEPTS STATEMENT
            OF PROBLEM.

                  <HISTORY & LAB DATA BASE>
                  ----------------------------
                                          /MD212/ 15:14 11/19/70
  PATIENT ADMINISTERED HISTORY FOLLOWS:
                                          /MD014/ 18:55
     (HX NOT ENTERED BY PT)

CONSTITUTIONAL SUMMARY/GENERAL:           /MD212/ 15:14
     ADULT MALE.
     NO WEIGHT LOSS.  PECENTLY LOST APPETITE.  NOT FEELING TIRED.
          NOT HAVING FEVER.
                                          /MD014/ 18:55
     @DRUGS - HIGH BLOOD PRESSURE PILL 1-2 PER DAY DEPENDING ON THE
          WAY HE FELT@
```

NOTE: The sections of the HISTORY which are not typed in have been produced by the patient sitting at a terminal answering questions. The responses are translated and stored by the STORE program (SM=T), utilizing the translation dictionary created by the DETRAN program. The I.D. code (MD212) in this case refers to the individual who started the patient on the history.

NOTE: Typed in sections of the HISTORY are corrections or additions made by the physician after reading the patient generated HISTORY (SM=D).

```
SOCIAL PROFILE:
                                          /MD212/ 15:14
     ADULT MALE.  AGE 49.
     BORN IN VERMONT: TOWN OF LESS THAN 5.000 PEOPLE.  LIVED IN AREA
          OF CURRENT RESIDENCE FOR ALL OF LIFE.
     LAST COMPLETED GRADE: 10TH.  WOULD NOT LIKE FURTHER EDUCATION
          OR TRAINING.
     MARRIED.  DOES NOT LIVE W/ WIFE.  LIVES ALONE.  COOKS OWN
          MEALS.  WIDOWED FOR >1 YEAR.
     SATISIFIED W/ PRESENT LIVING CONDITIONS.
     SERVED IN ARMED FORCES.
     PRESENTLY EMPLOYED.  SATISFIED WITH PRESENT JOB.  DOES WORK
          SITTING DOWN.  GETS DAILY EXERCISE.
     SUPPORTED MAINLY BY SELF.
     DRINKS MANY DRINKS/DAY.  SOMETIMES: MISSES WORK ON MONDAY
          MORNINGS: TAKES A DRINK IN THE MORNING.  HOSPITALIZED
          DUE TO DRINKING.
     EATS 2 OR MORE MEALS/DAY: MEAT OR EGGS WITH 1 OR MORE OF THEM.
        26 POS. 16 NEG.  0 DNK.  0 DNU.
```

NOTE: The larger patient generated HISTORY sections are followed by a summary of the number of POSitive, NEGative, Do Not Know, and Do Not Understand responses to that section of the patient administered HISTORY.

```
INFECTIOUS DISEASE:
     HAS HAD CHICKEN POX. RUBELLA. PUBEOLA. HEPATITIS. MUMPS.
          TYPHOID. WHOOPING COUGH.
        7 POS.  1 NEG.  0 DNK.  0 DNU.

IMMUNIZATIONS:
```

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  10:58   PAGE   4
[redacted] SBHOLD2 543-507-8 M 49

     INJECTIONS IN PAST FIVE YEARS: INFLUENZA: SMALL POX
             (VACCINATION): TYPHOID: DPT.
     IN PAST TEN YEARS HAS HAD POLIO SHOTS.
          5 POS.  0 NEG.  0 DNK.  0 DNU.

FAMILY HISTORY/GENETIC:
     HYPERTENSION- FATHER.  STROKE- FATHER.  HYPERTENSION- MOTHER.
             HEART DISEASE- FATHER.  T.B.- SISTER.
     FATHER DIED AT AGE AGED 50 TO 59.  MOTHER DIED AT AGE 60 TO 69.
             PATERNAL GRANDFATHER DIED AT AGE UNKNOWN.  NOT KNOWN
             IF PATERNAL GRANDMOTHER LIVING.  MATERNAL GRANDFATHER
             DIED AT AGE NOT KNOWN.  MATERNAL GRANDMOTHER DIED AT
             AGE NOT KNOWN.
         16 POS.  6 NEG.  2 DNK.  0 DNU.

DERM-ALLERGY:
     OFTEN WORKED AROUND CHEMICALS. SOLVENTS. OR CLEANING FLUIDS.
          1 POS.  6 NEG.  0 DNK.  0 DNU.

EYE. EAR. NOSE. THROAT:
     WEARS OR HAS WORN GLASSES.  W/O GLASSES HAS TROUBLE SEEING UP
             CLOSE.  VISION SATISFACTORY W/ GLASSES.
          3 POS. 17 NEG.  0 DNK.  0 DNU.
     MUCH LOUD NOISE IN PLACE OF EMPLOYMENT.
     HAS SHOT A GUN A GREAT DEAL.
          2 POS. 24 NEG.  0 DNK.  0 DNU.

DENTAL:
     H/O DENTAL X-RAYS.
     EATS MUCH BREAD. POTATOES OR MACARONI.  SOMETIMES EATS RAW
             VEGETABLES.
     BRUSHES TEETH W/ TOOTHBRUSH ONCE A DAY.
          4 POS. 13 NEG.  0 DNK.  0 DNU.

HEMATOPOETIC:
          0 POS. 12 NEG.  0 DNK.  0 DNU.
                                                   /MD014/ 17:05
  CBC:
  SED RATE
                                                   /MD212/  8:45 11/20/70
  12/HPF   CBC: HCT (%) 39: HGB (GMS%) 12.5: WBC 8800 : WBC DIFF SEG/
               54%: NSEG/ 1%: LMC/ 33%: MON/ 10%: EOS/ 1%: BAS/ 1%:
               PLATELETS
  38MM/HR UNCORR .29MM/HR CORR . SED RATE
                                                   /MD014/ 21:51 11/23/70
  HGB
  HCT
                                                   /CC025/ 16:31 11/24/70
  11.4GMS %  HGB
  36%  HCT

RESPIRATORY:
                                                   /MD212/ 15:14 11/19/70
     H/O SINUS TROUBLE.
```

NOTE: These are LAB orders under the HX & LAB DATA BASE (IT1) section (SM=N). These are each stored as numeric blocks in the patient's record by the STORE program. After each addition to a patient's record, the STORE program calls STORE-LIST which then processes new entries to the patient's record (by using a RETRIEVE routine). The new entries in this case, would be placed on a list of this patient's outstanding lab. orders(see the lists on pages 67,68 for examples).

NOTE: These are the reported values for the previous two lab. orders, also stored as numeric blocks by the STORE program (SM=N). These orders are reported by calling up the list (done by the VISUAL LIST program) of outstanding lab. orders for this patient as constructed by STORE-LIST. The set of displays containing the appropriate choices for reporting the value is shown to the user when he chooses the order to report. This set of displays is pre-defined for each order at the time it is ordered by means of "N" parameters which cause the placement of a specific value in a specific field in the numeric block for that order. In most cases, these results can also appear on flowsheets. This is one reason why the lab. value in the actual report often preceeds the name of the test (e.g. sed. rate report). After these results have been stored in the patient's record by STORE, STORE-LIST will process these numeric blocks, removing the "CBC" and "SED.RATE" orders from the list of outstanding orders, since unless otherwise specified, lab. orders are assumed to be executed only once.

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  10:58   PAGE   5
[redacted] SBHOLD2 543-507-8 M 49

     CONTACT W/ PERSON W/ T.B..
     SMOKES CIGARETTES: 1 PACK/DAY FOR 31-40 YEARS.
          5 POS. 15 NEG.  2 DNK.  0 DNU.
                                                   /MD014/ 17:05
  CHEST X-RAY (PA)
                                                   /CC025/  0:58 11/22/70
  <DONE>  CHEST X-RAY (PA) @PA AND LAT CHEST ..A NORMAL CARDIAC
               SILHOUETTE. ..LUNG FIELDS..CLEAR EXCEPT FOR A RT.
               PLEURAL ANGLE BLUNTING WHICH WAS NOT PRESENT IN 1968.
               THIS MAY REPRESENT OLD PLEURAL REACTION FROM A
               PREVIOUS ILLNESS BUT COULD REPRESENT A NEW PROCESS.
               FLUOROSCOPIC EXAM RECOMMENDED.@

BREAST:
                                                   /MD212/ 15:14 11/19/70
          0 POS.  4 NEG.  0 DNK.  0 DNU.

CARDIOVASCULAR:
     PAINLESS SWELLING FREQUENT IN BOTH FEET OR ANKLES WHICH GOES
             DOWN OVERNIGHT.
     HAS TAKEN HYPERTENSIVE MEDS
     TOLD HAS HAD HIGH BLOOD PRESSURE.
          5 POS. 17 NEG.  1 DNK.  0 DNU.
                                                   /MD014/ 17:05
  EKG
                                                   /CC025/ 17:28 11/23/70
  <DONE>  EKG @RHYTHM=SINUSA &V RATES=ABOUT 90. PR INT=0.16. QRS
               INT=0.08. QTC=NL. AXIS=RLQ. INTERPRETATION=SINUS
               RHYTHM. PVC'S ARE PRESENT. POOR R WAVE UNTIL V3.@

GASTROINTESTINAL:
                                                   /MD212/ 15:14 11/19/70
     H/O ESOPHAGEAL X-RAYS.
     H/O STOMACH X-RAYS.
     H/O LIVER DISEASE. JAUNDICE.
     H/O GALL BLADDER X-RAYS.
     H/O X-RAYS OF BOWEL.
     H/O HEMORRHOIDS.
          7 POS. 29 NEG.  0 DNK.  0 DNU.

MUSCULO-SKELETAL:
     H/O SHOULDER TROUBLE.
          1 POS. 18 NEG.  0 DNK.  0 DNU.

ENDOCRINE:
     MORE CLOTHES WORN IN COLD WEATHER THAN BEFORE.
     LOWEST WEIGHT AS ADULT 110-120 LBS
     GREATEST WEIGHT AS ADULT 160-170 LBS (AT AGE 20-30).
          4 POS. 16 NEG.  1 DNK.  0 DNU.
                                                   /MD014/ 17:05
  2 HR. P.C. GLUC
                                                         19:28
  NA
  K
```

NOTE: This demonstrates that material can be typed in (using the KEY program) when reporting a test result that is stored as a numeric block by the STORE program (SM=N).

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  10:59   PAGE   6
[redacted] SBHOLD2 543-507-8 M 49

  CO2
  CL
                                                /MD212/  8:45 11/20/70
  135 MEQ/L NA
  3.5 MEQ/L K
  27 MEQ/L CO2
  95 MEQ/L CL
  112 MG/100 ML 2 HR. P.C. GLUC  *NOT STATED ON LAB SLIP WHETHER 2
            HR PC*

GENITO-URINARY/RENAL:
                                                      15:14 11/19/70
     URINE HAS BEEN BLOODY OR COFFEE COLORED.
     TOLD HAD KIDNEY OR BLADDER STONES: STONES PASSED W/ URINE.
     TOLD HAD KIDNEY OR BLADDER INFECTION
     DOES NOT HAVE ANY SEX PROBLEMS.
     HAS NOT BEEN CIRCUMSIZED.
          6 POS, 18 NEG,  2 DNK,  0 DNU.
                                                /MD014/ 17:05
  BUN
  ROUTINE URINE
  QUAL. VDRL
                                                      18:55
     *HAD KIDNEY OR BLADDER STONES SOME TIME AROUND 1942, NONE
            SINCE. STATES THE DRS. "DISSOLVED" THE STONES WITH A
            SPECIAL LIQUID (DR. [redacted]). NO GU SYMPTOMS
            RECENTLY/*
                                                /MD212/  8:45 11/20/70
  ROUTINE URINE YEL. CLR, SPGR: 1.005, PH 5, PROT NEG, CHO NEG. AC.
            NEG. SED: SP, WBC: FEW./HPF,
  3 MG/100 ML BUN
                                                /CC025/ 15:51
  NEGATIVE.  QUAL. VDRL
                                                /MD014/ 21:51 11/23/70
  S. CREATININE
                                                /CC025/ 16:31 11/24/70
  1.0 MG/100 ML S. CREATININE

NEUROLOGY:
                                                /MD212/ 15:14 11/19/70
     TROUBLE MOVING ARMS AND LEGS ON BOTH SIDES.
     NUMBNESS OR TINGLING OF ARMS, HANDS, LEGS OR FEET PRESENT FOR
            SEVERAL WEEKS.
          5 POS, 18 NEG,  0 DNK,  0 DNU.
                                                /MD014/ 18:55
     *DIFFICULTY MOVING THE LEFT LEG IS BECAUSE OF THE PAIN
            ASSOCIATED WITH THE ARTHRITIS IN THE LEFT KNEE. DENIES
            NUMBNESS OR TINGLING IN THE ARMS AND LEGS.*

PSYCHIATRY:
                                                /MD212/ 15:14
     DECREASED INTEREST OR ENJOYMENT IN SEX.
          2 POS, 34 NEG,  0 DNK,  0 DNU.
```

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  10:59   PAGE   7
[redacted] SBHOLD2 543-507-8 M 49

SUMMARY STATISTICS:
    INTERVIEW COMPLETED.
    TOTALS.
       105 POS,267 NEG,  8 DNK,  0 DNU. <----
```

NOTE: These are the totals for the patient administered HISTORY sections as stored in the patient's record by the STORE program (SM=T).

```
               <PHYSICAL DATA BASE BY SYSTEM>
                -----------------------------
                                   /MD212/ 15:03 11/19/70

  VITAL SIGNS:                                                  <----
     TEMP, ORAL(DEGREES C): 36.8
     PULSE, RADIAL: 80/MIN, RHYTHM NOT NOTED,
     RESPIRATIONS: 20/MIN,
     BP, LT ARM, SITTING: 1 62/90 MM HG, VP: NOT EVALUATED.
     WEIGHT, LB: 1 25. HEIGHT/LENGTH: NOT MEAS.
                                   /MD014/ 17:05
```

NOTE: These vital signs were entered as soon as the patient arrived on the ward.

```
  VITAL SIGNS: TEMP: NOT TAKEN.                                 <----
     PULSE, RADIAL: 1 08/MIN, REGULAR,
     RESPIRATIONS: 20/MIN,
     BP, RT ARM, SUPINE: 170/105 MM HG,
     BP, LT ARM, SUPINE: 170/105 MM HG,
     BP, RT ARM, STANDING: 1 70/104 MM HG,
     JVP: 0 CM ABOVE STERNAL ANGLE AT 45 DEGREES ELEV. WEIGHT: NOT
            DETERMINED. HEIGHT/LENGTH: NOT MEAS.
```

NOTE: These vital signs and the rest of the physical exam were entered by the physician following the normal patient work-up on the ward.

```
  GENERAL APPEARANCE:
     THE PATIENT IS A CHRONICALLY ILL, NORMALLY NOURISHED, MIDDLE
            AGED (APPEARS STATED AGE), CAUCASIAN MALE: RESPONSIVE
            COOPERATIVE, AND CAN CARE FOR SELF.
     CURRENTLY REQUIRES: NO AIDS. PT IS ANXIOUS.

  SKIN: NORMAL.

  HEAD: NORMAL.

  EYES:
     EYE MOVEMENTS: EYE MOVEMENTS NORMAL.
     PUPILS: ROUND, EQUAL, 4 MM, ESTIMATED, REACT TO LIGHT & ACCOM.,
     FUNDUS:
        BILAT: NORMAL.

  EARS:
     EXTERNAL CANAL: BILAT: NORMAL.
     TYMPANIC MEMBRANE: BILAT: NORMAL COLOR, MID POSITION, LIGHT
            REFLEX NORMAL,
     HEARING: NORMAL BILAT.

  NOSE & NASOPHARYNX: NORMAL.

  OROPHARYNX:
     TONGUE: BILAT, *COARSE TREMOR, GENERALIZED*
     TEETH:
        GENERALLY: MANY ABSENT,
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  10:59   PAGE   8
[redacted]SBHOLD2 543-507-8 M 49

NECK:
   THYROID: NORMAL SIZED, SYMMETRICAL.

LYMPH NODES: NONE PALPABLE.

CHEST AND LUNGS:
      RESPIRATION: NORMAL.
      INSPECTION: NORMAL.
      PALPATION: NORMAL.
      PERCUSSION: RESONANT THROUGHOUT:
      AUSCULTATION:
      NORMAL BREATH SOUNDS: BILAT. ENTIRE CHEST.

CARDIOVASCULAR:
   ARTERIAL PULSES: ALL NORMAL.
   JUGULAR VENOUS PULSE: VENOUS PRESSURE: 0 CM ABOVE STERNAL ANGLE
            AT 45 DEGREES ELEVATION.
   PALPATION:
      APEX BEAT: LOCALIZED AT MCL IN 4TH ICS.
   AUSCULTATION: NORMAL.

ABDOMEN: INSPECTION NORMAL. PERCUSSION NORMAL. AUSCULTATION
            NORMAL.

RECTAL:
   NORMAL.
   STOOL ABSENT.

EXTREMITIES:
   JOINTS:
      WRIST: RT: SWELLING/ENLARGEMENT: OVER ENTIRE JOINT. MAINLY
            SOFT TISSUE. RUBBERY. TENDER. HOT.
   JOINTS:
      KNEE. LT: PAIN ON MOTION: ACTIVE & PASSIVE. CONSTANT.
            INFLAMMATION W/O SWELL.: OVER ENTIRE JOINT.

EXTREMITIES:
   LOWER LEG: 3+ PITTING EDEMA. ANKLES. BILAT. RT = LT.

BACK/FLANK/SPINE:
      PALP/PERCUSSION PAIN: BILAT. SOFT TISSUE: FLANK.
            COSTOVERTEBRAL ANGLE. MILD. BILAT. SOFT TISSUE: MILD.

MALE GENITALIA:
   PUBIC HAIR: FEMALE ESCUTCHEON.

NEUROLOGIC EXAM:
   MENTAL STATUS: FULLY RESPONSIVE. DAY OF WEEK. DATE OF MONTH.
            FULLY COOPERATIVE.
      ACTIVITY & BEHAVIOR: NORMAL MOTOR ACTIVITY. ABLE TO CARE FOR
            SELF. NORMAL BEHAVIOR.
      APPEARANCE: UNKEMPT. SLOPPY.
      MEMORY.INTELL..PARIETAL: REMOTE MOD. IMPAIRED. NORMAL
```

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  10:59   PAGE   9
[redacted]SBHOLD2 543-507-8 M 49

         INTELLIGENCE. ABLE TO ABSTRACT.
      FUND OF KNOWLEDGE: SLIGHTLY DEFICIENT.
      SPEECH: NORMAL.
      CALCULATIONS: DOES SERIAL 7'S.
      INSIGHT & JUDGEMENT: UNDERSTANDS ILLNESS. JUDGEMENT NORMAL.
      MOOD & AFFECT: WITHDRAWN. FLAT.
      THOUGHT CONTENT: NORMAL.
   CRANIAL NERVES:
      I: NOT TESTED.
      II: ACUITY NOT TESTED. COLOR VISION NOT TESTED. VISUAL
            FIELDS NORMAL TO CONFRONTATION. FUNDI NORMAL.
      III.IV.VI: RECORDED UNDER EYE EXAM
      V: MASTICATION INTACT. SENSATION INTACT.
      VII: MOTOR INTACT. TASTE NOT TESTED.
      VIII: HEARING NORMAL. CALORICS NOT TESTED.
      IX.X: NORMAL.
      XI: NORMAL.
      XII: TREMOR. BILAT.
   MOTOR: (RT HANDED)
      STRENGTH: NORMAL.
      BULK:
         ATROPHY. SLIGHT: BILAT
      TONE: NORMAL.
      COORDINATION: NORMAL.
      GAIT. STANCE: VEERS. TO LT.
      ABNORMAL MOVEMENTS: TREMOR. ENTIRE BODY. AT REST.
            CONTINUOUS. MODERATE (6-9/SEC). COARSE.
   REFLEXES:
      DTR'S:
         BICEPS JERK: BILAT. 2+.
         TRICEPS JERK: BILAT. 2+.
         KNEE JERK: BILAT. 2+.
         ANKLE JERK: BILAT. 0.
      PLANTAR RESPONSE: RT FLEXOR. LT FLEXOR.
      ABDOMINAL PRESENT BILAT.
   SENSORY: COOPERATION GOOD.
      SUPERFICIAL PAIN: NORMAL.
      TOUCH: NORMAL.
      VIBRATION: MODERATELY DECREASED RT SIDE: LEG.
      VIBRATION: SLIGHTLY DECREASED LT SIDE: LEG.

--->

                    <GEN WARD INFO>
              ---------------------------
PLN-GEN WARD:
                                              /MD014/ 19:28 11/19/70
  PT.ACT:  UP W/ ASSISTANCE @AS TOL.@
  VISITING  REGULAR.
  HOUSE DIET.  FLUIDS AD LIB.
  CHLORAL HYDRATE  1000MG  PRN X1. P.O.
  MILK OF MAGNESIA SUSP.  30ML  PRN X1. P.O.
  WEIGHT  QOD
  TEMP  Q8H
```

← NOTE: "GENERAL WARD INFORMATION" contains information necessary for general ward/hotel functions. This section is not problem oriented.

← NOTE: These are actual orders which are stored in the patient's record as numeric blocks by STORE (SM=N). Similar to the method mentioned above, these are added to a list of this patient's outstanding orders by STORE-LIST. (Note these on page 67 on the patient Non-Rx order list.)

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  10:59   PAGE  10
[redacted] SBHOLD2 543-507-8 M 49

  PULSE  Q8H
  RESP  Q8H
  BP  Q8H
OBJ:
                                              /UN024/ 22:04
  CHLORAL HYDRATE 1000M*
                                              /US010/  8:07 11/20/70
  37.6°C  TEMP Q8H
  88 /MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                              /UN020/ 10:58
  142/100 BP Q8H  ¶TAKEN AT 8:00 AM.¶
                                              /UN025/ 17:10
  160/110 BP Q8H
  88/MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                                   21:17
  37.3°C  TEMP Q8H ¶8PM TEMP RECORDED LATE¶
  84/MIN  PULSE Q8H ¶8PM RECORDED LATE¶
  20/MIN  RESP Q8H ¶8PM RECORDED LATE¶
                                              /US010/  9:16 11/21/70
  37.2°C  TEMP Q8H
  88 /MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                              /UN013/  9:56
  150/100 BP Q8H
  88/MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                              /UN025/ 15:56
  178/110 BP Q8H
  84/MIN  PULSE Q8H
  22/MIN  RESP Q8H
                                              /US010/ 16:04
  37°C  TEMP Q8H
  84/MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                              /UN014/  6:39 11/22/70
  123 LBS  WEIGHT QOD
                                              /UN015/  7:41
  190/112 BP Q8H
  88/MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                              /US010/  8:46
  36.8°C  TEMP Q8H
  88 /MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                              /UN013/ 10:50
  138/90 BP Q8H
  84/MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                              /UN025/ 15:43
  160/90 BP Q8H
  92/MIN  PULSE Q8H
  22/MIN  RESP Q8H
```

NOTE: This is the execution of the "CHLORAL HYDRATE" ordered immediately above. This execution was specified by the Unit Nurse (UN024) by choosing the order from the outstanding list of orders displayed by the VISUAL LIST program from the list created by STORE-LIST. Note that the entire order is not shown, but rather is truncated with a "*". This was done to make the record easier to read; the entire order is always available in the record. The order is assumed to be executed as stated unless otherwise stated (c.f. blood pressure (BP) entered by UN020 at 10:58). Since this order was specified to be executed only once ("X1"), the order is removed from the list of outstanding orders when the numeric block reflecting the execution (stored by STORE, SM=N) is processed by STORE-LIST.

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:00   PAGE  11
[redacted] SBHOLD2 543-507-8 M 49

  37.4°C  TEMP Q8H                              /US010/ 15:54
  88 /MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                                      7:33 11/23/70
  37.1°C  TEMP Q8H
  108 /MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                                /UN022/  9:49
  160/90 BP Q8H
                                                     14:17
  <DONE>  PT.ACT: UP W/ ASSISTA*
                                                /US010/ 16:05
  37.2°C  TEMP Q8H
  100 /MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                                /UN017/ 18:25
  92/MIN  PULSE Q8H
  2 0/MIN  RESP Q8H
  180/100 BP Q8H
                                                /US010/ 19:36
  92/MIN  PULSE Q8H
  2 0/MIN  RESP Q8H
  162/100  BP Q8H
                                                      7:57 11/24/70
  36.9°C  TEMP Q8H
  92 /MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                                /UN022/ 10:34
  92/MIN  PULSE Q8H
  20/MIN  RESP Q8H
  160/98 BP Q8H
                                                /US010/ 18:17
  37.2°C  TEMP Q8H
  88 /MIN  PULSE Q8H
  20/MIN  RESP Q8H ¶THIS IS A 4:00PM TPR.¶
                                                /UN020/ 18:45
  76/MIN  PULSE Q8H
  2 0/MIN  RESP Q8H
  166/104  BP Q8H
                                                     23:10
  104/MIN  PULSE Q8H
  20/MIN  RESP Q8H
  120/90  BP Q8H ¶TAKEN AT 8:00 PM.¶
                                                /UN015/  7:38 11/25/70
  80/MIN  PULSE Q8H
  170/100 BP Q8H
  20/MIN  RESP Q8H
                                                /US010/  8:19
  36.5°C  TEMP Q8H
  82 /MIN  PULSE Q8H
  20/MIN  RESP Q8H
                                                /UN022/ 13:40
  84/MIN  PULSE Q8H
```

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:00   PAGE  12
[redacted]  SBHOLD2 543-507-8 M 49

20/MIN  RESP Q8H
150/90 BP Q8H
                                               /US010/ 16:09
36.8'C  TEMP Q8H
84 /MIN  PULSE Q8H
20/MIN  RESP Q8H
                                               /UN020/ 17:36
84/MIN  PULSE Q8H
1 6/MIN  RESP Q8H
156/116  BP Q8H @AT 4:00 PM.@
                                                       21:56
96/MIN  PULSE Q8H
2 0/MIN  RESP Q8H
182/100  BP Q8H
                                               /UN043/  6:39 11/26/70
120 LBS  WEIGHT QOD
                                               /US010/  7:59
37.'C  TEMP Q8H
88/MIN  PULSE Q8H
20/MIN  RESP Q8H
                                               /UN020/ 13:37
68/MIN  PULSE Q8H
2 0/MIN  RESP Q8H
188/120  BP Q8H
                                               /UN024/ 15:52
92/MIN  PULSE Q8H
20/MIN  RESP Q8H
160/100 BP Q8H
                                               /UN017/ 15:54
37.'C  TEMP Q8H
96/MIN  PULSE Q8H
24/MIN  RESP Q8H
                                               /UN024/ 19:45
84/MIN  PULSE Q8H
18/MIN  RESP Q8H
164/98 BP Q8H
                                               /UN017/ 20:15
   @NEOLOID 60 CC.S GIVEN P.O. FOR IVP PREP@
                                               /UN014/  0:16 11/27/70
150/92 BP Q8H
80/MIN  PULSE Q8H
18/MIN  RESP Q8H
                                               /US010/  7:45
36.8'C  TEMP Q8H
88 /MIN  PULSE Q8H
20/MIN  RESP Q8H
                                               /UN022/  7:48
162/110 BP Q8H
PLN-GEN WARD:
                                               /MD014/ 14:21
 DISCHARGE PT.  WITH FRIEND, VALUABLES, VIA AMBULATORY.
OBJ:
                                               /US010/ 16:06
 37.2'C  TEMP Q8H
```

NOTE: This is an execution of an order which is part of a preparation regimine for another order. For cases such as this, we instruct the nurse to type-in (using KEY) the order as it was executed under the proper problem or under "GENERAL WARD" if they do not know the problem number. This should have been entered under problem #5 (HYPERTENSION, H/O), for which the IVP was ordered.

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:00   PAGE  13
[redacted]  SBHOLD2 543-507-8 M 49

 96 /MIN  PULSE Q8H
 20/MIN  RESP Q8H
                                               /UN017/ 16:45
    @PT. DISCHARGED AT 5:00 P.M. WITH PRESCRIPTION.@

                       <PLANS - INITIAL>
                  ---------------------------

1.ALCOHOLISM, CHRONIC .
 PLN-R/O:
                                               /MD014/ 19:28 11/19/70
     R/O @IMPENDING DT'S@
 PLN-DDA:
   @PARALDEHYDE@ 7.5ML  Q4H PRN STANDING, @FOR AGITATION@ P.O.
   @CHLORAL HYDRATE@ 1GM  QHS PRN STANDING P.O.
 PLN-PRO:
   @PADDED SIDE RAILS, PADDED TONGUE BLADE HANDY, HAVE SYRINGE WITH
              10 MG. VALIUM IN ROOM@

                         <PROG NOTES>
                  ---------------------------

1.ALCOHOLISM, CHRONIC .
  RX-GIVEN:
                                               /UN025/ 20:37 11/19/70
   PADDED SIDE RAILS, PA*
   @5:20PM@  PARALDEHYDE 7.5ML Q4H*
 SX:
                                               /MD014/  9:31 11/20/70
     @SPENT FAIR NIGHT.@
 OBJ:
     @APPEARS LESS SHAKY THAN LAST NIGHT. PARALDEHYDE WORKING WELL.
            ORIENTED EXCEPT FOR DAY OF MONTH. THINKING STILL
            SLOWED HOWEVER.@
  RX-GIVEN:
                                               /UN020/ 10:58
   PADDED SIDE RAILS, PA*
                                               /UN017/ 22:18
   CHLORAL HYDRATE 1GM Q*
 SX:
                                               /MD014/  9:46 11/21/70
     @IMPROVING. APPETITE RETURNING.@
        OVERALL COURSE: GETTING BETTER.
PLN-DDA:
  @THIAMINE@ 50MG  1/DAY X1, S.O. I.M.
  @THERAGRAN-M@ 1CAP.  1/DAY STANDING, S.O. P.O.
 RX-GIVEN:
                                               /UN017/ 22:16
  CHLORAL HYDRATE 1GM Q*
OBJ:
                                               /CC025/  1:18 11/22/70
    @11/20/70 2 HR PC=110 MG%.@
 RX-GIVEN:
                                               /UN013/ 10:13
```

NOTE: The remainder of this printed recora consists of all of the data in sequence on each of the problems on the "TOTAL PROBLEM LIST", beginning with problem #1. All information on a single problem is not stored physically together by the STORE program in the patient's record, but rather is physically stored in a chronological fashion with pointers to each paragraph (identified by IT1, IT2, Date, Time, and Problem #) so many different associative modes of retrieval are made possible, e.g. RETRIEVE data as it was chronologically entered, RETRIEVE data grouped by problem, RETRIEVE time ordered list of physiologic parameters (FLOW-SHEET), etc. The complexibility and flexibility of the stored patient data is not obvious to most users. The RETRIEVE program, in the "cycled" mode specified here, goes through the record and brings together all information on each problem. All PROGRESS NOTES on a problem are preceeded by any INITIAL PLANS on that problem.

NOTE: These drug orders, after being stored in the record by STORE (SM=N), will be added to the patient's list of outstanding orders by STORE-LIST. Note that the drug names were typed in. This indicates that the drugs given were not on the list of pre-defined drugs for that problem statement. Since these are standing orders, they can still be seen on the list of outstanding Rx orders on page 68.

NOTE: Here are executions of two of the orders from above. Note they are truncated with a "*" for ease in reading the record. Note also that the PARALDEHYDE was not executed as ordered but rather was given at 5:20 p.m.

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:00   PAGE  14
[redacted] SBHOLD2 543-507-8 M 49

  THERAGRAN-M 1CAP. 1/D*
  THIAMINE 50MG 1/DAY X*
SX:
                                             /MD014/ 10:26
    @FEELING BETTER, EATING.@
OBJ:
    @LITTLE TREMOR, CALM. ONLY REQUIRED  INITIAL DOSE OF
           PARALDEHYDE ON ADMISSION, NONE SINCE.@
PLN-DB:
  @SOCIAL SERVICE CONSULT@
PLN-DDA:
                                                  12:47
  @THIAMINE@ 50MG  STAT X1 ONLY, I.M.
 RX-GIVEN:
                                             /UN017/ 22:15
  CHLORAL HYDRATE 1GM Q*
                                             /UN011/ 10:33 11/23/70
  THERAGRAN-M 1CAP. 1/D*
                                             /UN017/ 23:01
  CHLORAL HYDRATE 1GM Q*
CON REPLY:
                                             /MD320/  9:17 11/24/70
    AGREE W/ PROBLEM AS FORMULATED.
    AGREE W/ PLANS AS FORMULA
    AGREE W/ PROBLEM AS FORMULATED.
    AGREE W/ PLANS AS FORMULA
OBJ:
  @ONDAS@ @ON DAY OF ADMISSION, [redacted]
           CALLED ME TO
           REQUEST LIAISON BETWEEN HIS AGENCY AND HOSPITAL. MR
           [redacted] HAS KNOWN PATIENT FOR A LONG TIME AND KNOWS HIS
           ETOH HISTORY QUITE WELL. MR.[redacted] AND OTHER MEMBERS
           OF HIS AGENCY WILL OFFER HELP TO PATIENT WHILE IN
           HOSPITAL AND AFTER DISCHARGE. MR. [redacted] HAS EXPRESSED
           CONCERN AND QUESTIONS IF PATIENT IS COMPETENT TO
           HANDLE HIS PERSONAL AFFAIRS AFTER DISCHARGE FROM THE
           HOSPITAL. @ @
  [redacted]                                  - MFU@
 RX-GIVEN:
                                             /UN013/ 10:22
  THERAGRAN-M 1CAP. 1/D*
CON REPLY:
                                             /MD320/ 12:17
    AGREE W/ PROBLEM AS FORMULATED.
    AGREE W/ PLANS AS FORMULA
OBJ:
  @PLAN: 1. WILL ACT AS LIASON BETWEEN O.E.O. ALCOHOLISM PROGRAM AND
           MFU STAFF. 2. IF OUR DEPARTMENT CAN BE OF HELP IN SOME
           OTHER WAY,PLEASE LET US KNOW.@  @A@ @A@
  [redacted]
OBJ:
                                             /UN020/ 18:45
    @PATIENT APPEARS SOMEWHAT CONFUSED THIS EVENING@
```

NOTE: Here is another order in the form of a request for a SOCIAL SERVICE CONSULT.

NOTE: This is the reply to the request for a SOCIAL SERVICE CONSULT entered by the consultant. There are a few duplicate entries and errors in using the type-in program (KEY).

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:00   PAGE  15
[redacted] SBHOLD2 543-507-8 M 49

 RX-GIVEN:
                                             /UN024/ 22:40
  CHLORAL HYDRATE 1GM Q*
 SX:
                                             /MD014/  9:39 11/25/70
    @IN SPEAKING WITH MR.[redacted] AND WITH TWO OF PATIENTS
           LOCAL RELATIVES. CONCERN HAS BEEN EXPRESSED OVER THE
           QUESTION OF MENTAL DETERIORATION IN THE PATIENT. THEY
           HAVE SUGGESTED THAT HE HAS BECOME SOMEWHAT DULL
           MENTALLY AND AT TIMES SEEMS TO HAVE FALSE IDEAS SUCH
           AS SUGGESTING THAT HE HAS AN APPLICATION IN FOR WORK
           AT THE WEEKS SCHOOL OR THINKING THAT HIS WIFE IS STILL
           ALIVE (THIS WAS SEVERAL WEEKS AGO).@
 OBJ:
    @S@
    @PATIENT NOTED BY STAFF TO BE ACTING STRANGE AND CONFUSED AT
           TIMES. FOR EXAMPLE, YESTERDAY HE ASKED ONE OF THE
           NURSES SEVERAL TIMES ABOUT A TAG THAT HE THOUGHT WAS
           SUPPOSED TO BE ON THE DOOR. ALSO, PATIENT IS VERY SLOW
           WHEN ASKED TO NAME THE DATE: NEVERTHELESS, HE USUALLY
           MANAGES TO GET AT LEAST THE YEAR AND MONTH CORRECT.
           ORIENTED TO PLACE AND NAME.@
 ASMT:
  @THERE IS A QUESTION OF ORGANIC BRAIN SYNDROME SECONDARY TO
           CHRONIC ALCOHOLISM HERE.@
 PLN-PRO:
  @PSYCHOMETRIC TESTING- TO BE DONE TODAY BY DR.[redacted] GROUP-
           HAVE CALLED@
 RX-GIVEN:
                                             /UN015/ 12:06
  THERAGRAN-M 1CAP. 1/D*
 CON REPLY:
                                             /MD320/ 17:08
    AGREE W/ PROBLEM AS FORMULATED.
    AGREE W/ PLANS AS FORMULATED.
 SX:
  @SOCIAL SERVICE NOTE: RECEIVED CALL FROM [redacted],OEO ALCOHOLISM
           PROGRAM SHORT TIME AGO. MR.[redacted] REPORTS FOLLOWING:
           "I HAVE TALKED WITH DR.[redacted] REGARDING PLANS. DR.
           WOULD LIKE TO SEE PT. ON OUR PSYCHIATRIC UNIT PRIOR TO
           DISCHARGE HOME. DR. AND I BOTH FEEL THAT PT. SHOULD
           NOT GO DIRECTLY HOME FROM MEDICAL UNIT, AS HE WILL GO
           BACK TO OLD PATTERN. DR. DOES NOT WANT V.S.H.
           ADMISSION. " I TOLD MR.[redacted] THAT I WOULD PASS THIS
           INFO. ON TO MEDICAL STAFF . I DID QUESTION WHAT OTHER
           PLAN THEY MIGHT HAVE IN MIND IF OUR PSYCHIATRIC UNIT
           COULD NOT TAKE PT. MR.[redacted] DID NOT HAVE ANY ANSWER AT
           THIS TIME. MR.[redacted] HAS REQUESTED THAT I CALL HIM
           FRIDAY. WILL FOLLOW. @ [redacted]
           [redacted]-SUPERVISOR, FU@ @@
    [redacted] ,SUPERVISOR-MFU@
 RX-GIVEN:
                                             /UN024/ 22:13
  CHLORAL HYDRATE 1GM Q*                     /MD014/ 22:38
```

NOTE: Additional information from consultant.

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:00   PAGE  16_
[redacted]  SBHOLD2 543-507-8 M 49

     IMPROVING OVERALL.
OBJ:
   @DR. [redacted] HAS SEEN THE PATIENT  AND THE PSYCHOMETRIC TESTS
          HAVE BEEN ADMINISTERED. THE PRELIMINARY RESULTS
          SUGGEST THAT THERE IS SOME INTELLECTUAL IMPAIRMENT,
          PERHAPS MORE THAN COULD BE EXPLAINED MERELY BY AGE,
          BUT THAT THE DEFECT IS NOT SEVERE ENOUGH THAT IT SEEMS
          TO BE IN KEEPING WITH THE FRIENDS' AND RELATIVES
          CONCERNS. FULL REPORT TO FOLLOW.@
```

NOTE: This status is reflected in the selectable list (see page 65) of the patient's problem in terms of being able to look at all problems or those that are GETTING WORSE. It is also possible to look at lists of patients on a ward with problems in a certain subspeciality, problems getting worse or problems in a certain subspeciality getting worse.

```
ASMT:
  @FEEL THAT PATIENT IS STABLE ENOUGH FOR DISCHARGE. WE HAVE REACHED
          THE POINT OF M H B (MAXIMUM HOSPITAL BENEFIT). WILL BE
          FOLLOWED BY MR. [redacted] OFFICF
          OF ECONOMIC OPPORTUNITY. AND BY DR. [redacted]
```

NOTE: Assessment of a problem is always typed in.

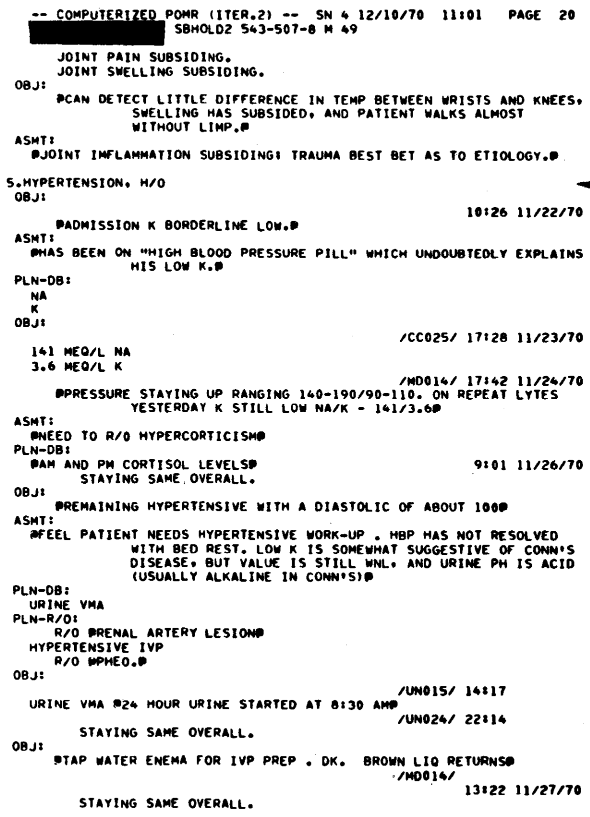
```
 RX-GIVEN:
                                               /UN025/ 10:01 11/26/70
  THERAGRAN-M 1CAP. 1/D*
                                               /UN017/ 21:49
  CHLORAL HYDRATE 1GM Q*
                                               /UN015/ 10:00 11/27/70
  THERAGRAN-M 1CAP. 1/D*                       /MD014/ 13:22
     IMPROVING OVERALL.
OBJ:
   @HAVE CALLED DR. [redacted] WHO FEELS THAT PATIENT CAN BE
          DISCHARGED. HE STATES THAT HE HAS QUESTIONED WHETHER
          THE PATIENT MIGHT BE DISPLAYING SIGNS OF EARLY PICK'S
          DISEASE. BUT SEEMS SATISFYED WITH OUR PSYCHOLOGICAL
          EVALUATION. HE HAS ASKED THAT THE PATIENT HE SENT HOME
          ON PHENOBARBITAL. FOR HE FEELS THAT WHEN HE GETS HOME
          HE WILL PROBABLY BE QUITE ANXIOUS.@
PLN-PAT ED:
  PT.WILL BE TOLD: NATURE OF THE PROBLEM. PROBABLE COURSE W/RX.
          THERAPY PRESCRIBED. @SENT HOME ON PHENOBARBITAL 15 MG.
          QID@
CON REPLY:
                                               /MD320/ 18:22
     AGREE W/ PROBLEM AS FORMULATED.
     AGREE W/ PLANS AS FORMULATED.
SX:
  @SOCIAL SERVICE NOTE: MR. [redacted] ALCOHOLISM SPECIALIST FOR
          O.E.O. IN TO SEE ME PRIOR TO TAKING PT. HOME. MR.
          [redacted] STILL EXPRESSED CONCERN REGARDING FUTURE OF
          PATIENT. HE FEELS THAT PATIENT STILL NEEDS MORE RX FOR
          ETOH PRIOR TO GOING HOME. BUT PATIENT DOES NOT WISH TO
          GO TO HALF WAY HOUSE OR MAPLE LEAF FARM FOR
          ALCOHOLICS. MR [redacted] TO FOLLOW PT. AT HOME AFTER MFU
          DISCHARGE TODAY. NO FURTHER ACTIVITY FROM OUR
          DEPARTMENT AT THIS TIME. CASE INACTIVE AS OF THIS
          DATE. @
     DISCUSSION/ASMNT: @OUTLOOK IS GUARDED FOR PSYCHO-SOCIAL RX OF
          PATIENT AT THIS TIME. MR. [redacted] HOPES THAT HE WILL BE
          ABLE TO GET PT. INVOLVED WITH A.A. PROGRAM. BUT IS FAR
          FROM SURE IF IT WILL WORK. HE FEARS THAT PT. WILL
          RETURN TO OLD PATTERN. @



  -- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:00   PAGE  17
  [redacted]  SBHOLD2 543-507-8 M 49

        [redacted], ACSW SOCIAL WORK SUPERVISOR- MFU@
```

NOTE: Another entry by the consultant.

```
                      <PLANS - INITIAL>
                   ------------------------------

2.(LAENNEC'S) CIRRHOSIS:
 PLN-OB:
                                               /MD014/ 19:28 11/19/70
   SER.BIL.T/D
   S.PROT.ELECTR.
   SGPT
   ALK.P
   PRO T.

                       <PROG NOTES>
                   ------------------------------

2.(LAENNEC'S) CIRRHOSIS:
 OBJ:
                                              /CC025/ 15:51 11/20/70
   TOT:0.7MG% DIR:MG%  SER.BIL.T/D @LESS THAN 0.1 MG%@
   13 UNITS.  ALK.P
   12.1 SEC.: >50% OF CONTROL.  PRO T. @12.2 SEC.@
                                                   22:52
   50.0, 6.2, 14.6, 14.6, 14.6,  S.PROT.ELECTR. / %ALB. %A1. %A2. %B.
            %G. @TOTAL PROT=6.2 GM%@
   10 UNITS.  SGPT
                                              /MD014/  9:46 11/21/70
     @ @
     @ALKALINE PHOS. 43 RATHER THAN 13 AS RECORDED.@
ASMT:
  @LFT'S SUGGEST MINIMAL. IF ANY DYSFUNCTION AT PRESENT.@
```

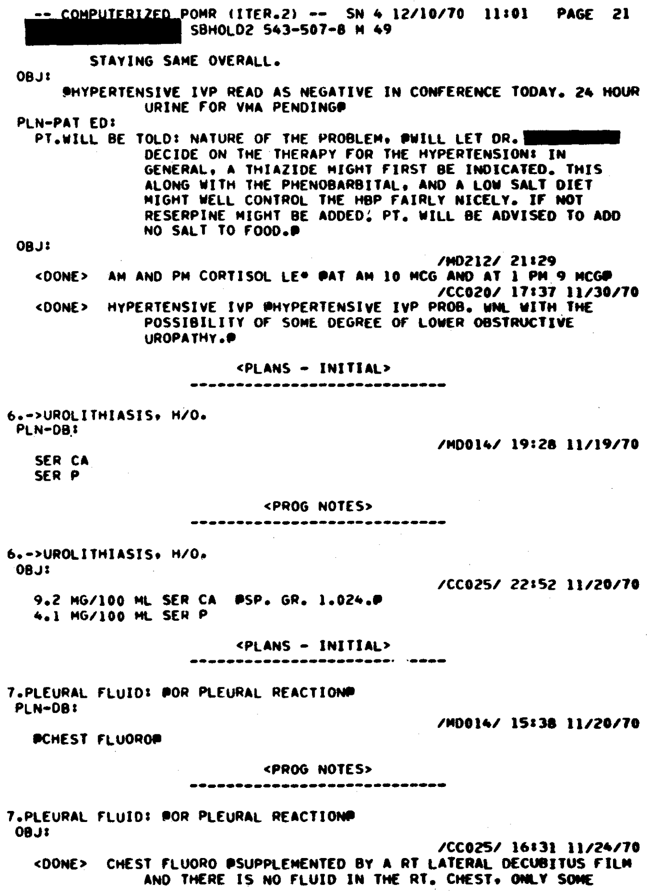
NOTE: Since the information in the patient's record (as stored by the STORE program) cannot be deleted, any mistakes in entering information must be corrected by entering and noting the correct information. In this case, the alkaline phosphatase value was originally entered incorrectly.

```
                      <PLANS - INITIAL>
                   ------------------------------

3.EDEMA, PITTING, SECONDARY TO: NOT SPECIFIED
 PLN-R/O:
                                              /MD014/ 19:28 11/19/70
     R/O @SECONDARY TO HEART DISEASE@
     R/O @SECONDARY TO RENAL DISEASE@
     R/O @SECONDARY TO RENAL DISEASE@
   @HAVE UA, BUN, PEP ORDERED@
     R/O @SECONDARY TO LIVER DISEASE@
   @LFT'S ORDERED@
     R/O @ARTHRITIS@

                       <PROG NOTES>
                   ------------------------------

3.EDEMA, PITTING, SECONDARY TO: NOT SPECIFIED
 PLN-PPO:
                                              /MD014/ 21:22 11/19/70
   @ELEVATE LEGS. JOBST STOCKINGS@
   -- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:01   PAGE  18
   [redacted] SBHOLD2 543-507-8 M 49

  OBJ:
                                                            17:42 11/24/70
       @EDEMA OF ANKLES CLEARING@
  ASMT:
    @ @ @ETIOLOGY OF EDEMA SOMEWHAT UNCLEAR WITH SERUM PROT. OF 6.2
                AND ESSENTIALLY NL. CHEST FILM, LOW BUN, ESSENTIALLY
                NL. LFT'S. IN VIEW OF RE @ @ETIOLOGY OF EDEMA SOMEWHAT
                UNCLEAR SINCE LFT'S, BUN, CHEST FILM ALL ESSENTIALLY
                WNL. SERUM PROTEIN OF 6.2 GMS. HOWEVER EDEMA CLEARING
                AT PRESENT, AND ALSO EVIDENCE OF ARTHRITIS L ANKLE,
                PROBABLY DUE TO SPRAIN@

                          <PLANS - INITIAL>
                      ----------------------------

 4.INFLAM. ARTICULAR DIS: INVOLVING: WRIST/CARPALS RT. KNEE, LT.
                SECONDARY TO:
  PLN-R/O:
                                                  /MD014/ 19:28 11/19/70
       R/O @RA@
    LATEX FIX.
       R/O @RF@
    ASLT
       R/O @GOUT@
    URIC AC.
       R/O @INFECTIOUS (DOUBT IN VIEW OF ABSENCE OF FEVER)@
       R/O @TRAUMA@
    X-RAY: WRIST, RT.
    X-RAY: KNEE LT.

                            <PROG NOTES>
                      ----------------------------

 4.INFLAM. ARTICULAR DIS: INVOLVING: WRIST/CARPALS RT. KNEE, LT.
                SECONDARY TO:
  PLN-DB:
                                                  /MD014/ 21:22 11/19/70
    @BLOOD CULTURES TO BE COLLECTED- THREE SETS TOTAL (ONES AT AM
                DRAWING, ONE AT 1PM DRAWING, ONE AT AM DRAWING SAT.)@
       CONTINGENCY: @IF OTHER STUDIES NEGATIVE@
       CONTINGENCY: @IF OTHER STUDIES NEGATIVE WILL ASPIRATE LEFT KNEE
                JOINT@
  PLN-PRO:
    @HOT PACKS TO LEFT KNEE, RIGHT WRIST FOR 20 MIN. PERIODS TID.@
  RX-GIVEN:
                                                  /UN020/ 10:58 11/20/70
    <NONE>  HOT PACKS TO LEFT KNE* @PT. REFUSED HOT PACKS THIS AM.@
                                                  /UN025/ 21:17
    HOT PACKS TO LEFT KNE*
  OBJ:
                                                  /CC025/ 22:52
    5.0 MG/100 ML URIC AC.
                                                           0:58 11/22/70
    <DONE>  X-RAY: WRIST, RT. @AP AND LAT VIEWS DEMONSTRATE SMALL


   -- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:01   PAGE  19
   [redacted] SBHOLD2 543-507-8 M 49

                CYSTIC AREAS OF THE CARPAL BONES AND POSSIBLY SOME
                SCLEROTIC REACTION. THERE IS ALSO OVERLYING SOFT
                TISSUE SWELLING AND THE PICTURE WOULD BE COMPATIBLE
                WITH RHEUMATOID ARTHRITIS, POSSIBLY GOUT.@
    <DONE>  X-RAY: KNEE LT. @L. KNEE DEMONSTRATES A SMALL FUSION IN
                THE SUPRAPATELLAR BURSA AND POSSIBLY POSTERIORLY.
                THERE IS MAINTENANCE ON THE ARTICULATING SURFACES AND
                NO SUGGESTION OF CYSTIC CHANGES. THERE IS HOWEVER
                SMALL LINEAR STREAK OFCALCIFICATION OF SOFT TISSUE
                WHICH COULD REPRESENT SOME CALCIFICATION OF THE
                POPLITEAL ARTERY. @
    NEGATIVE.  LATEX FIX.
    REACTIVE.  ASLT @40 TODD UNITS.@
  SX:
                                                  /MD014/ 10:26
       JOINT PAIN GETTING BETTER.
  OBJ:
       @RIGHT WRIST STILL SLIGHTLY SWOLLEN, HEAT DIFFERENCE NOT SO
                MARKED, NOT SO TENDER.@
       EXAM OF KNEE(S): LEFT, SWELLING: UNCHANGED.
  ASMT:
    @WRIST FILM ON RIGHT WOULD BE CONSISTENT WITH EITHER GOUT OR RA.
                LATEX FIX. NEG. BUT THIS DOESN'T RULE OUT RA. URIC
                ACID NORMAL - DOUBT GOUT AS ETIOLOGY. CONDITION
                IMPROVING AT PRESENT, MAKING INFECTIOUS PROCESS
                UNLIKELY. MUST STILL CONSIDER THAT PATIENT MAY WELL
                HAVE MERELY INJURED THESE JOINTS WHILE IN AN ALCOHOLIC
                STUPOR. @
  RX-GIVEN:
                                                  /UN025/ 21:44
    HOT PACKS TO LEFT KNE*
                                                  /UN022/ 14:17 11/23/70
    HOT PACKS TO LEFT KNE*
  OBJ:
                                                  /MD014/ 17:42 11/24/70
       @WRIST AND KNEE MUCH BETTER. NOW COMPLAINS OF SORENESS L ANKLE.
                EDEMA PRESENT ON BOTH ANKLES HAS CLEARED.@
  ASMT:
    @SUSPECT THAT THESE JOINT PAINS RELATED TO TRAUMATIC LESIONS -
                PATIENT ACTUALLY DESCRIBES SPRAINING L KNEE AND ANKLE
                AT SAME TIME WHEN HE FEEL DOWN.@
  RX-GIVEN:
                                                  /UN020/ 18:45
    HOT PACKS TO LEFT KNE*                        /MD014/  9:01 11/26/70
          IMPROVING OVERALL.
  SX:
       JOINT PAIN SUBSIDING.
  ASMT:
    @NORMAL URIC ACID, ABSENCE OF FEVER, LOW ASOT, NEG. LATEX FIX. ,
                GRADUAL SUBSIDANCE OF SYMPTOMS ALL IN FAVOR OF
                TRAUMATIC ETIOLOGY, BUT CORRECTED SED RATE 29. RA
                STILL CANNOT BE RULED OUT.@              10:42 11/27/70
          IMPROVING OVERALL.
  SX:
```



```
 -- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:01   PAGE  20
[redacted]        SBHOLD2 543-507-8 M 49

     JOINT PAIN SUBSIDING.
     JOINT SWELLING SUBSIDING.
 OBJ:
     @CAN DETECT LITTLE DIFFERENCE IN TEMP BETWEEN WRISTS AND KNEES.
              SWELLING HAS SUBSIDED. AND PATIENT WALKS ALMOST
              WITHOUT LIMP.@
 ASMT:
   @JOINT INFLAMMATION SUBSIDING: TRAUMA BEST BET AS TO ETIOLOGY.@

5.HYPERTENSION. H/O
 OBJ:
                                                     10:26 11/22/70
     @ADMISSION K BORDERLINE LOW.@
 ASMT:
   @HAS BEEN ON "HIGH BLOOD PRESSURE PILL" WHICH UNDOUBTEDLY EXPLAINS
              HIS LOW K.@
 PLN-DB:
   NA
   K
 OBJ:
                                             /CC025/ 17:28 11/23/70
   141 MEQ/L NA
   3.6 MEQ/L K
                                             /MD014/ 17:42 11/24/70
     @PRESSURE STAYING UP RANGING 140-190/90-110. ON REPEAT LYTES
              YESTERDAY K STILL LOW NA/K - 141/3.6@
 ASMT:
   @NEED TO R/O HYPERCORTICISM@
 PLN-DB:
   @AM AND PM CORTISOL LEVELS@                        9:01 11/26/70
        STAYING SAME OVERALL.
 OBJ:
     @REMAINING HYPERTENSIVE WITH A DIASTOLIC OF ABOUT 100@
 ASMT:
   @FEEL PATIENT NEEDS HYPERTENSIVE WORK-UP . HBP HAS NOT RESOLVED
              WITH BED REST. LOW K IS SOMEWHAT SUGGESTIVE OF CONN'S
              DISEASE. BUT VALUE IS STILL WNL. AND URINE PH IS ACID
              (USUALLY ALKALINE IN CONN'S)@
 PLN-DB:
   URINE VMA
 PLN-R/O:
     R/O @RENAL ARTERY LESION@
   HYPERTENSIVE IVP
     R/O @PHEO.@
 OBJ:
                                             /UN015/ 14:17
   URINE VMA @24 HOUR URINE STARTED AT 8:30 AM@
                                             /UN024/ 22:14
        STAYING SAME OVERALL.
 OBJ:
     @TAP WATER ENEMA FOR IVP PREP . DK.  BROWN LIQ RETURNS@
                                            /MD014/
                                                     13:22 11/27/70
        STAYING SAME OVERALL.
```

NOTE: Initial plans should have been specified for this problem.



```
 -- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:01   PAGE  21
[redacted]        SBHOLD2 543-507-8 M 49

        STAYING SAME OVERALL.
 OBJ:
     @HYPERTENSIVE IVP READ AS NEGATIVE IN CONFERENCE TODAY. 24 HOUR
              URINE FOR VMA PENDING@
 PLN-PAT ED:
   PT.WILL BE TOLD: NATURE OF THE PROBLEM. @WILL LET DR. [redacted]
              DECIDE ON THE THERAPY FOR THE HYPERTENSION: IN
              GENERAL. A THIAZIDE MIGHT FIRST BE INDICATED. THIS
              ALONG WITH THE PHENOBARBITAL. AND A LOW SALT DIET
              MIGHT WELL CONTROL THE HBP FAIRLY NICELY. IF NOT
              RESERPINE MIGHT BE ADDED; PT. WILL BE ADVISED TO ADD
              NO SALT TO FOOD.@
 OBJ:
                                             /MD212/ 21:29
   <DONE>  AM AND PM CORTISOL LE* @AT AM 10 MCG AND AT 1 PM 9 MCG@
                                             /CC020/ 17:37 11/30/70
   <DONE>  HYPERTENSIVE IVP @HYPERTENSIVE IVP PROB. WNL WITH THE
              POSSIBILITY OF SOME DEGREE OF LOWER OBSTRUCTIVE
              UROPATHY.@

                        <PLANS - INITIAL>
                   ----------------------------

6.->UROLITHIASIS. H/O.
 PLN-DB:
                                             /MD014/ 19:28 11/19/70
   SER CA
   SER P

                          <PROG NOTES>
                   ----------------------------

6.->UROLITHIASIS. H/O.
 OBJ:
                                             /CC025/ 22:52 11/20/70
   9.2 MG/100 ML SER CA  @SP. GR. 1.024.@
   4.1 MG/100 ML SER P
                        <PLANS - INITIAL>
                   ----------------------------

7.PLEURAL FLUID: @OR PLEURAL REACTION@
 PLN-DB:
                                             /MD014/ 15:38 11/20/70
   @CHEST FLUORO@

                          <PROG NOTES>
                   ----------------------------

7.PLEURAL FLUID: @OR PLEURAL REACTION@
 OBJ:
                                             /CC025/ 16:31 11/24/70
   <DONE>  CHEST FLUORO @SUPPLEMENTED BY A RT LATERAL DECUBITUS FILM
              AND THERE IS NO FLUID IN THE RT. CHEST. ONLY SOME
```

```
-- COMPUTERIZED POMR (ITER.2) --  SN 4 12/10/70  11:01   PAGE  22
[redacted]      SBHOLD2 543-507-8 M 49

           BLUNTING OF THE COSTOPHRENIC ANGLE. ■
                                          /MD014/ 17:42
     ■CHEST FLUORO. AND LL DECUBTUS REVEAL THAT THE X-RAY FINDINGS
           WERE THOSE OF AN OLD PLEURAL SCAR- NO FLUID■

8.LOW HEMATOCRIT/HGB                                22:38 11/25/70
        STAYING SAME OVERALL.
 OBJ:
     ■REPEAT HB/HCT 11.4/36 IN A PATIENT  WITH PAST HX OF "ZIEVE'S
           SYNDROME".HAS COME DOWN FORM 12.9 /37 ON ADMISSION.■
                                                    23:01
        STAYING SAME OVERALL.
 OBJ:
     ■ON REVIEW OF THE INITIAL SMEAR IT APPEARS NORMOCNRONIC.
           NORMOCYTIC TO ME.■
 PLN-R/O:
     R/O ■GI LOSS■
   STOOL HEMATEST->CHART  X3

                  -- RETRIEVAL REQUEST --

        FRINTER  ENTIRE. CYCLED
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

NOTE: Initial plans should have been specified for this problem.

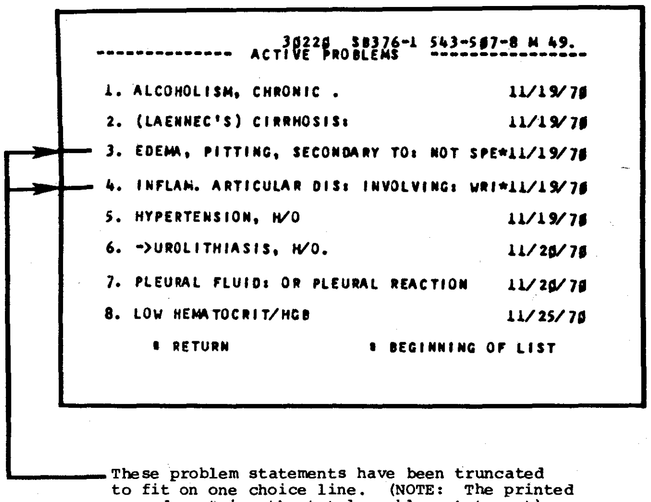
NOTE: Since the results for this order were not recorded, it remains an outstanding order, as seen on the selectable list on page 67.

NOTE: At the end of each retrieval is the complete statement of the retrieval request. This defines clearly what the output represents as there are many different retrieval requests possible.

The first list is the patient's Problem List. This is used to define which problem a Progress Note is to be written on or to define a specific problem for the retrieval of all data in sequence on one problem. The physician or nurse selects the specific problem and then proceeds to enter the data. The last two lists (for therapy and non-therapy orders) are used to indicate when something has been given and to allow the entry of laboratory and x-ray results.

Associated with each patient's computer-based record are these three Intermediate Selectable List Files. They contain copies of information also contained in the record. (Compare the Problem List of the printed record with the one on the screen and the Plans for problem # 8 in the printed record with the General Ward and non-therapy order list in selectable form on the cathode ray tube.) The data are stored redundantly in the lists to facilitate speedy retrieval.

THE PATIENT'S PROBLEM LIST IN SELECTABLE FORM

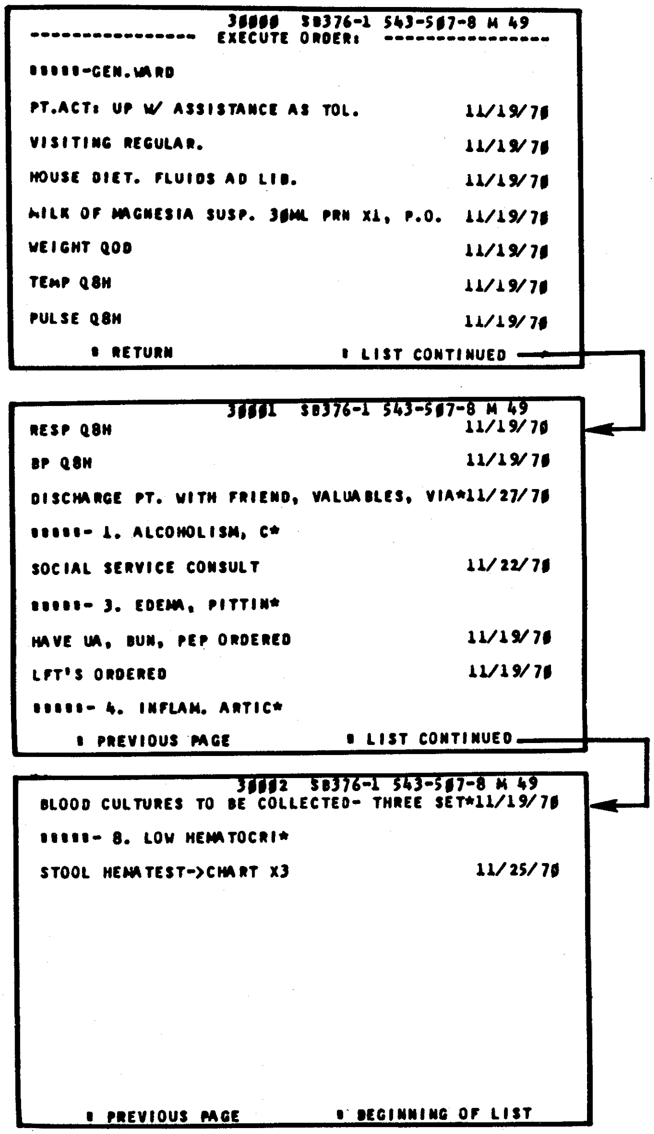


```
                       3Ø22Ø  SØ376-1 543-5Ø7-8 M 49.
--------------  ACTIVE PROBLEMS   ----------------

 1. ALCOHOLISM, CHRONIC .                   11/19/7Ø
 2. (LAENNEC'S) CIRRHOSIS:                  11/19/7Ø
 3. EDEMA, PITTING, SECONDARY TO: NOT SPE*11/19/7Ø
 4. INFLAM. ARTICULAR DIS: INVOLVING: WRI*11/19/7Ø
 5. HYPERTENSION, H/O                       11/19/7Ø
 6. ->UROLITHIASIS, H/O.                    11/2Ø/7Ø
 7. PLEURAL FLUID: OR PLEURAL REACTION      11/2Ø/7Ø
 8. LOW HEMATOCRIT/HGB                      11/25/7Ø
     ■ RETURN               ■ BEGINNING OF LIST
```

These problem statements have been truncated to fit on one choice line. (NOTE: The printed record contains the total problem statement)

THE PATIENT'S GENERAL WARD AND NON-THERAPY ORDER LIST IN SELECTABLE FORM ON THE CATHODE RAY TUBE SCREEN

NOTE: The problem titles and specific orders have been truncated. In case of ambiguity, it is possible to see the complete problem statement and/or the complete orders by referring back to the printed record or by a specific retrieval to the cathode ray tube terminal.



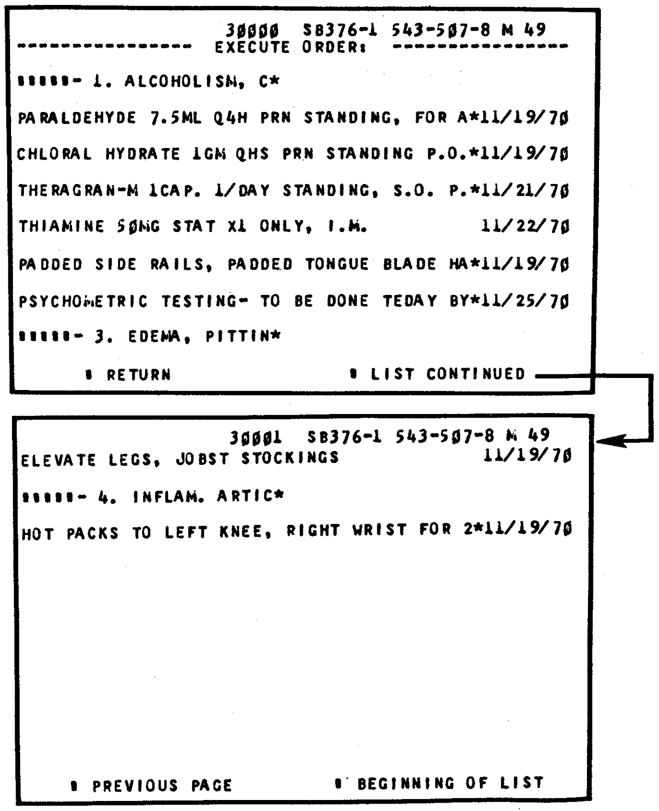
```
                    3ØØØØ  SB376-1 543-5Ø7-8 M 49
----------------  EXECUTE ORDER:  ----------------
▮▮▮▮▮-GEN.WARD
PT.ACT: UP W/ ASSISTANCE AS TOL.                11/19/7Ø
VISITING REGULAR.                               11/19/7Ø
HOUSE DIET. FLUIDS AD LIB.                      11/19/7Ø
MILK OF MAGNESIA SUSP. 3ØML PRN X1, P.O.        11/19/7Ø
WEIGHT QOD                                      11/19/7Ø
TEMP Q8H                                        11/19/7Ø
PULSE Q8H                                       11/19/7Ø
      ▮ RETURN                  ▮ LIST CONTINUED
```

```
                    3ØØØ1  SB376-1 543-5Ø7-8 M 49
RESP Q8H                                        11/19/7Ø
BP Q8H                                          11/19/7Ø
DISCHARGE PT. WITH FRIEND, VALUABLES, VIA*11/27/7Ø
▮▮▮▮▮- 1. ALCOHOLISM, C*
SOCIAL SERVICE CONSULT                          11/22/7Ø
▮▮▮▮▮- 3. EDEMA, PITTIN*
HAVE UA, BUN, PEP ORDERED                       11/19/7Ø
LFT'S ORDERED                                   11/19/7Ø
▮▮▮▮▮- 4. INFLAM. ARTIC*
    ▮ PREVIOUS PAGE             ▮ LIST CONTINUED
```

```
                    3ØØØ2  SB376-1 543-5Ø7-8 M 49
BLOOD CULTURES TO BE COLLECTED- THREE SET*11/19/7Ø
▮▮▮▮▮- 8. LOW HEMATOCRI*
STOOL HEMATEST->CHART X3                        11/25/7Ø




    ▮ PREVIOUS PAGE             ▮ BEGINNING OF LIST
```

THE PATIENT'S THERAPY ORDER LIST IN SELECTABLE FORM

NOTE: The problem titles and specific orders have been truncated yet a complete copy of everything is contained in the record in case of any ambiguities.



```
                    3ØØØØ  SB376-1 543-5Ø7-8 M 49
----------------  EXECUTE ORDER:  ----------------
▮▮▮▮▮- 1. ALCOHOLISM, C*
PARALDEHYDE 7.5ML Q4H PRN STANDING, FOR A*11/19/7Ø
CHLORAL HYDRATE 1GM QHS PRN STANDING P.O.*11/19/7Ø
THERAGRAN-M 1CAP. 1/DAY STANDING, S.O. P.*11/21/7Ø
THIAMINE 5ØMG STAT X1 ONLY, I.M.                11/22/7Ø
PADDED SIDE RAILS, PADDED TONGUE BLADE HA*11/19/7Ø
PSYCHOMETRIC TESTING- TO BE DONE TEDAY BY*11/25/7Ø
▮▮▮▮▮- 3. EDEMA, PITTIN*
    ▮ RETURN                    ▮ LIST CONTINUED
```

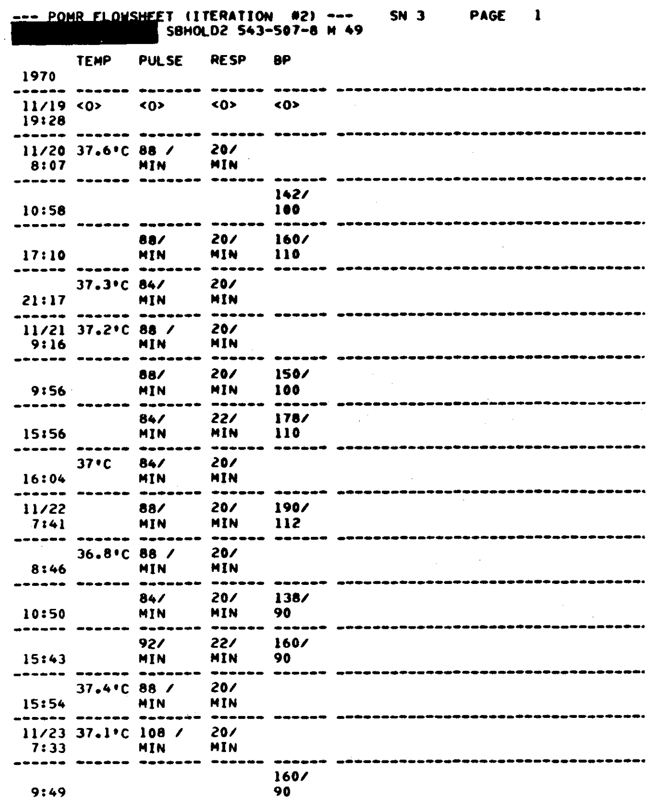
```
                    3ØØØ1  SB376-1 543-5Ø7-8 M 49
ELEVATE LEGS, JOBST STOCKINGS                   11/19/7Ø
▮▮▮▮▮- 4. INFLAM. ARTIC*
HOT PACKS TO LEFT KNEE, RIGHT WRIST FOR 2*11/19/7Ø




    ▮ PREVIOUS PAGE             ▮ BEGINNING OF LIST
```

```
--- POMR FLOWSHEET (ITERATION  #2) ---    SN 3
[redacted] SBHOLD2 543-507-8 M 49
       SGPT    SER.BIL.T/   PRO T.           ALK.P
1970           D
------ ------- ------------ ---------------- -------  -------------
11/19 <O>     <O>          <O>              <O>
19:28
------ ------- ------------ ---------------- -------  -------------
11/20         TOT:0.7MG% D 12.1 SEC.: >50%  13 UNIT
15:51         IR:MG%       OF CONTROL.      S.
------ ------- ------------ ---------------- -------  -------------
       10 UNIT
22:52  S.

     PRINTER: 15 INCH WIDE  HEPATIC PARAMETERS: ALL  FOR PRESENT
                                                       ADMISSION
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

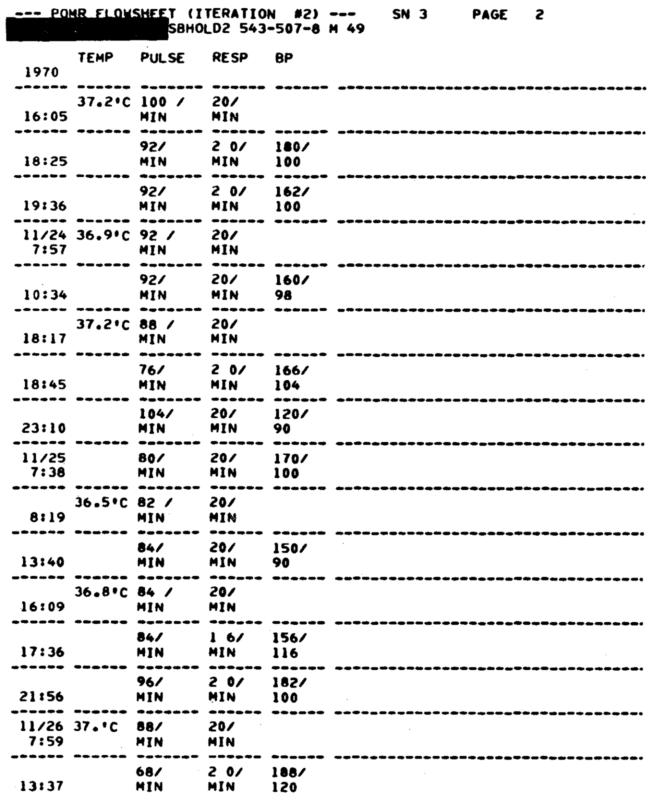
NOTE: This is a flowsheet on all the hepatic parameters for the patient. Note that the <0> is when the treatment was ordered. The following pages contain a flowsheet of all the vital signs on the patient. For both flowsheets, the information displayed can be seen in narrative form in the preceding record.

CONCLUSION

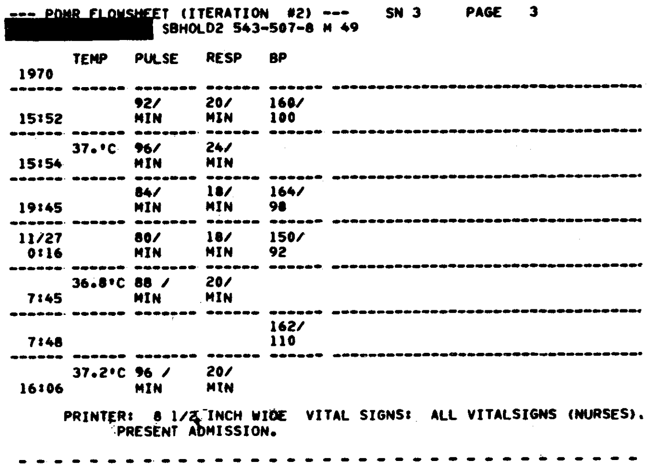
An experimental time-shared medical information system has been developed upon the organizing principle of the Problem Oriented Medical Record. The boundaries of the system are defined by the content of the patient's Problem Oriented Medical Record and all the information that is a natural extension of that document, e.g., the billing, the pharmacy, laboratory and X-ray data, the admitting office information, etc.



```
--- POMR FLOWSHEET (ITERATION  #2) ---     SN 3     PAGE   1
[redacted] SBHOLD2 543-507-8 M 49

       TEMP   PULSE   RESP   BP
 1970
------ ------ ------- ------ ------ ----------------------------------------
 11/19 <0>    <0>     <0>    <0>
 19:28
------ ------ ------- ------ ------ ----------------------------------------
 11/20 37.6'C 88 /    20/
  8:07        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
                             142/
 10:58                       100
------ ------ ------- ------ ------ ----------------------------------------
              88/     20/    160/
 17:10        MIN     MIN    110
------ ------ ------- ------ ------ ----------------------------------------
       37.3'C 84/     20/
 21:17        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
 11/21 37.2'C 88 /    20/
  9:16        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
              88/     20/    150/
  9:56        MIN     MIN    100
------ ------ ------- ------ ------ ----------------------------------------
              84/     22/    178/
 15:56        MIN     MIN    110
------ ------ ------- ------ ------ ----------------------------------------
       37'C   84/     20/
 16:04        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
 11/22        88/     20/    190/
  7:41        MIN     MIN    112
------ ------ ------- ------ ------ ----------------------------------------
       36.8'C 88 /    20/
  8:46        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
              84/     20/    138/
 10:50        MIN     MIN    90
------ ------ ------- ------ ------ ----------------------------------------
              92/     22/    160/
 15:43        MIN     MIN    90
------ ------ ------- ------ ------ ----------------------------------------
       37.4'C 88 /    20/
 15:54        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
 11/23 37.1'C 108 /   20/
  7:33        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
                             160/
  9:49                       90
```



```
--- POMR FLOWSHEET (ITERATION  #2) ---     SN 3     PAGE   2
[redacted]SBHOLD2 543-507-8 M 49

       TEMP   PULSE   RESP   BP
 1970
------ ------ ------- ------ ------ ----------------------------------------
       37.2'C 100 /   20/
 16:05        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
              92/     2 0/   180/
 18:25        MIN     MIN    100
------ ------ ------- ------ ------ ----------------------------------------
              92/     2 0/   162/
 19:36        MIN     MIN    100
------ ------ ------- ------ ------ ----------------------------------------
 11/24 36.9'C 92 /    20/
  7:57        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
              92/     20/    160/
 10:34        MIN     MIN    98
------ ------ ------- ------ ------ ----------------------------------------
       37.2'C 88 /    20/
 18:17        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
              76/     2 0/   166/
 18:45        MIN     MIN    104
------ ------ ------- ------ ------ ----------------------------------------
              104/    20/    120/
 23:10        MIN     MIN    90
------ ------ ------- ------ ------ ----------------------------------------
 11/25        80/     20/    170/
  7:38        MIN     MIN    100
------ ------ ------- ------ ------ ----------------------------------------
       36.5'C 82 /    20/
  8:19        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
              84/     20/    150/
 13:40        MIN     MIN    90
------ ------ ------- ------ ------ ----------------------------------------
       36.8'C 84 /    20/
 16:09        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
              84/     1 6/   156/
 17:36        MIN     MIN    116
------ ------ ------- ------ ------ ----------------------------------------
              96/     2 0/   182/
 21:56        MIN     MIN    100
------ ------ ------- ------ ------ ----------------------------------------
 11/26 37.'C  88/     20/
  7:59        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
              68/     2 0/   188/
 13:37        MIN     MIN    120
```



```
--- POMR FLOWSHEET (ITERATION  #2) ---     SN 3     PAGE   3
[redacted] SBHOLD2 543-507-8 M 49

       TEMP   PULSE   RESP   BP
 1970
------ ------ ------- ------ ------ ----------------------------------------
              92/     20/    160/
 15:52        MIN     MIN    100
------ ------ ------- ------ ------ ----------------------------------------
       37.'C  96/     24/
 15:54        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
              84/     18/    164/
 19:45        MIN     MIN    98
------ ------ ------- ------ ------ ----------------------------------------
 11/27        80/     18/    150/
  0:16        MIN     MIN    92
------ ------ ------- ------ ------ ----------------------------------------
       36.8'C 88 /    20/
  7:45        MIN     MIN
------ ------ ------- ------ ------ ----------------------------------------
                             162/
  7:48                       110
------ ------ ------- ------ ------ ----------------------------------------
       37.2'C 96 /    20/
 16:06        MIN     MIN

    PRINTER:  8 1/2 INCH WIDE  VITAL SIGNS:  ALL VITALSIGNS (NURSES).
         PRESENT ADMISSION.

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The principle focus of our effort has been to directly interface the patient, the physician, and the nurse with the computer in a manner that allows the input and retrieval of *all medical information normally generated by them* for the medical record. The subsequent computer interfaces of the same medical information via message switching with the business office for billing, the pharmacy, the laboratories, admitting office, etc., in a meaningful problem oriented fashion are planned but not yet implemented.

Now that the physician-nurse component has been effectively integrated into a computerized system, final assembly of all the components is possible. In this regard, as Jay Forrester found in management and engineering, we are finding in medicine through the Computerized Problem Oriented Record and its many extensions that the amplification and interactions among the components of the system may at times be more important than the components themselves. We are now prepared to develop a model whereby the ultimate role of the computer in the delivery of health services can be defined.

ACKNOWLEDGMENT

We would like to acknowledge the medical philosophical leadership and clinical expertise of Lawrence L. Weed, M. D.; the clinical experience and expertise of Laura B. Weed, M. D. for the massive effort of developing the bulk of the medical content material (in the branching structured logic displays) in the system; the development of the drug displays by George E. Nelson, M. D.; the early computer software specification efforts of Mrs. Lee Stein; the medical superstructure specification efforts of Charles Burger, M. D.; and the efforts, as represented by the actual patient record, of the house staff and nurses at the Medical Center Hospital, Burlington, Vermont. Important contributions have also been made by many other people whom we wish to thank for their efforts. The authors would like to state explicitly that the computer software described in the paper would be but a skeleton without the medical content material contained in the branching displays.

REFERENCES

1 L L WEED
Medical records, medical education and patient care
Case Western Reserve University Press Cleveland Ohio 1969

2 L L WEED
Medical records that guide and teach
New England Journal of Medicine Volume 278 pp 593-600 652-657 1968

3 M D BOGDONOFF
Clinical science
Archives of Internal Medicine Volume 123 p 203 February 1969

4 R M GURFIELD J C CLAYTON JR
Analytic hospital planning: A pilot study of resource allocation using mathematical programming in a cardiac unit
RAND Corporation RM-5893-RC April 1969

5 MASSACHUSETTS GENERAL HOSPITAL
Memorandum nine
Hospital Computer Project (Status Report) February 1966

6 C M CAMPBELL
Akron speeds information system slowly
Modern Hospital Volume 104 p 118 April 1965

7 R W HAMMING
One man's view of computer science
Journal of the Association for Computing Machinery Volume 16 pp 3-12 January 1969

8 CONTROL DATA CORPORATION
SHORT operating system program rules reference manual
Publication Number 60259600 April 1968

9 CONTROL DATA CORPORATION
SHORT operating system basic formats reference manual
Publication Number 60259700 February 1969

10 CONTROL DATA CORPORATION
SETRAN selectable element translator reference manual
Publication Number 60249400 May 1968

11 L L WEED
Technology is a link not a barrier for doctor and patient
Modern Hospital pp 80-83 February 1970

12 L L WEED et al
The problem oriented medical record—Computer aspects
(A supplement to *Medical Records, Medical Education and Patient Care*)
Dempco Reproduction Service Cleveland Ohio 1969

Laboratory verification of patient identity

by SAMUEL RAYMOND, LESLIE CHALMERS and WALTER STEUBER

Hospital of the University of Pennsylvania
Philadelphia, Pennsylvania

In considering the installation of a computer-based laboratory report system, what are the legal and professional responsibilities created by such systems? Computer-generated reports and records are acceptable, legally, in place of the original handwritten laboratory request form, but there is nevertheless an increased legal duty, as well as a strong professional responsibility, to see to it that the computer record is correct in every detail. By adoption of proper verification procedures, similar in principle to quality control procedures now regarded as essential in every laboratory, a computer-based record system can be made much more accurate and reliable and far more accessible than the usual manual methods of record-keeping. And this can be done without substantially increasing the burden of laboratory work.

Preliminary studies of laboratory requests coming into our laboratory before installation of a computerized report system showed that over 20 percent of the requests carried patient identification data unacceptable by objective standards of verifiability: this means that patient information was incomplete, so that it could not be verified, or wrong, so that verification would lead to the wrong patient. And in a manual system, there appeared to be no practical way to improve the data. In studying other laboratories we found no reason to believe that our experience was unusual.

Our manual system involved returning the *original request* to the referring physician or other source, with the laboratory report transcribed on to it. This placed the identity of laboratory data in the hands of ward clerks and others outside the control of the laboratory; and for a significant fraction of the laboratory work, left the laboratory without any valid record of the work done on the patient for whom it was done. This was clearly an unsatisfactory state of affairs, although one which was hidden until we began our preliminary analyses for the computer system.

The problem of errors in data input was, in fact, by far the most serious one encountered in the development of our computer report system, and one which was essentially out of control until the completion of the work described in this paper. Beyond a few references to slang expressions (GIGO) this is a topic that is discussed very little in the published literature. A recent well-documented monograph,[1] for example, contains no reference to this topic in index, bibliography, or text. Yet our experience has been that every source of raw data, and every transcription step in the data processing operation, carries an error rate of one to two percent. Furthermore, unless these errors are sought for and corrected, they are additive. Since there may be ten or more identifiable steps involved in ordering a laboratory test on a specific patient, errors accumulate until finally as many as 20 percent of the requests received in the laboratory are unacceptable in some particular of patient or specimen identification. That has in fact been our experience over the last four years, and there has been no significant downward trend in the 20 percent figures during this time.

Part of the reason for this has been the policy, which we adopted at the beginning of our data processing development, of not attempting to make any change in the procedures of data processing used outside the laboratory. Although the problem can, and perhaps should, be considered in its entirety as embracing the whole hospital, to do so immediately involves admissions policy, patient accounting practices, the medical record room, in fact almost every aspect of hospital operation. Faced with that much, we decided it would be better to solve some small definable segment of the problem, and the boundaries of the laboratory seemed a convenient place to draw the line.

In our system, consequently, the hospital staff are still free to make occasional errors in preparing a request for laboratory service; we have accepted for ourselves the responsibility for finding and correcting all errors before sending out our reports.

The principal object in our error-correcting system is to make sure that laboratory results get back to the correct patient, and to keep one patient from getting another's results. To perform and report an extra or unordered test—this cannot do very much harm; at most it may result in an extra charge on the patient's bill. To miss doing a test which was ordered—this is a situation which very quickly corrects itself by way of an indignant phone call from the physician who ordered the test. But to report results to the wrong patient—this is the ultimate disaster. Such a mistake could be, quite literally, fatal. It could cause a wrong diagnosis to be made or a wrong treatment to be given. Even if the physician receiving the report recognizes that the results reported do not apply to his patient, he will inevitably place all his reports in the suspect category, and will lose all confidence in the computer report system. And of course, there is the question of legal liability for negligent errors.

Our system, therefore, is designed to assure us of 100 percent correctness in the patient identification, and to provide us with an exact record of every input, a separate record of every wrong input, and every step taken to correct the input. The system requirement is satisfied only if there is verification of every input in two independent traces from the original source of the data. This includes both data from outside the laboratory, specifically patient identification, and data generated inside the laboratory, specifically the laboratory result.

Even within the laboratory itself, where the data processing procedures are under our direct supervision, there seems to be no possibility of eliminating errors completely: no degree of motivation or discipline that can be applied will suffice. Outside the laboratory, where conditions are completely beyond our control, not even an attempt can be made. This is a fact of life which justifies the policy decision referred to above. Our objective, then, is to catch the errors and to correct them before the reports leave the laboratory.

Before developing the details of our system it is useful to introduce three terms which may well have been applied by others earlier in this field; they have certainly clarified our thinking and our system.

Audit trail is a printed record, with necessary annotations, of every record entered into the computer, any errors which were found, and the actions taken to correct them. The tendency, all too human, to bury the error once it has been corrected must be strongly resisted. The audit trail is an essential part of the system; it has both prospective and retrospective functions. It is a necessary record for every input, as from this record alone can one identify later-discovered errors and can know how and where to correct them. It identifies problem areas in the input procedure in the same way that a quality control chart gives warning of procedural error in the laboratory. With the audit trail, any error which does get through the check system, but is later brought to the attention of our staff, can be identified as to occurrence and responsibility. The identification of the source of the error has proved essential in developing psychological defenses within our staff against repetition of the error at a later time. The varieties of error are infinite.

Checking and *validation*: To distinguish between these processes, which are essentially different, is to provide a logical basis for the design of an error-proof input system. *Checking* is an entirely mechanical process, although it may be very complex. It can be carried out by computer, only providing that a set of logical decision rules can be given. *Validation* is a human process, involving human judgment, which changes as each new experience is assimilated. Checking is always catching up to validation, as the judgment is analyzed and formulated into logical rules, which can be programmed into the computer. Validation is always ahead of checking, as judgment is always being increased by experience.

Checking should be done only by the computer. To give this task to a human is wasteful and inefficient and in the end impossible, because, when boredom and fatigue set in, the rate of human error increases faster than the check errors caught. The computer can perform with unrelenting accuracy any checks of any complexity, once the decision tables are specified with logical precision.

The computer can never replace professional judgment.

A necessary condition of the validation process is that the information to be validated must be *meaningful*. That is to say, the data must not only have meaning in themselves, but they must be presented to the mind in a form which conveys meaning to the validator. It follows that a mere list of numbers cannot be validated (Figure 1); the validation in this case consists of noting that two numbers do not match. Did you, as a careful reader, pick up this point?

One final feature which we believe contributes substantially to the efficiency and completeness of our system: each step is conceptually separate and distinct from the others. This is a principle taught by experience in programming a computer. In practice, it means that we demand of computer or of operator only one type of checking or validation at a time. It enables us to see, not only that a given step depends on the successful completion of a precedent step, but, more importantly, that in many places, the system allows of parallel paths through to the final error-corrected report.

In summary, our system is built upon the following rules:

1. The computer is always right.
2. The input is wrong until confirmed by checking or validation.
3. Checking is done by computer.
4. Validation must be meaningful.
5. Two independent sources are required for checking or validation.
6. One thing at a time.

With a basis thus established, we are now ready for the development of a system for controlling input errors. In accordance with standard literary conventions of scientific publication,[2] we shall omit any description of the initial fumblings, outright mistakes, false trails, and utter disasters which we encountered during the first three years of this project. Starting from scratch, our system has now been in full successful operation for nearly a year. We shall describe it as an orderly logical progression of ideas and events. Like most successful systems, it looks very easy now. Had we known just how to do it, it would have taken us three months instead of three years.

In describing our input validation system, the following outline will be useful:

1. Enter census information
*2. Verify census file
3. Enter test requests
4. Print work documents for laboratory
*5. Verify
6. Keypunch lab results
*7. Verify test runs
8. Print patient reports
*9. Validate reports.

As this outline shows, most of the steps in our overall procedure can be carried out in parallel. While we insist that every entry of data into the system be independently verified, we also recognize that most of the information is correct as it goes in. Therefore, we go ahead and use the input in our laboratory system before

ACCESSION NUMBER	CENSUS NUMBER	REQUEST NUMBER
801	312486	312486
802	236925	236925
803	318470	381470
804	334862	334862

Figure 1—Validity check of patient identification numbers

it has been verified or corrected, but always under conditions which do not permit the release of unverified or uncorrected data, and which do permit the exact and secure correction of any errors before they can possibly damage the system.

The system starts with the control of patient identification information, which includes name, hospital unit record number, date of admission, hospital location, age and genetic data, physician, and hospital service assigned. The nominal source of this information is the patient himself, but in accordance with our principle of independent verification, we require two independent traces back to the original source. Fortunately, the hospital operating system does provide two: one from admission desk through hospital accounting office to patient census, and one from admission desk through the patient's chart and his charge plate.* From the first of these we receive a punched-card deck on Monday containing a complete record on each patient in the house, and update decks each day containing patient admissions, transfers, and discharges. From the second we receive an imprinted laboratory request slip.

Although the data entry starts with the patient at the admission desk, the error entry may begin long before, when a patient number, properly belonging to one patient, is improperly assigned to another. Under an old system, formerly used in this hospital, this particular error was to be corrected if, and as soon as, it was discovered. A new commercial accounting system** recently installed by the hospital has the astonishing feature of *forbidding* any correction of this error. Our experience to date has been too short to demonstrate what effect this rule will have on our system, but it is evident that it will make some of the medical records actively misleading for retrospective studies.

* Our hospital uses a charge plate imprinter system with embossed plastic plates like credit cards, but *without* machine-readable features.

** SHAS, supplied by International Business Machines Corp.

ACCESSION NUMBER	CENSUS NAME	REQUEST NAME
801	TARBELL SUSA	TARBELL SUSAN A.
802	GRICE GEORGE	GRICE G. D.
803	QUIGLEY RALP	JAFFE CHARLES
804		EATON ANDREW

Figure 2—Validity check of patient names

In any case, we accept the census and update decks as input data requiring correction. Our experience has been that one to two percent of the records in each new deck are in error. This is substantially lower than the error rate which would be expected on the basis of the number of steps intervening between the data source and the final record, indicating that substantial efforts at error correction are being made all along the line. Nevertheless, we make a final purge.

We maintain, in our computer, a file of patient identification comprising the current house list plus records of all patients who were in the house at any time in the last two weeks.† This file contains about 2000 names. Our first operation is to combine the current file with the new deck, sort it, and delete all duplicate records from the internal computer file. The cards containing the duplicate records are also ejected from the card file by the computer. An audit trail is generated by printing all the records deleted. This is, of course, a checking operation, not a validating one, and requires only a minute or two of computing and printing time each day.

The duplicate list printed usually contains less than twenty names. These are subjected to a validation by the computer staff. This process may include anything from correction of an obvious misspelling of a patient name to a direct inspection of the original admitting record, according to the judgment of the operator. Most discrepancies are cleared up by a telephone call. The corrected records are re-entered and the process is repeated. Usually the first repetition confirms the accuracy of the corrected census.

It is important to keep in mind just what the accuracy of the corrected census comprises. We have, in effect, a census file in which we are certain that each patient has been assigned a unique hospital unit record number. Each patient record has been compared with every other record, both current and recent past admissions. Every discrepancy has been removed, except one. The exception is the patient who has the same unit record number as some other patient who had been admitted at some more distant past time. This exception makes the record doubtful, to that extent, for retrospective studies. It could be removed by enlarging our file storage to cover all past admissions, but this does not seem worthwhile. Even with this exception, we are sure that no laboratory results can be sent to the wrong patient under the control of a wrong patient number.

† We would like to go back further, but our file space is limited and the indicated interval covers a sufficiently high fraction of our needs.

The next major operation in our system is to enter the laboratory test requests into the computer. The laboratory receives requests in the form of the usual 3-part request form, on which the patient identification is imprinted from the charge plate and the tests selected are indicated by handwritten marks. The request is assigned an accession number, and 6-digit patient number, the particular tests required, and the accession number are transcribed by key-punching on a punch-card* which is used to enter the request into the computer. Note that this is the minimal information which must be entered: the accession number, the patient identification number, and the test requested. If every one of these passes the tests which are now to be applied, the request is accepted without further entry; if any one fails, corrections must be made and additional information must be supplied.

As the test requests are entered into the computer, an audit list is printed, showing the accession number and the patient number. At the same time, the patient number is checked against the census file, and if the patient number is found in the file the corresponding name is printed also; if the patient number is not found, no name is printed (Fig. 2). Usually two to ten percent of the entries are missing a name. These are filled in, often by calling the source of the specimen, so that each entry in the entire entry list has an associated name. The list is then validated by direct comparison of the list, name-by-name, with the names on the request slips. This process is not as burdensome or as time-consuming as it sounds, because the validation is comparing two lists of meaningful names (most names convey some sense of familiarity to a literate person, and even the unusual name is meaningful *ipso facto*) and the two lists are in the same order. The validation is highly significant, since the computer-generated name list comes from the census file, while the name on the request slip comes from the patient's hospital chart or charge plate, or—in a fair proportion of the cases—as a handwritten entry on the request slip. Judgment is required to decide when the two apparently similar names are to be accepted as identical.

Note that this process does not require any operator checking or verifying of the six-digit patient number,

* The mechanics of this transcription will be described in a separate report.

since the validation could fail to turn up an error only in the unlucky coincidence that (1) an incorrect patient number exactly matched a patient number in the census file, and (2) the name in the file, in turn, was identical with the name of the actual patient. Since the patient number is six digits, the chance of the first is on the order of one in a million, and considering the statistical distribution of names, the chance of the second must be one in a thousand: the combined chance of one in a billion is acceptable. At least, this mischance has not shown up yet.

At this point we can be certain that within the above probability, each test request is correctly and unambiguously matched with some individual patient in the hospital, and that the three-digit accession number or the six-digit patient identification number, either one, will unambiguously lead us to the correct inpatient.

As the requests accumulate in the day's run, and interspersed with the above-described verifying activities, we also print, by computer, a series of test checklists of accession numbers arranged by the laboratory work-station which is to handle them. Also, laboratory personnel have been carrying forward preliminary processing on the specimens received. The test checklist is compared, by laboratory personnel, with the specimens being processed, and any discrepancies are reported to the computer staff for correction as necessary. This is the first of several verifications made of the tests ordered on each request. The source document for this information is the test request itself, supplemented by telephone calls from the patient areas, adding to, altering, or cancelling the list of tests requested. Because very little machine checking is possible, and a great deal of human checking and

```
033 1376703 RAV6 GOLDNER EUG
     376703 RAV6 GOLDNER  EUG   563 24 02 W M

034 1376704 RAV6 DUBROW SOL
     376704 RAV6 DUBROW  SOL    622 58 05 W M

038 2336838 WD N MOLDEN MARGA
     336838 WD N MOLDEN  MARG   028 36 28 N F

041 1317132 MAL4 RAYNOR JOHN
     317132 MAL4 RAYNOR  JOHN   799 61 01 W M

043 1366042 GIBS DEEGAN EDWIN
     366042'GIBS DEEGAN  EDWI   725 53 03 W M

052 1329766 NEUR THOMAS JESSI
     329766 NEUR THOMAS  JESS   028 51 20 N F

****  CENSUS ENTRY HAS THE SAME NUMBER BUT NOT THE SAME NAME  ****
055 1346710 MAL4 FREIDMAN LEO
     346710 MAL4 FRIEDMAN  LE   434 55 01 W M

057 1178790 WHI7 LEHR ETHEL
```

Figure 3—Audit trail: duplicate census entries

```
12/03/70
ACC  PATIENT NAME   NUMBER  CLASS
**
316                 376739    1
317  FELDMAN   EVA  365248    1
318  URBAN   MITCH  376713    1———— Schaffnit, Gladys # 376711
319  POMANO PASCA   000000    4
320  FREEMAN   SOP  374916    1     Called Mr. Smith, admissions.
321  MARTIN PERCU   000000    4
322  CALAB ALLEN    000000    4     Two history numbers in
323  SYLVIA BEARM   000000    3
324  VANKOUWENBER   376664    1     the house are the same.
325  WOODARD   MIN  115048    2
326  BASCH   SADIE  113689    1     Changed Schaffnit num-
327  HENRY RALPH    000000    4
328  KROUSE LEONA   000000    4     ber to 376711
329  TARASON EVA    000000    3
*
**

DATE 12/3           TIME 12:00        SIGNATURE Gary
```

Figure 4—Audit trail: verification of patient identity

human verification is required, more human effort, perhaps, is expended in keeping this list correct than in getting the patient identification correct. We regard this as principally a public relations effort, since an extra test or a missing test on a patient's specimen cannot have very serious consequences. Nevertheless, we do what we can.

All the while, preparations are being made for printing the daily worksheets for the laboratory personnel, one of the two main tasks of our computer system. These worksheets carry complete information about every patient specimen—name, number, hospital location, hospital service assigned, doctor code, age, sex, and genetic information. We expect the laboratory personnel who are professionally trained at all levels from staff physician to technician, to notice this information, to take a personal interest in the *people* for whom they are helping to provide medical care, and to notify the computer staff of any technical or data processing discrepancies they may pick up. It is largely due to their alert interest that the last one-tenth percent of errors is corrected which make the difference between success and failure.

Among the subsidiary but useful records which the system generates in this period, which occupies the first two or three hours of the working day, are the master accession list, arranged in numerical accession order, and the alphabetic list of patients. Both of these carry the complete lists of tests entered for each patient. The alphabetic list is highly useful in answering telephone inquiries from the house staff who want to know if they "forgot to order" and like excuses. The provision of this service by the computer has undoubtedly reduced the number of "stat" requests received during the day.

Enough has been said already to illustrate our general approach to input, so that only a brief summary of our input verification of results is necessary. Here we have two distinct problems: (a) the correct transcription of the results from laboratory to computer and (b) the medical significance of the results as reported.

(a) As to the first, we have had no direct personal experience yet with direct on-line acquisition of data from the laboratory analyzers. Our observation of such systems installed in other laboratories leads us to believe that on-line direct data acquisition will raise just as many problems as it solves. In our system, the laboratory technician key-punches her own results direct from her original record. The key-punched cards are checked by another technician or operator. After the key-punched results are entered, the computer prints a reconstruction of the laboratory record, which is used for a second check against the original record. This reconstruction not only affords a second check on the numerical results reported, but especially calls attention to (1) extra results reported which were apparently not called for on the original test requests, and (2) results missing on tests which were originally entered into the computer.

(b) Still more important, however, and one of the principal benefits to be sought from a computer-based report system, is a professional evaluation of the medical validity of the laboratory results reported. In the days before the overwhelming expansion of volume in the laboratory work, every result reported from our laboratory was personally examined by the chief of the laboratory. This protected the laboratory from many embarrassing mistakes and made the results, even with the crude and unspecific tests of those days, more significant medically in many cases than the excessive number of tests which are indiscriminately reported today. It is now humanly impossible for the chief, or even any reasonable number of assistants, to examine attentively and with judgment all the results which are reported on hospital patients today. We need some sifting procedure to separate the laboratory results which are obviously reasonable, or for which no informed judgment is possible, from those which require and would benefit from the attention of an experienced clinical pathologist. No one, for example, can make anything out of a single blood sugar determination on a patient for whom no previous laboratory work has been reported. There is no value in taking up the time and attention of the professional staff on such a report. If, however, the computer is programmed to bring together all the patient's results and to print out, for human attention, the blood sugar which is dubious when compared to other values for that patient, much valuable time could be saved for more productive use. We are just beginning to see the benefits of this approach; it does not properly belong in a discussion of input error checking.

The cost. The elaborate scheme proposed above for human and machine verification of computer data may seem all out of proportion to the benefits it produces. We do not deem it so, even though we cannot estimate either the cost of one negligent error in a laboratory or the added cost of preventing it. But more than this, the verification procedure actually costs us very little. We have run our computerized report system for several years with 98 percent accuracy and some—occasionally extreme—dissatisfaction on the part of the final users of it, i.e., the medical staff. We now run at better than 99.9 percent accuracy in patient identification with no increase in costs. The explanation is that in any clinical laboratory system there are periods of great activity and periods of comparative quiet; the human effort required by the above system of verification can easily fit into the quiet periods. The procedures usually need not be carried out a predetermined time, either in relation to the clock or in relation to the system procedures, so long as they are completed before the first external report is generated. The verification procedures are each one simple in themselves, and they use printed lists and batches of source documents which are in simple orderly relationship to each other, so that there is not a lot of frantic back-and-forth searching involved. Each error that is detected can be pinpointed as to source, occurrence, and effect, and the fear of unknown and unknowable responsibilities has been dissipated. The staff has confidence in the system. It is this fact alone which makes our system a practicable one.

In recent months, following circulation of this paper in preliminary form, the level of input accuracy has risen appreciably. *Post hoc ergo propter hoc.* We have noticed that when the input is 100 percent accurate, our verification system reduces to nothing more than reading each laboratory request slip twice: once when it is transcribed for keypunching and again with the validation list produced by computer. This does not seem unduly burdensome. In pharmacy practice, each prescription is read three times—once when picking up the stock bottle, second when counting out the pills, and third when returning the stock bottle to the shelf. We should do no less than twice in laboratory processing.

REFERENCES

1 D A B LINDBERG
The computer and medical care
C C Thomas Springfield

2 J D WATSON
The double helix
Atheneum N Y 1968

The data system environment simulator (DASYS)*

by LAURENCE E. DeCUIR and ROBERT W. GARRETT

System Development Corporation
Santa Monica, California

INTRODUCTION

For the past nine years, SDC has been Integration Contractor to one of the largest satellite tracking, commanding and control networks in the nation.

During this period the software portion of the total system has increased in both dollars and development time. Software is now a major element in the over-all system cost.

One of the prime factors complicating this situation is the typical requirement to implement such software on a hardware system that is in a parallel state of development. A second is the use of actual mission supporting system hardware for development and integration of new software capabilities. The nature of this problem differs from that of initial over-all system development since the hardware already exists in developed form. However, the interference between normal operations of support hardware and software development can cause major disruptions to both efforts.

Past attempts to alleviate this problem have included a variety of approaches utilizing software only, hardware only, or a combination of the two to simulate system functions in a non-operational environment. Although the results of these efforts have been good, they have also been piecemeal, and the time required in the operational environment has still been painfully large.

The support role and configuration

The mission of the organization for which SDC is Integration Contractor is to acquire, track, command, monitor and recover spacecraft in a multiple satellite environment.

The organization utilizes a very large, extremely complex computer-based general purpose command and control system. The system consists of a central computer complex with associated command functions and a number of tracking stations located throughout the world. An elaborate communications network connects the stations with the central computer complex.

The general purpose portion of the system provides acquisition data to the various tracking stations, acquires the satellite as it passes over, collects telemetry data, transmits commands to the satellite, and provides tracking data used to update the ephemeris for future acquisitions.

The system provides the eyes, ears, nervous system, and muscle to the over-all process.

Each tracking station has one operational computer. This computer handles the tracking, commanding, and telemetry data. It is tied to a buffer computer in the central computer complex, which can be automatically switched from station to station as it follows the orbiting satellite. Orbit ephemeris computations, acquisition predictions, command generation, and other associated functions are performed on off-line computers using data from the buffers. Acquisition and command data are transmitted through the buffers to the tracking stations.

Because of the size and complexity of the computer programs that make up the system, the organization employs a number of separate firms for software development. SDC has contracted to integrate the resulting computer program system.

SDC's integration role

A software system is a host of computer programs: each performs one aspect of a complex job, each must be coordinated and compatible with the others. A system must be complete and must effectively fulfill the requirements for which it was designed and built. To say that a system is complete and effective means that all of its parts fit together, that they cover all of the tasks

* © Copyright System Development Corporation 1970.

necessary to do the total job, that the parts and the system have been tested and proved reliable, and that personnel have been trained to operate the system and keep it running. The process of making a system complete and assuring its quality is called integration.

To comprehend how sizable the integration effort can be, one need only consider the following:

- There are currently five system program models in various stages of work, and there have been as many as seven. Two are operational, one is in the final phase of validation testing prior to installation, one is being coded, and one is in design.
- Each program model consists of approximately 1,600,000 instructions. New models are introduced based on changing requirements.
- Change control (design changes and program changes resulting from detected errors) must be maintained for each model. A typical program model has dozens of design changes and hundreds of discrepancy correcting changes made during its dynamic operational life.
- The system uses three different computers. Conversion to new configurations requires retraining of personnel.
- Interface requirements must be specified for each model. New or modified simulation techniques and tools, so vital in the validation testing activity, must be developed continually.
- Training of operations personnel and on-going support must be provided for continuity and feedback.

The scope of SDC's integration effort varies slightly depending on the schedule for each program model, but in order to provide continuous quality control and assurance, a level of effort concept is clearly mandatory. A discussion of problems encountered in software development in the live environment follows.

OPERATIONAL PROBLEMS

The software developer has a great deal of confidence in his ability to cope with problems which occur in a closed computer and computer peripheral environment. When this environment is extended to include complex, real-time asynchronous acting and geographically remote system elements, his confidence and ability are significantly reduced.

In any network-wide test condition which is set up to exercise software much of the system technical performance is invisible to, and beyond the control of, the software developer or integrator. He will be uncertain as to the actual system element status and will generally rely on a telephone network for some of the information transfer and control. The setup of his test condition will require scheduling which impacts on normal system operations. Test personnel availability and travel to scattered locations are also some significant problem areas. This type of operation may also require extended time at remote locations to run tests, analyze results, and factor system changes back into the software under development.

The use of live vehicles in a system test environment is designed to provide conditions as close to operational as possible; however, even the best of these tests are often analyzed on a statistical basis in an attempt to get the most information out of a very expensive set of test conditions, but here again compromises are made to hold the cost and time involved to reasonable values.

In general it can be said that a command and control system involving real-time operational software must be tested or demonstrated in an environment as close to operational as possible prior to final acceptance. However, these tests or demonstrations are very difficult and costly to design, set up, staff, and control. These problems make it very desirable to reduce as much as possible the system time required for these activities in the operational environment.

After careful evaluation of the cost and utility of providing a complete data environment functional simulation system (test bed) to minimize the problems, SDC built the Station Ground Environment Simulator (STAGES) for the Air Force. The system consisted of interfacing hardware, a simulation computer and the simulation system software. An updated version of STAGES is being installed now.

The Data System Environment Simulator (DASYS) represents a refinement and expansion of the STAGES system on a custom basis to accommodate any simulation requirements.

ALTERNATIVES

The ground rules established for the design and development of the software test bed are:

1. That the total support environment be available (includes telemetry plus tracking and command),
2. That a direct operator interface be provided through the Operator Console,
3. That the operational computer process be as close to real-time as possible,
4. That software being tested be in the exact deliverable condition (no octals required to accommodate test bed peculiarities),

5. That dynamic control of every bit of each word passing through the computer interface be available,
6. That recording capability of the operations be available for analysis,
7. That environment modification require a minimum of software changes.

Two alternatives for a software test bed were considered by the Air Force before selection of SDC's DASYS approach.

1. That the functional support environment be provided to the real-time computer by means of digital simulation (the DASYS approach),
2. That the configuration consist of all support components of the actual tracking station.

The advantages of Software Checkout by simulating the functional support environment are:

1. Complete real-time environment is provided for testing, using dynamic inputs,
2. Implementation and operation costs are significantly less than in a facility using real hardware,
3. The user controls the test environment; there is no requirement for a larger complement of support personnel or extensive communication systems,
4. Versatility is provided by simulation rather than actual components. This allows:
   (a) Practically unlimited computer provided data input parameter control
   (b) Experimentation with alternatives
   (c) Automatic or manual control of environment
   (d) Operation and control by user with minimal support,
5. Computer time, calendar time, and cost for program checkout are reduced,
6. There is ability to repeat specific tests for each program model and automate comparison of results,
7. The system can be readily modified to reflect new interface characteristics,
8. Non-interference between hardware and software subsystem checkout processes is guaranteed through final acceptance,
9. Real-time dynamic and post-test analysis of all computer/hardware interface is available,
10. Real-time program checkout is conducted independent of operational system availability,
11. Simulation of hardware anomalies and future hardware capabilities is available prior to equipment availability,
12. Software checkout and integration support are the sole roles,
13. Additional applications areas are available for:
    (a) Mission rehearsal
    (b) Operator training under nominal and anomalous real-time conditions
    (c) Data system test and exercise.

The only disadvantages of simulating the support environment are:

1. Inability to investigate hardware-specific problems,
2. No checkout of hardware diagnostic types of programs.

The advantages inherent in the actual tracking station configuration were:

1. The system could be used to accept, check out, and integrate hardware to be installed in the tracking stations,
2. Hardware modifications could be tried and tested in a non-operational environment,
3. Hardware diagnostics and other supporting programs could be checked out on a non-interference basis,
4. The equipment components could be used as a training aid for both hardware operators and maintenance personnel,
5. The system would provide a limited operational software checkout capability on operational computers.

The fact that only a limited checkout capability existed in the duplicated tracking station test bed (alternative 2) plus the disadvantages listed below were the reasons for not taking this approach. The disadvantages are:

1. The configuration resembles a tracking station to such an extent that many of the disadvantages of site use for software checkout also apply to the test bed.
   (a) Full-scale preventive maintenance is required for configuration components because real, rather than simulated, hardware is used.
   (b) All engineering changes apply to the components, the same as they would to the tracking stations. This increases system costs and interferes with software checkout.
   (c) Extra manning is required for monitoring and operating the components that have no real bearing on the software subsystem test.

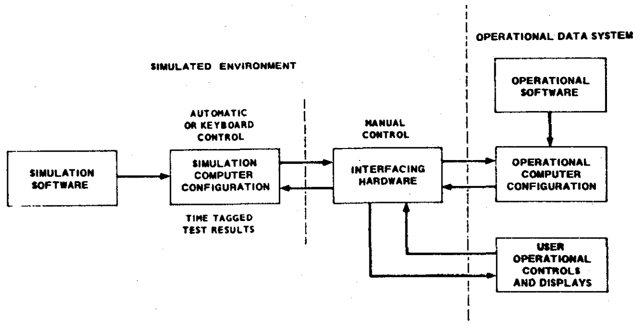


Figure 1—Data system environment simulator

2. The primary role of the test bed would be hardware support. The amount of software checkout time required would all but eliminate this role.
3. Parameter control is severely limited.
4. No capability exists to:
   (a) Checkout completely controlled error conditions for any vehicle command functions
   (b) Run with multiple controlled environment
   (c) Repeat tests under manual control.
5. The absence of system-oriented hardware/software technical interface personnel hinders resolution of problems.

The first alternative was chosen as being the best and most economical solution for developing the software test bed. The entire data system functional environment would be provided by a relatively small simulation computer.

THE DATA SYSTEM ENVIRONMENT SIMULATOR (DASYS)

There were two approaches available for the design of DASYS. First, the simulation computer could be large enough and fast enough to provide information to the operational computer in real-time or near real-time. This would require the simulation computer to be several times larger than the operational computer and the test programs would have to be essentially real-time programs. This approach—although used by several government agencies—did not offer any savings over the conventional method of program testing and evaluation; in fact, the cost of the simulation computer and the cost of writing the real-time programs for this system would cause it to be more expensive than using the actual tracking station.

The second approach was to use a relatively small simulation computer, operating in nonreal-time, and to modify the operational computer to make it operate as if it were in real-time. This is accomplished by stopping the clock in the operational computer while the simulation computer processes the next input stream. Thus, the operational computer is operating in segmented real-time.

General description

The Data System Environment Simulator consists of a small simulation computer with associated peripherals plus simulation software and interfacing hardware (Figure 1). In this system, user operational controls and displays are included in the interfacing hardware.

The interfacing hardware in the system provides all of the externally generated signals and data to the operational computer including user controls in the same manner that the live environment does when interacting with a live moving vehicle.

Since the simulation computer must complete its functions without compromising the timing integrity of the operational computer, the operational computer clock may be stopped before the next input from the simulation computer. This permits operational computer transactions to be performed as if the operational computer were operating in a continuous time mode, and allows the simulation computer to run in nonreal-time. Thus, the environment simulation is not time constrained by the simulation computer.

One part of the simulation system "drives" the environment by functionally simulating data, equipment responses, and operator interactions to whatever level is required for the purpose at hand. Operations in real-time, delayed time, and accelerated time are all possible. During software tests, data rates and complexity may be varied over ranges that are much broader than those in actual operations.

Another part of the simulation system records the results of each run, so that the software's interaction with its environment may be analyzed in detail after the simulation run. The required analytic tools may be run on the simulation computer.

The user can operate the system in any of three ways: there is manual control of the system using the hardware alone, automatic control using previously generated tapes, and semi-automatic control using tapes and manual intervention with the hardware.

The capability to simulate a data system environment is defined generically as data system environment simulation. The parameters of an operational system which must be considered are operational computer I/O channels, word size, and environment functional responses at the computer I/O channels.

Interfacing hardware

The interfacing hardware consists of electronic equipment that connects the simulation computer to the operational computer. The hardware complements the simulation computer by providing functions not feasibly provided by software either in cycle time or economy. It also provides monitoring functions of the data exchange occurring at the operational computer interface by providing all intelligence with time tagging in microseconds for recording on magnetic tape. No modification has to be made to the operational computer data channels due to the matched interface logic provided by the hardware.

General functions provided are: master timing, operational clock control, data recording and time tagging, simulation computer interface, controls and indicators, and environmental functions which can be hardwired.

Operational controls and displays are provided to the user through the interfacing hardware to allow recording of user-generated inputs or inputs from the simulation computer.

Simulation computer

The simulation computer configuration includes appropriate peripherals to provide the programmed functional environment and control signals through the interfacing hardware to the operational computer. The simulation computer is smaller than the operational computer but will transfer data in and out of the interfacing hardware at a rate fast enough to cause the simulated environment to respond to the operational computer stimulus in near real-time. The ratio of operating time to real-time will normally be of the order of 1.5 : 1. The simulation computer configuration can perform utility and off-line functions for the operational computer when the required peripherals are included in the system.

Simulation software

The software operating in the simulation computer will provide nominal environmental inputs and responses to the operational computer unless a specific anomalous condition is programmed into the system. This computer will generally input varying environmental parameters from exercise-specific, system parameter tapes containing data such as track, telemetry and operator inputs, command responses, and communication data. These data will be output to the operational computer with timing integrity. Anomalies can be entered manually from a card reader or keyboard, or automatically from a tape.

The software can cause the printout of selected time-tagged operational computer interface data (during an exercise or after an exercise) from the recording tape which has been written by the simulation computer.

Summary

Using DASYS, a programmer can checkout operational programs in a real-time environment without disrupting system operations or having to involve others. There is considerable versatility in the way he can use both hardware and software. DASYS also permits experimentation with alternative programming approaches and determination of the exact time-tagged operational implication of each. There can be manual control capability of the system, using the simulator hardware alone, or the user can operate from a previously generated environment tape that provides automatic control. This real-time simulation is also practical for mission rehearsals, system tests, and operator training.

DASYS can handle functional simulation requirements for nearly all system elements—sensors, command equipment, telemetry receivers, radars, system operator consoles, timing systems, alarms, and displays. Functional simulation of hardware being developed and objects being or to be supported, sensed, or directed by the software system being tested can also be provided. It makes little difference whether these objects are satellites, missiles, aircraft, railroads, or items being run through processing plants.

COST EFFECTIVENESS

The process of developing, testing, and integrating large computer programs includes detecting and correcting program system errors. This points to a very significant way of measuring the effectiveness of any means to accomplish these tasks.

The parameters involved are numbers of errors, time between locating and correcting errors, and cost to spend system time in locating and correcting errors. Additional considerations include errors remaining in the system, level of confidence, system performance parameters control, and availability of test environment.

The measurement of progress in development of computer programs has been the subject of much study and analysis. The detection and correction of an error constitutes a step in this process. Measures of program status can be related to the rate at which new errors

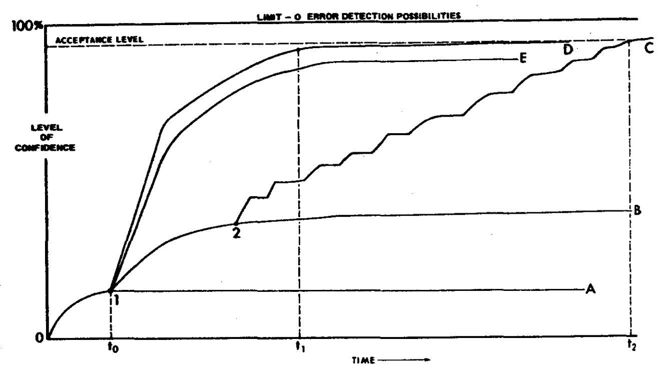


Figure 2—Level of confidence chart

come to light. The lower the error detection rate, the closer the system is to the error-free asymptote.

Figure 2 portrays the method in which software is normally tested and accepted. The vertical axis represents the level of confidence represented as percentage. The top horizontal line represents the limit or 100 percent level of confidence while the horizontal dashed line represents an acceptance level of confidence. When the acceptance level is reached the program or system is usually declared operational and turned over to the maintenance programmers. The horizontal axis represents time from the start of system testing.

The slope of Curves A, B, C, D, and E represents the error detection possibilities. Therefore, any curve with zero slope has zero error detection possibilities. Curve A represents the assembly and testing cycle. The initial error rate is steep since the program is assembled and cycled for the first time but levels off very fast as the limitation of this procedure is reached.

Curve B starts (point 1)when Curve A approaches its limitation and represents the system where the environment is simulated by hardware, function generators, and special purpose simulators. The error detection rate will immediately go up since the capability of simulating various subsystems of the live environment has been introduced. Again the cumulative number of errors detected and corrected cannot exceed the capability of the facility and it is necessary to go to the next facility. This system is limited primarily by the inability to test the total system and the limitations of special purpose hardware, generally analog, which cannot be controlled to the desired precision.

Curve C represents final testing in the live environment. It is a step curve since testing is limited by hardware status, scheduled support of operations, and the inability to repeat the exact set of conditions to test corrections. The length of the horizontal segments of this curve will vary depending on the above factors. The system is accepted when it reaches the acceptance line of the chart (t_2).

Curve D represents the error detection possibilities of using a simulation computer operating in nonreal-time. Due to the fact that it can exercise the entire system and repeat the exact conditions as many times as necessary, it has steep error detection capability up through the acceptance level. Realizing that no one would accept a system that had not operated satisfactorily in the live environment, the acceptance test on the operational system could be accomplished at time t_1.

Curve E represents a simulation computer operating in real-time. The only reason it does not have as high a confidence level as Curve D is that at some time in the testing, the capability of the computer to continue to provide real-time input to the operational computer would be exceeded. Larger and larger systems could be installed to eliminate this problem, but the hardware and software costs would be prohibitive.

There have been no statistics gathered as to how great a savings in time and money can be realized through use of the Data System Environment Simulator. It is estimated that the time savings ($t_2 - t_1$) could be as much as 70 percent and that the dollar savings could easily be 50 percent or more.

Management information systems—What happens after implementation?

by D. E. THOMAS, JR.

Western Electric Company, Incorporated
Greensboro, North Carolina

INTRODUCTION

One of the greatest challenges facing the modern business firm is the development of means to efficiently and prudently harness computer technology, and make it produce. Classical computer production has been directed toward making existing tasks more efficient. A typical indicator of this direction has been the automation of clerical accounting and payroll systems. Recent computer production, however, is becoming increasingly oriented toward automation of the decision making processes themselves (the "what if" question). This latter trend has forced the development of large, fast memory, tremendous on-line storage capabilities, powerful operating systems, and generalized, easy retrieval programming languages.

Furthermore, it is recognized that these hardware and software capabilities alone may not be sufficient to provide the desired results. The operating environment must also undergo change. As the complexity and importance of automated information continues to grow, the role of the classical, job-shop service organization must be abandoned. A vital, dynamic support group of Production System Analysts must be substituted which will be responsible for the management and control of this computer based "information utility."

USE OF MANAGEMENT INFORMATION SYSTEMS

One means by which many firms utilize computing capability is through the development and implementation of management information systems (MIS). MIS attempts to solve the information problem by providing relevant information in the right form to the right person at the right time.[3]

Appropriately, MIS has been defined as "an accessible and rapid conveyor belt for appropriate high quality information from its generator to its user."[3]

The information system should provide not only a confrontation between the user and information, but also, the interaction required for relevant and timely decision making. The heart of an effective MIS is a carefully conceived, designed and executed data base. By fully utilizing MIS, it can become "an intelligence amplifier" and the computer can become an extension of the manager's memory. Ultimately, the computer should free him from routine tasks, providing more time to devote to the creative aspects of his job.[1]

TECHNOLOGY NOW AVAILABLE

The technology for MIS is available now for most companies:

—Computers have large memory capacities and great speed.
—Memory is now economical.
—Multiprogramming and multiprocessing systems have been developed.
—On-line storage in tremendous quantities is now available.
—Programming languages specifically designed for quick and variable retrieval are available.

Yet, in a recent poll, the users of MIS from 655 firms (73 percent of those responding) stated that their companies had not used computers to maximum advantage in meeting management's information needs.[2]

WHY IS MIS NOT WORKING?

What is the problem? What then remains for MIS to be widely and effectively utilized by managers as a partner in the management of their business? Except for the continued improvements in hardware and software technology that will occur, the problems of MIS are primarily problems of how to manage the technology existent. Many reasons may be advanced to account for this mismanagement of MIS and computer technology:

- Difficulty in finding and training EDP personnel.
- Insufficient participation of management in supporting computer application development.
- "Communications gap" between systems personnel and management.
- Improper location of the EDP function within the organizational structure.
- Lack of involvement of user organizations in computer decisions.

Each of these is valid and each must be solved if MIS is to efficiently perform as a "conveyor belt" of information. However, too often, a well conceived, well-designed, and well-programmed information system will fail because it is mismanaged after production implementation. There has been little attention devoted to how an information system should function after it has been conceived, designed, programmed, and implemented. How should the system be operated? What should be the user involvement? Development personnel involvement? These questions must be answered with workable solutions dependent upon the system functions, company policy, and company goals present.

CLASSICAL OPERATION OF INFORMATION SYSTEMS

The classical modes of operation of computer systems are: (1) The design organization, which included system analysts and programmers, continued to be responsible for the execution of the system, (2) the user was educated to a degree whereby he could, at least mechanically, exercise the system to produce what he needed, or (3) the Data Center operation personnel were held responsible for the execution of the system, this responsibility either being exercised by computer operators or control clerks.

The first of these modes of operation, execution by the design organization, is inefficient because human resources will not be effectively utilized. An unbalanced work load would exist, whereby some people would spend most of their time on production runs whereas others would spend only a small amount of time. Furthermore, trained systems analysts and programmers should be utilized as systems analysts and programmers and not performing in a repetitive, production environment.

Development organizations would find it difficult to provide the necessary subsystem interface because they would not be aware of activity in those subsystems in which they had not been involved during the development stages. For example, subsystems of a total system may be developed by different programming groups, relatively independent of each other. In summary then, the management and control of production information systems should be recognized as a full-time job, and programmers and systems analysts would not, and should not be expected to, devote this required effort.

The second possible mode of operation emphasized the development organization designing and programming a system and educating the user to submit his own work to the data center. This was a mechanical process for the user because he neither had the time nor interest to learn enough to solve any EDP problems or effect any system improvements. Consequently, the development organization was responsible for solving all problems. Furthermore, the system was infrequently analyzed to determine if it was continuing to meet user needs and, as a result of this, the user frequently became disenchanted with the information system. Another drawback to this type of operation was that systems could not be designed with more than one direct source of input to or disposition of output from the data base. Data collection and document distribution organizations were formed to exercise this interface with the data base for these areas in which volume demanded it.

The most serious drawback, however, was that the user of a subsystem would be naturally concerned only with a subset of the total system, that subset being the subsystem which produces the information he needs. If his reports are validly produced, he does not care that he has entered "garbage" into the data base that some other user might inherit. The Data Center, furthermore, because it has no control over what it does or when it does it, will not be in control of its operation and, hence, the operation of the center might tend to be wasteful and inefficient. This is particularly true in the multi-programming environment where much efficiency can be gained by careful scheduling of various job mixes.

The third mode of operation, the use of computer operators or control clerks to set up and run what was requested by the users, was deficient for some of the same reasons mentioned above—(1) The management and control of a system required an indepth knowledge of that system; it is not a part-time job, (2) interfaces between subsystems must be timely and accurately made to generate meaningful output, and (3) input data control and computer job control must be exercised by the same group for greatest efficiency. This mode of operation, while it provided the data center with some control over its operation, was primarily mechanical with problem solving and system improvement by development personnel.

ADVANTAGES OF PRODUCTION SYSTEMS ANALYST GROUP

The incorporation of a strong support group solely dedicated to managing production information systems will solve many of these traditional operating problems. This group of Production System Analysts provides the following advantages:

1. Development personnel can devote full time to development effort.
2. User devotes full time to improving accuracy and timeliness of data content.
3. Operations—computer operators and control clerks—devote full time to improving the efficiency of the computer.
4. Central point of coordination for data and file maintenance is established.
5. System is constantly analyzed for improvement, ensuring that it meets user's everchanging needs. Development and maintenance effort is requested as required.
6. Quick and efficient processing of data into the base and the resultant generation of documents, interconnecting dependent information subsystems as necessary, is effected.

PROBLEM OF STAFFING

Proper staffing of this group is perhaps the greatest problem encountered. The types and levels of personnel required for this group, in that the function had no precedent, has been largely experimental, but is now stabilizing itself. One key element is becoming increasingly dominant—the group should be more user-oriented than EDP-oriented. The members of the group, furthermore, must be highly motivated for they are the only bridge between company needs and the computer. It is recognized that many of the tasks performed by the group can be done by non-professional personnel if they are provided adequate documentation and guidance. However, the need for some professionals as group leaders is clearly noted, with each being responsible for several systems or subsystems. He then has the responsibility of directing the activities of the personnel assigned to him. The direction of future staffing and organization will depend upon the specific requirements to efficiently operate and manage the individual information systems. Each system, depending upon its importance, requirements, and complexity, is treated differently, with some systems requiring more or less support than others. The number of non-professional personnel and the strategic placement of the professional personnel needs to be continuously reanalyzed as new systems are implemented.

FUNCTIONS

The primary function of the Production Systems Analyst group is to manage the company data base. This broad function is interpreted to include data verification, file updating, information retrieval, file storage, and quality control.

Generally, all functions and activities can be grouped into three categories—preprocessing tasks, postprocessing tasks, and coordinating tasks.

PREPROCESSING TASKS

Preprocessing tasks consist of:

- Input data collection and verification.
- Tabulation coordination.
- Job setup.

The group provides a focal point toward which flows all data to be entered into the data base. The analyst is expected to act as a screen for all input data, sifting out that which is incomplete, not in the proper format, or unusable. Unusable data may, for example, be in the form of an I-O error on a tape, mispunched data cards, or unreadable data sheets. When input data problems are encountered, the analyst is expected to initiate corrective actions which will expediently result in correct data for input into the base. While he is expected to be a data expert insofar as format, he is also expected to be able to recognize invalid content in many instances. The criteria of content expertise varies with the system, but in general, where constraints can be placed on data variability, he should provide a proper screen.

Tabulation coordination includes, when necessary, the preparation and transmission of hardcopy data to the keypunch room, requesting of data card listings, input tape dumps and other associated activities of the computer room, and verifying the results of these activities. If a problem is encountered by the keypunch or computer room in a job, it is resolved, not by these operator groups contacting the user, but by the analyst examining the problem and determining a solution, which may include user involvement.

Job setup requires a considerable portion of a typical Production Systems Analyst's time. It includes the determination of correct files and programs to utilize, preparation of computer operator instructions, and modification of control cards (both program control and job control). Often an analyst must obtain file information such as information content, file numbers, and file names from other personnel. This coordination of system dependencies is vital to successful data base management.

The analyst is required to know capabilities of and be able to correctly execute all programs in those systems for which he is responsible. Computer operators turn to him for the resolution of all problems with the requested work, rather than the user or the development organization. He is expected to analyze job failures and actively seek the correct solution. He may request the user to submit correct data; he may correct a control card error; or he may determine that there may be a program error. In this last case, the development or maintenance organization is notified of the problem and is provided the hardcopy documentation required for their analysis. Note that the Production Analyst does not make any program modification himself, this task being reserved for development and/or maintenance organizations. After a program correction has been made and tested by development, the Production Analyst will instate the new issue of the program into the production cycle as soon as possible. The Production Management group, however, writes utility programs to perform functions such as data listings, tape copies, and data reformatting. Also, they are expected to be experts in the preparation of Job Control Language (JCL) statements and the modification of these control cards as necessary to exercise a series of programs. The analyst is thus given considerable latitude to modify runs as the user's needs change, as the computer operating system changes, or to enable the system to be operated more efficiently. Development support should be required only when a source program needs to be modified.

POSTPROCESSING TASKS

Postprocessing tasks consist of:

- Job checkout.
- Tabulation coordination.
- Document review and distribution.
- Updating of production records.

The task of job checkout includes the analysis of both job control (JCL) messages and user written, program generated diagnostic messages. JCL messages are analyzed for each job to ensure that proper and orderly execution of each program has transpired. As discussed previously, if any failures are encountered, the Production Analyst is expected to actively seek resolution, involving the user, computer operations, or the development group as necessary. The analyst requires an excellent ability to read core dumps. He is also responsible for notifying the user of any program diagnostics so that he may take corrective action. This type of diagnostic is typical of data validation and update programs, and the information usually must be quickly conveyed to the user.

Tabulation coordination includes the transmission of written instructions to computer tab operations in the data center regarding decollating, bursting, card interpreting and other similar activities.

Document review includes verification of (1) correct printing, (2) correct heading and format, (3) correct number of copies, (4) correct paper size and type, and (5) a reasonable content. Note that the user must certify that the content of all documents is correct, but nevertheless, the Production Systems Analyst is expected to measure the content of a document within the framework of the input data used to generate it prior to distributing to the user. This document review, furthermore, is not intended to be a substitute for the normal quality control procedures of the computer operations group.

Record keeping usually includes (1) a log of file names and numbers, (2) contractual documents generated, (3) date and time of runs, (4) run flowcharts, and (5) a log of problems encountered and resolutions enacted. These records become important in determining future schedules and usage forecasts but, more importantly, are critical if reruns are necessary. The contents of a data base may number several hundred reels of tape. Without correct file information, incorrect runs may be made or part of the data base may be accidently scratched. The maintenance of meaningful production run information, in summary, is possibly the most important function of this control group.

COORDINATING TASKS

Coordinating tasks consist of:

- Scheduling keypunch and computer resources.
- Forecasting keypunch and computer usage.
- Production planning.
- Establishing and properly storing system data bases.

As opposed to preprocessing and postprocessing tasks which are easily definable, coordinating tasks are not so easily categorized. These tasks are performed primarily by Information Systems professionals within the group.

These persons are responsible for working jointly with the users and the computer and keypunch sections to establish schedules which will first, meet the needs of the business, and second, optimize the use of the data center's productive resources. Schedules must be constantly analyzed for effectiveness in regard to both objectives above and modifications negotiated when needed. Forecasting of resource usage is also a responsibility of this group, in that the user is almost always too far removed from the data processing function itself to accurately forecast computer and keypunch usage. However, the analyst must obtain from the user some indication of what his projected level of activity (relative to past activity) is to be before he can project computer and keypunch resources required. This method of forecasting has been demonstrated to be extremely accurate.

Production planning activities include a continuing analysis to (1) improve the operation of the system, and (2) better meet user requirements. This analysis may include recommendations to the development organization for new programs or to the maintenance organization for modifications to existing programs. It may include the development of a new data transmittal form and subsequent instruction may be necessary to the keypunch section for efficient encoding of the data.

The data base organization, including filing and labeling procedure, must be developed. Record keeping procedures must be devised uniquely for each system, emphasizing that information which is necessary and practical.

The needs and requirements of the user must be frequently analyzed to ensure the continued relevancy of the entire system. An outdated system or subsystem is an expensive process. After a system has been in production for years, the development organization will not and should not be responsible for this analysis.

CONCLUSION

As stated in the introduction, a great challenge facing the modern business firm is development of means to harness computer technology to provide the right information to the right people at the right time. One tool which has received considerable attention in the past few years and which is now practical with the stabilization of third generation technology is an Integrated Management Information System. Nevertheless, as evidenced by a recent poll in which seventy-three percent of those responding indicated that their companies were not using computers to maximum advantage, this tool, Management Information Systems, is not working in many instances. Valid reasons for this lack of success are available as previously discussed. However, it is this writer's contention that many systems do not develop to their potential, not because of ill-conceived design or poor programming, but rather, because of mismanagement of the system after implementation. While it may not eliminate all the problems in a system, the recognition that Management Information Systems must be managed and controlled will certainly provide many solutions.

BIBLIOGRAPHY

1 ANON
Management and the information revolution
Administrative Management pp 26-28 January 1970

2 J H BARNETT
Information systems: Overcoming the barriers to successful utilization
Management Review pp 9-14 September 1969

3 J H NORTON
Information systems: Some basic considerations
Management Review pp 2-8 September 1969

A methodology for the design and optimization of information processing systems*

by J. F. NUNAMAKER, JR.

Purdue University
Lafayette, Indiana

INTRODUCTION

The design of an Information Processing System (IPS) can be divided into two major problems:

1. What are the requirements of an information system, e.g., what outputs should be produced?
2. What is the best way to produce the required outputs, on time, given the requirements developed in 1?

This paper is concerned with the second problem. However, the methodology developed to solve the second problem provides cost information that can be used by the problem definer to assist him with the determination of the requirements. One approach to determining the requirements of an information system involves comparing the information value of a report with the cost of producing the report.**

Considerable time and money are expended in system design and programming when a firm acquires, leases or in any way uses a computer. With each new system the task of getting the system operational seems to take longer and becomes more costly than the time before. A methodology for analyzing and designing Information Processing Systems is needed if we are to keep from getting further behind.

The general purpose of this paper is to continue the formalization of the process for designing Information Processing Systems and to improve it by increased application of operations research techniques and by more use of the computer itself.

SODA (**S**ystems **O**ptimization and **D**esign **A**lgorithm) is presented as a methodology for automating the system design functions. The objective of SODA is to generate a complete systems design starting from a statement of the processing requirements.

INFORMATION PROCESSING SYSTEMS

An Information Processing System is here defined as a set of personnel, hardware, software packages, computer programs and procedures that have been assembled and structured so that the whole set accomplishes some given data processing requirements in accordance with some given performance criterion.

An important aspect of this definition is that it includes an explicit statement of the "performance criterion" by which performance of the system is measured. A consequence of including performance measures is that the emphasis is focused on the overall performance of the system rather than on any one part. The study of large scale IPS is in essence a study of the performance of the total system: hardware, software and other procedures.

One characteristic of an IPS is that data files are stored on auxiliary memories and it takes a number of interrelated computer programs to meet the specific requirements of the problem definer. The large number of interrelated programs distinguishes the problem that is described here from that aspect of Computer Science which is concerned with individual programs. These systems almost always depend on a large amount of data, now frequently called a data base. A duality exists between the programs and the data, and the structure of each is quite important.[1,2]

The selection of expensive hardware for a given set of requirements is frequently involved and often the expenses increase significantly since the requirements of the system are continuously changing.

* The work described in this paper was sponsored in part by the ISDOS (Information Systems Design and Optimization System) Project, a University of Michigan Research Program directed by Professor Daniel Teichroew and by a Krannert Research Grant at Purdue University.

** The value of a report is often arrived at by a process of "guesstimation" and it is difficult to accurately measure the value.

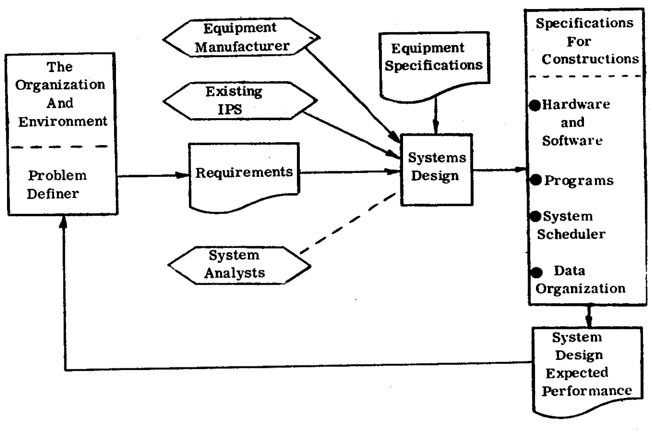


Figure 1—The systems design process

What is needed is a flexible systems design process that can accommodate changing requirements.

THE IPS DESIGN PROCESS

The IPS design process has a number of similarities to any physical design process, such as a production plant or a bridge. In each case there must be an initial recognition of a need. Next, preliminary studies are conducted in which major alternatives are considered, the technical feasibility determined and costs of alternatives estimated. If a decision to proceed is made, the requirements must be stated in sufficient detail for designing the system. The design phase consists of preparing a set of specifications (blueprints) which are detailed enough for the construction phase.

The major functional activities and decision points in the design process of IPS are shown in Figure 1. The design process is initiated through the statement of requirements from "Problem Definers." After the requirements have been documented, the systems analysts consider the equipment available and any constraints (such as the existing system) on the design activity. The design phase consists of producing the specifications for the four major parts of the system:

—Hardware and software packages that will be used
—Programs to be written
—System Scheduler, schedule for sequencing the running of the programs
—Data Organization, specifications of file structure and how the files will be stored in hardware memories

Beneath the surface similarities between the design process of physical structures and IPS there are some differences in emphasis. Typically, more attention is given to the planning and generating alternative designs in developing a production facility than in development of an IPS. This is probably true because there are more external constraints associated with the design of a new facility, e.g., architects, contractors, equipment suppliers, and governmental zoning commissions. In the design and implementation of the Information Processing System, the requirements for the formalization of the design functions are not so apparent, hence there is a tendency to do some of the design work concurrently with construction of the system. This practice often leads to problems.

External constraints, similar to those involved in the physical design process, can be created to formalize the requirements of the Information Processing Department (IPD) through the use of information budgets.[3] The information budget will directly involve management in the operation of the IPD and force more attention to be given to the IPS design.

CURRENT PRACTICE IN SYSTEMS DESIGN

While some formal techniques have been proposed and the computer is sometimes used for calculating estimated processing time, most of the systems design is done in an *ad hoc* basis. The need for formal analysis techniques, of course, has long been recognized.[1,2,4,5]

Information Processing Department managers generally recognize a distinction between systems analysts and programmers. The systems analyst is usually responsible for systems design. In most cases he has had no formal education or training for systems design and has obtained his knowledge by experience. He uses little in the way of tools other than graphical communication devices such as flow charts and decision tables.

Complicating the problems of inexperience and lack of training of the systems analyst is the fact that IPS problems must be subdivided to reduce the task for a systems analyst to a reasonable size and this introduces the problem of coordination of the many systems analysts on the project. In addition it involves the coordination of problem definers, systems analysts and programmers.

The systems design is carried out by one or more analysts who obtain the statement of requirements from those who specify what processing is to be done and what output is needed. The analysts specify programs and file design in sufficient detail so that programmers can write the programs and the files can be constructed.

There is considerable difference of opinion on how much detail of the systems design should be documented and how much other communication between systems analysts and programmers should be allowed. Despite all these problems, information processing systems are being designed.

There are, however, some major undesirable features of the present procedures for systems design. These are:

1. The performance criteria and the requirements of the IPS are not stated explicitly.
2. Programs become the only up-to-date documentation.
3. Accommodation of changes to the IPS is expensive.
4. The design process takes too long.
5. Construction of the system frequently starts before the system is completely designed in an effort to save time.
6. Few alternatives are examined in the design phase.
7. The systems do not work correctly.
8. They are costly to design.
9. Procedures become inefficient as changes in the IPS occur.

Optimization has not been completely overlooked in the design and implementation of Information Processing Systems, but any such effort has been applied to the evaluation and selection of equipment usually for a specified application. This should not be surprising since the commitment for computing system hardware represents a sizable outlay. Since a contract is to be considered, it represents a decision point in which management becomes directly involved. Because of this, care is given to the consideration of alternatives to assure the "best decision" is made—perhaps too much care, in relationship to the return that can be expected from review focused only on this decision point.

The difficulties both of time lag and problem expression summarized above are familiar to anyone who has been involved in management of a large scale Information Processing System department. In addition to these existing difficulties, the design problem will become even more difficult in the future. The hardware is already able to accommodate more powerful software systems than are available. Moreover, attempts to develop sophisticated software systems have been very expensive.

The first three points listed above are related to the deficiencies in problem definition and documentation. Points four and five are concerned with the time required to carry out the systems design and the sixth point relates to the lack of optimal use of resources. Points seven, eight and nine are concerned with the realization that the systems often do not work properly. They are very costly and become inefficient over time.

THE IPS DESIGN PROBLEM

It is becoming more and more apparent that the drawbacks listed in the previous section are likely to become more of a problem in the future. The design of IPS to handle more complex requirements cannot be achieved without unrealistic expenditures of effort. The number of analysts, designers and programmers required to handle these more complex requirements are not likely to be available.

There are basically four ways to improve the situation:

1. Education to increase the number of personnel and improve their quality.
2. Improvement of manual system design tools, techniques, and procedures.
3. Use of generalized rather than tailor-made software.
4. Automation of the system design process.

Most attention has been devoted to the first three approaches. The SODA methodology concentrates on the fourth approach and rests on the premise that specifications for the IPS can be generated directly from a statement of user requirements for a limited class of processing requirements. The specifications must be detailed enough to verify feasibility and to evaluate the performance of the proposed system but not more detailed than enough to specify construction because producing "too detailed specifications" are costly and they may have to be changed in any case. Also, "too detailed specifications" are embedded in processing procedures which tend to bind the design unnecessarily, at too early a stage and with negative payoff.

Since the purpose of the IPS is to produce outputs, it must respond to changing inputs. One of the design problems is to decide what changes should be accommodated with what degree of ease. Certain components are easier to change than others, e.g., it is easier to change a program than to change the operation performed in the hardware. It is easier to change the data in a file than it is to change the structure of the file.

The systems design decision hierarchy consists of certain decisions which constrain later activities. For example:

1. Selecting a hardware configuration constrains

everything that follows to what can be accomplished with the selected hardware.
2. Selecting the size of the main memory, number and type of input and output units and auxiliary memory constrains still further what can be accomplished.
3. Selecting an operating system determines what processing organization is possible.
4. Selecting compilers and utility programs constrains what the programmer can do for himself. The programmer, when he writes a particular program module, is constrained by the data organization, the input and output formats and the software and hardware on which the program is run.

The purpose of our discussion of design is to identify the design decisions that are made at any point, to enumerate the decisions that are possible, and to develop methods for:

1. Determining when (in the design process) the decisions should be made.
2. What is the optimal or suboptimal decision in a particular case.

Currently each organization tends to develop its own procedures with very little evaluation of methods developed elsewhere. IPS design can benefit by the synthesis of the available knowledge and practical know-how, and by the development of more powerful analytical methods to replace the current *ad hoc* methods.

In the course of the evaluation of computers and their use as information processing devices a variety of tools have been developed to aid the analyst in making the necessary design decisions and in facilitating the construction of these systems. Steiger[6] describes some of the steps in systems design and it is clear that as these aids have become more sophisticated the computer is being used more and more extensively in the design process. For example, there are commercial computer systems simulation packages[7,8,9] available for use in the evaluation of computer system performance.

However, most of the use of the computer has been in the construction phase, i.e., in the generation of computer code from source language statements. These source language statements could only be prepared once the system had been designed, i.e., once the analyst had decided what hardware would be used, how data would be organized and stored in the hardware, how the processing would be combined into programs, and how these programs would be organized, i.e., in what order the programs would be run to accomplish the total processing requirements. It is exactly these decisions which have a major influence on the performance of the system, i.e., how much computer hardware and how much time are needed to satisfy a given set of processing requirements.

SODA

It is with respect to systems design decisions that much work needs to be done and SODA is presented as a methodology for the design and optimization of IPS. The existing systems simulators[7,8,9] assume as given a systems design, i.e., a description of each program, schedule for a set of runs and structure of the data files. SODA is intended to specify a systems design from a statement of the requirements and to generate the set of programs and data files.

SODA consists of a number of sub-models that are solved using mathematical programming, graph theory and heuristic procedures. Since the overall design problem is very large, it is convenient to view the algorithm as a multilevel decision model with the decision variable of one level becoming a constraint at the next level and so on. The partitioning of the problem into a multilevel structure implies that a different set of decision variables are required for each level of the algorithm.

The decision making structure of SODA is described by (1) specification of the inputs and outputs, (2) specification of decision variables and determination of feasible alternatives, (3) selection of an objective function, (4) expression of objectives as a function of decision variables, (5) explicit statement of constraints which limit the value of the decision variable and (6) solution, i.e., determination of the values of the decision variables.

SODA is a set of computer programs which begins with the initial statement of requirements (i.e., what the system is to do) and proceeds through the design and specification of the system. SODA is not concerned with the determination of which requirements are to be stated. The assumption is made that the problem definer (PD) can accurately identify his requirements. The major components of SODA are:

Problem Statement Language (PSL)

SODA/PSL is a technique for stating the requirements of the IPS independent of processing procedures. It also provides the capability for easily handling changes in requirements.

Problem Statement Analyzer (PSA)

SODA/PSA is a program for analyzing the statement of the problem and organizing the information required in SODA/ALT and SODA/OPT. This program also provides feedback information to the problem definer to assist him in achieving a better problem statement.

Generation of Alternative Designs (ALT)

SODA/ALT is a procedure for the selection of a CPU and core size and the specification of alternative designs of program structure and file structure.

Optimization and Performance Evaluation (OPT)

SODA/OPT is a procedure for the selection of auxiliary memory devices and the optimization and performance evaluation of alternative designs.

Refer to Figure 2 for an overview of SODA.

The output of SODA is (1) a list specifying which of the available computing resources will be used, (2) specifications of the programs generated, (3) specifications of the file structure and the devices on which they will be stored, and (4) a schedule of the sequence in which the programs must be run to accomplish all the requirements.

SODA selects a set of hardware, generates a set of programs and files that satisfy timing requirements, core memory and storage constraints such that the hardware cost of the system is minimized.

SODA is limited to the design of uni-programmed batch systems, sequential auxiliary storage organization, the specification of linear data structures, and the selection of a single CPU. The model is deterministic. The problem statement technique is intended to handle "report oriented" data processing systems. Refer to Figure 3 for the interaction of the levels of SODA.

The overall structure of SODA/ALT and SODA/OPT is given in Figure 4 which describes the decision variables, objective function, alternatives, constraints and solution techniques for each level in a summary form.

PROBLEM STATEMENT LANGUAGE

It is assumed that someone called a Problem Definer (PD) is familiar with the operation of the organization and has the necessary training to describe the processing requirements of the organization. The Problem Definer states his data processing requirements in a problem statement (PS) according to SODA/PSL and the requirements are input to SODA/PSA in the form of a subset of a PS called a Problem Statement Unit (PSU). A PSU consists of three major categories: the data description, processing requirements and operational requirements. The data description is defined by Elementary Data Sets (eds) and Data Sets (ds). The processing requirements consist essentially of a set of formulas called Processes (pr). The operational requirements consist of information on volumes, frequency of output and timing of input and output.

An eds consists of a Data Name (dn), Data Value (dv), Descriptor Name (sn) and Descriptor Value (sv).

The sales of model X in the north region is an example of an eds.

Data Name		Descriptor Value	Descriptor Name		Data Value
SALES_MODEL_X	(in the)	NORTH	REGION	(is)	500

A Data Set is the set of all eds with the same dn. The sales of model X in all regions of the country is an example of a Data Set.

Data Name		Descriptor Value	Descriptor Name		Data Value
SALES_MODEL_X	(in the)	NORTH	REGION	(is)	500
SALES_MODEL_X	(in the)	SOUTH	REGION	(is)	600
SALES_MODEL_X	(in the)	EAST	REGION	(is)	300
SALES_MODEL_X	(in the)	WEST	REGION	(is)	400

There are four types of Data Sets and SODA makes use of the Data Set type in the file structure algorithm. An Input Data Set is any input data to the IPS. A Storage Data Set is that data which is stored in the IPS. A Terminal Data Set consists of output reports or forms and is not retained in permanent storage in the IPS. A Computed Data Set is the output of a Process that is neither a Storage nor a Terminal Data Set.

A Data Set may have up to three Descriptor Pairs to uniquely identify a Data Value. The set of Descriptor Pairs is referred to as an Identifier and a Descriptor

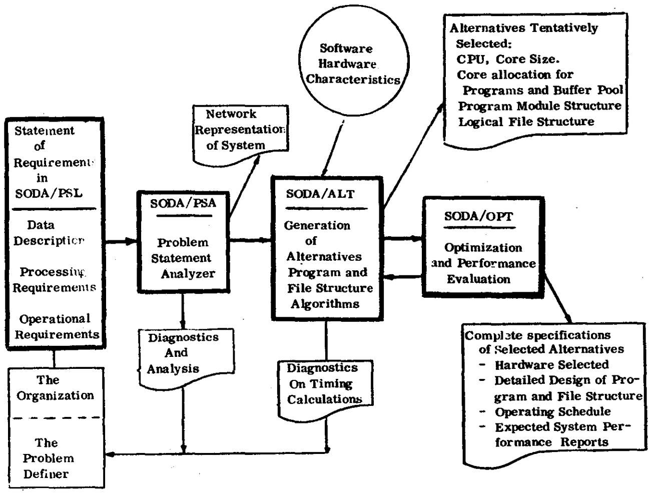

Statement of Requirement in SODA/PSL
Data Description
Processing Requirements
Operational Requirements
The Organization
The Problem Definer
SODA/PSA
Problem Statement Analyzer
Network Representation of System
Diagnostics And Analysis
Software Hardware Characteristics
SODA/ALT
Generation of Alternatives Program and File Structure Algorithms
Diagnostics On Timing Calculations
Alternatives Tentatively Selected:
CPU, Core Size.
Core allocation for Programs and Buffer Pool
Program Module Structure
Logical File Structure
SODA/OPT
Optimization and Performance Evaluation
Complete specifications of Selected Alternatives
- Hardware Selected
- Detailed Design of Program and File Structure
- Operating Schedule
- Expected System Performance Reports

Figure 2—SODA: Systems Optimization and Design Algorithm

Pair consists of a Descriptor Name and Descriptor Value.

Let id_k be the kth Identifier where $dn[id]$ is a shorthand notation for writing $\langle dn, sn\rangle$, $\langle sv, dv\rangle$.

Example: The GROSS_PAY of 5 employees each

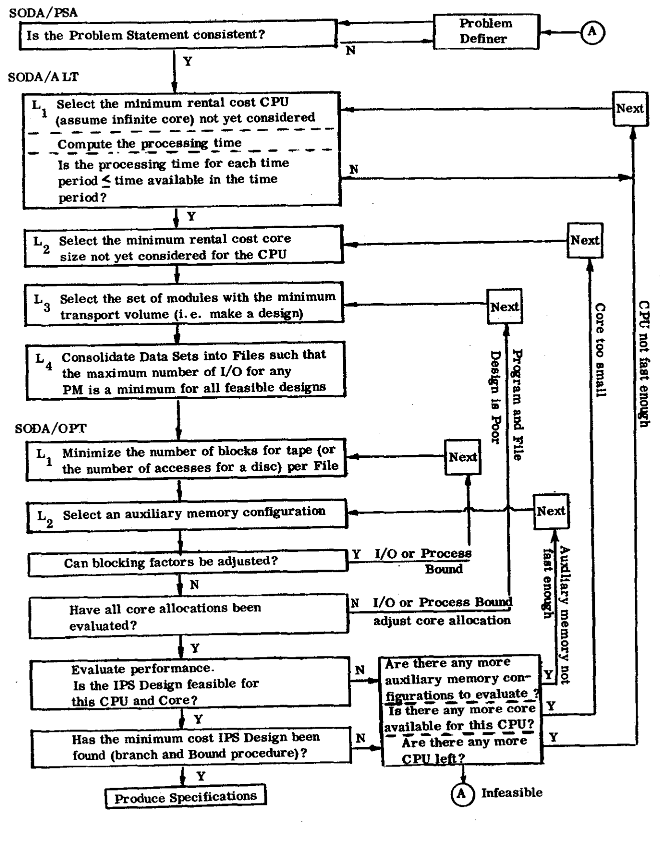

SODA/PSA
Is the Problem Statement consistent?
Problem Definer
N
Y
A
SODA/ALT
L1 Select the minimum rental cost CPU (assume infinite core) not yet considered
Compute the processing time
Is the processing time for each time period ≤ time available in the time period?
N
Next
Y
L2 Select the minimum rental cost core size not yet considered for the CPU
Next
L3 Select the set of modules with the minimum transport volume (i.e. make a design)
Next
L4 Consolidate Data Sets into Files such that the maximum number of I/O for any PM is a minimum for all feasible designs
Program and File Design is Poor
Core too small
CPU not fast enough
SODA/OPT
L1 Minimize the number of blocks for tape (or the number of accesses for a disc) per File
Next
L2 Select an auxiliary memory configuration
Next
Can blocking factors be adjusted?
Y I/O or Process Bound
N
Have all core allocations been evaluated?
N I/O or Process Bound adjust core allocation
Auxiliary memory not fast enough
Y
Evaluate performance. Is the IPS Design feasible for this CPU and Core?
N
Are there any more auxiliary memory configurations to evaluate?
Y
Is there any more core available for this CPU?
Y
Are there any more CPU left?
Y
Has the minimum cost IPS Design been found (branch and Bound procedure)?
N
Y
Produce Specifications
A Infeasible

Figure 3—Interaction of the levels of SODA

identified by the Identifier consisting of a single Descriptor EMPLOYEE_NUMBER is written as

$\langle$GROSS_PAY $[id_1]\rangle$ and is equivalent to:
$\langle$GROSS_PAY, $sn_1\rangle$,

$$\langle sv_1, sv_2 \ldots sv_5, dv_1, dv_2 \ldots dv_5\rangle$$

where:

$$\begin{array}{ll} sn_1 = \text{EMPLOYEE\_NUMBER} & \\ sv_1 = 17 & dv_1 = \$263 \\ \vdots & \vdots \\ sv_5 = 21 & dv_5 = \$300 \end{array}$$

A Process is the smallest unit of processing requirement that may be grouped, but never subdivided by SODA. A Process produces a Data Set and is a well defined assignment type statement. There are four types of Processes defined in SODA/PSL. They are: (1) COMPUTE, (2) SUM, (3) IF and (4) GROUP.

The Compute Process is a variation of the familiar assignment statement found in FORTRAN, ALGOL or COBOL.

An example of the COMPUTE Process is:

NET_PAY $[id_1]$ = COMPUTE GROSS_PAY $[id_1]$ − DEDUCTIONS $[id_1]$

This expression consists of three Data Sets, NET_PAY $[id_1]$, GROSS_PAY $[id_1]$, DEDUCTIONS $[id_1]$ and all are identified by id_1 (EMPLOYEE_NUMBER). It is implied that whenever the statement is to be executed it is executed for all valid Descriptor pairs. The number of Descriptor pairs and the number of

Level	Decision Variables	Objective	Alternatives	Constraints	Optimization Techniques
			SODA/ALT		
1	CPU	Select the minimum rental cost CPU class	list of available CPU classes	Time available for processing	search for ordered CPU classes
2	Core Size	Select the minimum rental cost core size	list of available core sizes	-Times available for processing -CPU class	search of ordered core sizes
3	Program Module	Select the set of modules with the minimum transport volume	feasible grouping arranged in tree structure	-Core Size available for modules	Network analysis and branch and bound search over feasible alternatives
4	Data Structures	Select Files such that the maximum number of I/O for any module is a minimum	feasible grouping arranged in tree structure	-Program Modules	Network analysis and branch and bound search over feasible alternatives
			SODA/OPT		
1	Storage Structure	Minimize the number of inter block gaps for tape or the number of accesses for a disk	1 char/block to upper limit	-Program Modules -Data structure	Non-linear programming model
2.	Number and type of auxiliary memories	Minimize the variable reading and writing time	list of available devices	-CPU and Core Size -Program Modules -Storage structures -Data structure	Integer Programming Model

Figure 4—Decision levels of SODA/ALT and SODA/OPT

each type of arithmetic operation (e.g., multiplication, addition) is used by SODA/ALT for the purpose of estimating running times. The number of each type of arithmetic operation is obtained from the list of Processes.

The SUM Process is an expression that makes it convenient to sum over two or three Descriptor Names and the associated Descriptor Values. For example, one could sum labor cost by employee numbers and department numbers. The IF Process is a conditional expression. The GROUP Process groups Data Sets that are required to represent output reports. The four types of Processes enables SODA/PSA to construct precedence graphs of the Processes and Data Sets that are necessary for the Program and File Structure Algorithms.

The time and volume characteristics of the IPS are also described in the problem statement. Time requirements are specified by stating absolute time deadlines, i.e., paychecks must be produced by 4:00 PM on Friday. The statement of the time requirements for the output reports of the IPS is expressed by a Need Vector. The Data Set volumes are computed from the volumes stated for each eds. All time and volume information is expressed in units specified by the problem definer.

The structure of a Problem Statement in SODA/PSL consists of:

- Problem Statement Name
  - List of Identifiers
  - List of Descriptors
    - Descriptor Name
    - The Number of Descriptor Values for each Descriptor Name
  - List of Data Sets
    - Data Name
    - Volume of Data Set
    - Type of Data Set
  - List of PSU
    - Contents of each PSU
      - PSU Number
      - PSU Name
      - Need Vector
      - List of Processes
      - END of PSU

END OF PROBLEM STATEMENT

SODA/PSL is a non-procedural Problem Statement Language in the sense that the PD writes a PSU without imposing any procedural ordering on the Processes. The precedence relationships of the Processes are inferred by SODA/PSA.

The PS must contain sufficient detail so that systems analysts and programmers could use it (if necessary) to design and implement the Information Processing System with no additional information.

PROBLEM STATEMENT ANALYZER

The problem statement analyzer (SODA/PSA) accepts the requirements stated in SODA/PSL, analyzes them and provides the problem definer diagnostics for debugging his problem statements and reports. SODA/PSA also produces a number of networks which record the interrelationships of Processes and data and passes the networks on to SODA/ALT.

Each type of input and output is specified in terms of the data involved, the transformation needed to produce output from input and stored data. Time and volume requirements are also stated. SODA/PSA analyzes the statement of the problem to determine whether the required output can be produced from the available inputs. The PS stored in machine readable form is processed by SODA/PSA which:

1. checks for consistency in the PS and checks syntax in accordance with SODA/PSL; i.e., verifies that the PS satisfies SODA/PSL rules and is consistent, unambiguous, and complete.
2. prepares summary analyses and error comments to aid the problem definer in correcting, modifying and extending his PS, and
3. prepares data to pass the PS on to SODA/ALT.
4. prepares a number of matrices that express the interrelationship of Processes and Data Sets.

There are a number of papers that discuss the use of graphs and their associated matrices for the analysis of program and data structure.[1,2,10,11,12,13]

SODA/PSA follows the papers of Börje Langefors[1,2] and Raymond B. Briggs.[10] Langefors discusses the use of matrix algebra and graph theory to represent the processing units and data units in an IPS. Langefors' work differs from others using graph theory for this purpose in that it includes a performance criteria to be optimized.

Briggs added to the matrix definitions of Langefors and provided the necessary structure to develop a Program and File Structure Algorithm.

The problem statement is defined in SODA/PSA as the set of Processes required, the set of Data Sets needed by each Process and the precedence relationships of the Processes (pr) and Data Sets (ds).

SODA/PSA generates the P, P^* and E matrices for each PSU and for the entire IPS.

P—Precedence Matrix: Data Sets

$p_{ij}=1$ if ds_i is a direct precedent of ds_j,
$p_{ij}=0$ otherwise.

*P**—Precedence Matrix: Processes.

$p^*_{ij}=1$ if pr_i is a direct precedent of pr_j,
$p^*_{ij}=0$ otherwise.

The precedence matrices are checked for consistency using Marimont's procedure.[14]

E—Incidence Matrix: Processes and Data Sets

$e_{ij}=1$ if ds_j is an input to pr_i
$e_{ij}=-1$ if ds_j is an output of pr_i
$e_{ij}=0$ if there is no incidence between ds_j and pr_i

Let v_j be the volume of ds_j, l_i be the number of inputs and outputs for each Process and m_j be the multiplicity of Data Set transport for ds_j. Let m_j represent the number of times ds_j is an input or output of a set of Processes.

$$l_i=\sum_{j=1}^{k} |e_{ij}|;\ i=1, 2, \ldots n.$$

$$m_j=\sum_{i=1}^{n} |e_{ij}|;\ j=1, 2, \ldots k.$$

The transport volume for ds_j is

$$tv_j=m_j \cdot v_j.$$

The transport volume for the set of Data Sets is

$$TV=\sum_{j=1}^{k} m_j \cdot v_j.$$

Transport volume is used as a criterion to evaluate alternative program and data designs and is discussed in the next section.

Let ds_j be represented by a ○ and pr_i be represented by a □. An example of an incidence graph and the associated incidence matrix is given in Figure 5.

The *R*, *R** and *M* matrices are generated for the entire set of Processes*.

R—Reachability Matrix: Processes.

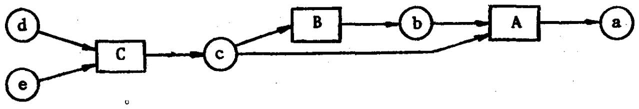
The *R* matrix is used to check precedence violations in the grouping procedure of SODA/ALT.

$$R=(P^*)V(P^*)^2V\ldots V(P^*)^{q-1},$$

where q is the index of the nilpotent matrix P^*.

$r_{ij}=1$ if pr_i has any precedence relationship with pr_j,
$r_{ij}=0$ otherwise.

*R**—Partial Reachability Matrix: Processes.



The associated incidence matrix is:

		Data Sets					
		a	b	c	d	e	ℓ_i
Processes	A	-1	1	1	0	0	3
	B	0	-1	1	0	0	2
	C	0	0	-1	1	1	3
	m_j	1	2	3	1	1	
	v_j	20	20	20	20	20	
	tv_j	20	40	60	20	20	

The transport volume for the Data Sets (TV) in this example is 160 units.

Figure 5—Incidence graph and matrix

The R^* matrix is used to calculate the *M* matrix.

$$R^*=(P^*)^2V(P^*)^3V\ldots V(P^*)^{q-1}.$$

$r^*_{ij}=1$ if pr_i has a higher (2 or more) order precedence with pr_j,

$r^*_{ij}=0$ otherwise.

It was shown by Briggs[10] that by using a theorem proved by Warshall[15] that *R* and *R** can be constructed without first computing successive powers of *P**.

M—Feasible Process Grouping Matrix: Processes.

If $m_{ij}=-1$ there exists higher (2 or more) order relationships between pr_i and pr_j and pr_i cannot be combined with pr_j. If $m_{ij}=0$ there is no precedence ordering and pr_i can be combined with pr_j. This indicates a feasible but not necessarily profitable grouping. If $m_{ij}=1$ there is a direct precedent relationship and pr_i can and should be combined with pr_j since this indicates a feasible and profitable grouping. If $m_{ij}=2$ there is an immediate reduction in logical input/output requirements when pr_i and pr_j are grouped.

$m_{ij}=-1$ if r^*_{ij} or $r^*_{ji}=1$.
$m_{ij}=0$ if $r^*_{ij}=0$ and $r^*_{ji}=0$ and $p^*_{ij}=0$ and $p^*_{ji}=0$; except when $p^*_{il}=1$ and $p^*_{jl}=1$ or $p^*_{li}=1$ and $p^*_{lj}=1$.

$m_{ij}=1$ if $r^*_{ij}=0$ and $r^*_{ji}=0$ and $p^*_{ij}=1$ or $p^*_{ji}=1$.
$m_{ij}=2$ if $r^*_{ij}=0$ and $r^*_{ji}=0$ and $p^*_{il}=1$ and $p^*_{jl}=1$ or $p^*_{li}=1$ and $p^*_{lj}=1$.

* A procedure is discussed briefly in the next section for partitioning the entire set of Processes into smaller groups when the number of Processes is very large.

pr_l has a first order precedence or succedent relationship with pr_i and pr_j.

A list of all feasible pairs for grouping of Processes is constructed from the M Matrix and passed to SODA/ALT.

GENERATION OF ALTERNATIVES

The information system design phase begins after the requirements have been stated, verified and analyzed in SODA/PSA. SODA/ALT accepts as input, the output of SODA/PSA and a statement of the available computing resources, hardware and utility programs. The hardware alternatives are ordered in a tree structure as shown in Figure 6.

A feasible CPU and core size are specified using a heuristic timing procedure. An ordered search of different CPU's is made in an attempt to find the minimum CPU assuming an infinite core memory with no auxiliary memory and all Processes and Data Sets in real core. The premise is that if a CPU cannot perform adequately under these "ideal" conditions it cannot possibly be adequate with limited core constraints.

The processing time for each time period (i.e., a week) is computed. If this is less than the actual time available to do this, then a lower bound of CPU capability is found. If not, the next CPU is tried. If it appears that some shifting of load from one or more time periods (i.e., week 1) to other time periods (i.e., week 3) could solve the problem, then the problem definer is advised about it and given a chance to "level" the requirements.

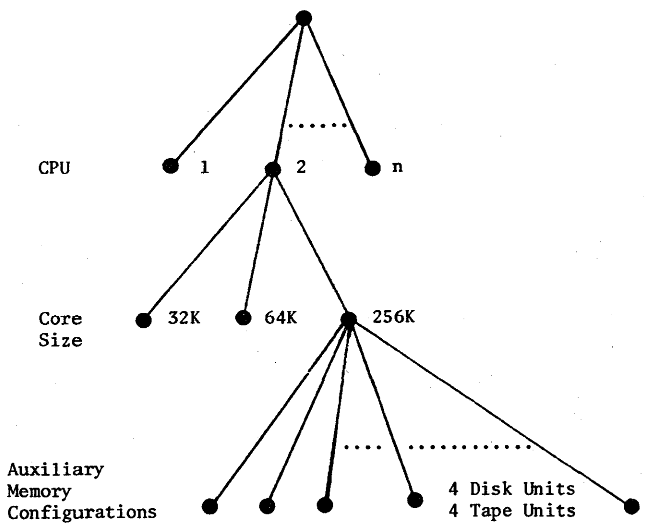


Figure 6—Hardware alternatives

Having found a CPU that will perform adequately under infinite core assumptions, an ordered search (starting with the smallest core size) is made of available core sizes for this CPU.

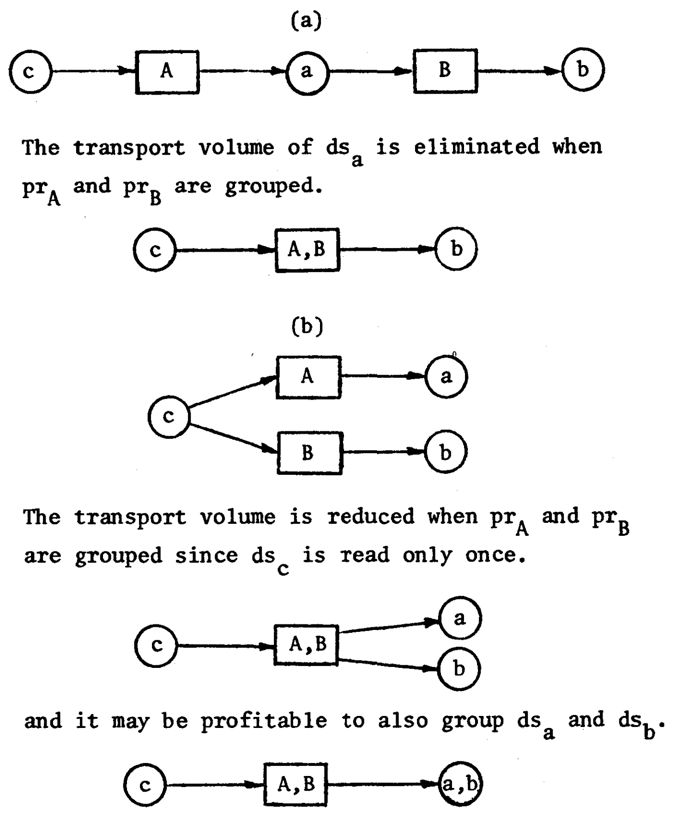
Using as constraints CPU and core size, a graph theoretic model generates alternative Program Module and File Designs. A Program Module (pm) is a set of Processes grouped together by SODA/ALT. A File (f) is one or more Data Sets that are grouped together. From the Matrix of feasible groupings (M) a list of feasible (profitable) groupings of Processes is obtained. The list of feasible groupings is partitioned into 3 cases to reduce the number of alternative program designs that must be evaluated. The cases separate output reports into classifications of due dates for reports. The cases are then divided into subcases. (A subcase is a group of Processes that has no Process precedence link to other subcases.) For each subcase, feasible Program Modules of size 3, 4, . . . N are generated where N is the number of Processes in the partitioned list (or subcase).

It is known that by grouping Processes into a composite process called a Program Module, the multiple input and output of Data Sets can be reduced. Such grouping of processes, however, requires additional main memory for the Program Modules. In generating an efficient design, it is necessary to decrease the transport volume (total number of characters read in and written out of main memory) in order to reduce the processing time. If Data Set volumes remain constant, in order to decrease the transport volume, the multiplicity (the number of times a Data Set is input and output) of Data Set transport must be decreased. After the Program Modules are specified the Data Sets are consolidated into Files for the purpose of reducing the number of input/output Files required and for better utilization of storage in auxiliary memory. Process grouping is shown to correspond to a grouping of rows of the incidence matrix, and data set consolidation is shown to correspond to a grouping of columns.

Program Module and File Design is concerned with the reduction of processing time and can be summarized by the two methods by which the processing time can be reduced. SODA/ALT determines:

1. which operations (Processes) will be grouped into Program Modules
   a. Group Processes which will eliminate the writing out and reading in of a Data Set. For instance Figure 7a.
   b. Group Processes which require the same Data Sets. For instance Figure 7b.

The objective is to reduce total transport volume and thus total processing time.



The transport volume of ds_a is eliminated when pr_A and pr_B are grouped.

The transport volume is reduced when pr_A and pr_B are grouped since ds_c is read only once.

and it may be profitable to also group ds_a and ds_b.

Figure 7—Methods for reducing processing time

2. which Data Sets will be grouped into Files (f)
   a. organize the data structure of the Files so that data which are needed together are close together in order to reduce searching time.
   b. organize the data structure of the Files such that fewer logical input/output devices are needed.

The following matrices are used in SODA/ALT to generate alternative Program Module and File Designs.

S—Program Module Selection Matrix: Program Modules and Processes.

The *S* matrixre presents the alternative grouping of Processes.

$s_{ij}=1$ if pr_j is a member of pm_i,
$s_{ij}=0$ otherwise.

E′—Incidence Matrix: Program Modules and Data Sets.

The *S* matrix is multiplied by the *E* matrix to produce the new incidence matrix *E′*, where $E'=S\Lambda E$.

The Boolean matrix operators "Λ" and "V" for Process grouping follows the rules of Boolean Algebra with the following exceptions for the Boolean addition operator.

$0\,V-1=-1$

$1\,V-1=0$ if the output ds_j is used only by the pm_i with which it is grouped and ds_j is not a Terminal ds or Storage ds.

$1\,V-1=1$ if the output ds_j is required in a Process that is a member of another Program Module or is a Terminal ds or Storage ds.

D—Feasible Data Set Grouping Matrix: Data Sets.

$d_{ij}=1$ if e'_{ij_α} and $e'_{ij_\beta}=1$ or $e'_{ij_\alpha}=-1$ and $e'_{ij_\beta}=-1$ and ds_{j_α} and ds_{j_β} are the same data types and have a common descriptor or if ds_i and ds_j are input Data Sets and have a common descriptor.

$d_{ij}=0$ otherwise.

ds_{j_α} and ds_{j_β} are two data sets required by pm_i. The same data type for ds_{j_α} and ds_{j_β} refers to the classification of Input *ds*, Computed *ds*, Storage *ds* and Terminal *ds*. The test for a common Descriptor Name is intended to link Data Sets together that have a relationship other than common input or common output of a Program Module.

G—File Selection Matrix: Data Sets and Files.

The *G* matrix represents the alternative grouping of Data Sets.

$g_{ij}=1$ if ds_i is a member of f_j,

$g_{ij}=0$ otherwise.

E″—Incidence Matrix: Program Modules and Files.
The *E′* matrix is multiplied (as described earlier) by the *G* matrix to produce the incidence matrix *E″* of Program Modules and Files, where $E''=E'\Lambda G$.

The selection procedure for program design is organized as a tree structure with all feasible alternatives ordered in terms of core memory requirements and transport volume. The procedure for File design is organized by descriptors (keys) and the number of input/outputs Files required for each Program Module.

If software modules such as sort modules are required to process Files they are inserted in the IPS Design.

The next step is to look for design improvements and to select a specific number and type of auxiliary memory units.

OPTIMIZATION AND PERFORMANCE EVALUATION

The optimization and performance evaluation phase generates a storage structure and scheduler, selects auxiliary memory devices, and searches for ways to improve the IPS design. SODA/OPT may return control to SODA/ALT to select another CPU, core size or to select another set of Program Modules and Files.

SODA/OPT selects the minimum cost hardware configuration that is capable of processing the stated requirements in the time available. This phase consists of a number of mathematical programming models and timing routines that are used to (1) optimize the blocking factors for all Files, (2) evaluate alternative designs; i.e., specify the number and type of auxiliary memory devices, (3) assign Files to memory devices and (4) generate an operating schedule for running program Modules. These sub-models follow the work of Schneidewind[16] and Thiess.[17] Refer to McCuskey[18] for another approach to the design of data organization.

In SODA/OPT the performance criterion is optimized within the constraint set by the capability of the hardware and by the processing requirements. SODA/OPT produces a report describing the system and stating its predicted performance. On the basis of this, the Problem Definer may decide to change his PS, or accept the design; SODA/OPT then provides detailed specifications for the construction of the system.

The output of SODA/OPT is:

1. A list specifying which of the available computing resources will be used.
2. A list of the Program Modules specifying the input, output and computations to be performed in each.
3. A list of Files to be maintained specifying their format and manner in which they will be stored. Assignment of Files to memory devices.
4. A statement of the sequence and manner in which the Program Modules must be run to accomplish all the requirements.

IMPLEMENTATION

SODA has been written for the Univac 1108 in FORTRAN at Case Western Reserve University;[19] the program has also been implemented on the IBM 360/67 at the University of Michigan by Professor Daniel Teichroew. The program is currently being modified and rewritten for the CDC 6500 at Purdue University.[20]

The Case version of SODA has the following drawback: If there are two feasible solutions

(1) Small CPU, large core
(2) Larger CPU, smaller core

such that (2) is cheaper, then the algorithm would not find (2). The assumption is made (the Case version) on the ordering of the hardware tree that this situation (2) would not occur. This was done in order to simplify the search procedure and to reduce the size of the large combinatorial problem involved. A branch and bound procedure is being implemented in the Purdue version so that alternative (2) would be found. The Case version also does not automatically evaluate all Program Module and File Designs. A partial set of Program Modules and File designs ($\sim$50,000) is generated. A smaller number of designs ($\sim$200 for the Company Y example) must be selected manually and then input to SODA to be evaluated. Procedures have been developed[20] to eliminate the need for man-machine interaction in SODA/ALT and the procedures are being implemented in the Purdue version.

The program has been run using an example problem called Company "Y." The example consists of 117 Processes and 180 Data Sets. Approximately 30 runs were required to debug the problem statement for Company Y.

A single run for SODA/PSA takes about 120 seconds of execution on the UNIVAC 1108. The total time required for the Company Y example is difficult to estimate since the SODA program was not run from beginning to end at one time; many of the submodels were run, then a data file passed to the next submodel and so on.

A series of hypothetical computers is described in the hardware file. The hardware file consists of 3 CPU's with 5 core options for each CPU. The auxiliary memory option consists of two types of tape drives and two types of disk units.

ACKNOWLEDGMENTS

The author is indebted to Mr. Hasan Sayani, Mr. Carl Singer and Dr. Daniel Teichroew of the University of Michigan for their contribution to the development of SODA.

REFERENCES

1 B LANGEFORS
Some approaches to the theory of information systems
BIT 3 4 1963

2 B LANGEFORS
Information system design computations using generalized matrix algebra
BIT 5 2 1965

3 J F NUNAMAKER JR A B WHINSTON
Computing center as a profit center
Computer Sciences Department TR52 Purdue University Lafayette Indiana January 1971

4 L LOMBARDI
Theory of files
Proceedings of the Eastern JCC 1960

5 D TEICHROEW
ISDOS—A research project to develop methodology for the automatic design and construction of information processing systems
ISDOS Working Paper Number 1 Case Institute of Technology August 1967

6 W STEIGER
Survey of basic processing function: Literature and related topics and (2) Derivation of primitives for data processing
ISDOS Working Paper Number 11 Case Institute of Technology May 1968

7 D J HERMAN F H IHRER
The use of a computer to evaluate computers
AFIPS Conference Proceedings Spring Joint Computer Conference Volume 25 pp 383–395 1964

8 L R HUESMANN R P GOLDBERG
Evaluating computer systems through simulation
The Computer Journal Volume 10 Number 2 August 1967

9 *Digest of the second conference on applications of simulation*
Sponsored by SHARE/ACM/IEEE/SCI New York December 1968

10 R B BRIGGS
A mathematical model for the design of information management systems
MS Thesis Division of Natural Science University of Pittsburgh 1966

11 T C LOWE
Analysis of boolean program models for time shared paged environments
Communications of the ACM Volume 12 Number 4 April 1969

12 T C LOWE
Automatic segmentation of cyclic program structures based on connectivity and processor timing
Communications of the ACM Volume 12 Number 1 January 1970

13 C V RAMAMOORTHY
Analysis of graphs by connectivity considerations
Journal of the Association for Computing Machinery Volume 13 Number 2 April 1966

14 R B MARIMONT
A new method of checking the consistency of precedence matrices
Journal of the ACM Volume 6 Number 2 April 1959

15 S WARSHALL
A theorem on boolean matrices
Journal of the ACM Volume 9 Number 1 January 1962

16 N F SCHNEIDEWIND
Analytical model for the design and selection of electronic digital computing systems
D B A Dissertation University of Southern California 1965

17 H E THIESS
Mathematical programming techniques for optimal computer use
Proceedings ACM 20th National Conference 1965

18 W A McCUSKEY
On automatic design of data organization
AFIPS Conference Proceedings Fall Joint Computer Conference Volume 37 1970

19 J F NUNAMAKER JR
On the design and optimization of information processing systems
PhD Dissertation Case Western Reserve University 1969

20 J F NUNAMAKER JR
SODA: Systems Optimization and Design Algorithm
Computer Sciences Department TR51 Purdue University Lafayette Indiana January 1971

Computer generated repeatable tests

by FRANKLIN PROSSER

Indiana University
Bloomington, Indiana

and

DONALD D. JENSEN

University of Nebraska
Lincoln, Nebraska

INTRODUCTION

While we wait for Computer-Assisted Instruction to revolutionize teaching practices, a number of more tractable computer techniques are proving useful in dealing with large university classes. One of these, the use of Computer Generated Repeatable Tests (CGRT), is the topic of this paper. The concept of repeatable testing has intrigued teachers for years. The idea is to have the flexibility in class examination procedures to test a student repeatedly over a section of material, and at the student's own pace. The implementation of such a plan involves producing, administering, and grading large numbers of different tests over the same material. In classes of moderate or large size, the mechanics of such procedures has defeated even the most dedicated instructors. The present computer augmented procedure—a result of collaboration between a psychology professor (DDJ) and a computer professional—overcomes the difficulties by fully automating the preparation and grading of individualized tests. Before discussing the technical details of the CGRT process, we will briefly give the rationale behind repeatable testing.

THE PROBLEM

Large classes are becoming an increasingly important part of American higher education. This trend is disturbing because available evidence indicates that the conventional large lecture class is an unsatisfactory educational system, one that is disliked by students and considered educationally ineffective by professors. The difficulties with large classes appear to stem from the inflexibility of many of the educational activities in the classes, and from the fact that the student's performance is monitored infrequently and often inadequately. The student has only a passive role in a large lecture section; the lecturer lectures and the student (at best) listens. Opportunities for the student to creatively display knowledge of the subject matter are limited, since the time required to grade essay exams or individual projects dictates that these be given very infrequently, if at all. Often the only measures of the student's knowledge of course material are the examinations he takes infrequently during the course.

Of the several inadequacies of large class instruction, the examination procedures are perhaps most serious'y deficient. The typical examination administered to a large class consists of objective questions of the true-false and multiple choice types. It is administered at one fixed time only. Several days elapse before information on the results is available to the students, and often the only information provided to the student is the total score, which is of little use to him in guiding his study. Such exams commonly are given infrequently and therefore cover an extensive amount of material.

There are several objectionable consequences of these procedures. First, the conventional large class examination is not adequate to motivate routine study. The infrequency of testing gives rise to the well known "loaf and cram" pattern of study. Second, conventional infrequent exams provoke excessive anxiety in the student. The exam covers a large amount of course material and accounts for a substantial part of a student's grade. A student who is not well up on his reading or is just not well will be very anxious about his performance on the exam; further, a poor performance may cause him to despair and abandon meaningful study in the course. Third, students exposed to multiple choice test questions become testwise. Unless test items are extremely well formulated, students develop the ability to *recognize* answers from among the choices, and they are thus not required to *recall* course material in order to construct a response to the question. Unfortunately, this recognition ability is not likely to result in a significant increase in active vocabulary or intellectual competence. Fourth, the necessity of writing new test questions semester after semester means that only in rare instances does the instructor have *a priori* objective evidence that each exam question validly discriminates between good and bad student performance.

A SOLUTION

The goal is to find a practical and economic way to overcome the principal deficiencies of conventional examinations for large classes. This necessitates using only those facilities readily available at most academic institutions. There are three fundamental ways in which the Computer Generated Repeatable Testing method departs from a conventional approach. First, students may be examined *more frequently*. This encourages students to keep up with their course work and allows them to evaluate their performance frequently. Second, the tests provide *immediate feedback* to the student. When a student turns in his answer sheet, he keeps his test questions and receives a matching list of correct answers to the items on the test along with study aids such as textbook page references.

Third, and most important, examinations have been made *repeatable*. The digital computer is used to generate individualized repeatable tests. Large numbers of unique but equivalent tests are generated by a computer program which takes stratified random samples from an item pool and prints out questions and answers in a format appropriate to test taking and machine or hand grading. Since the instructor may prepare large quantities of individualized tests, he may allow the student to take tests over an examination unit as often and whenever the student desires. Students are free to take a test and find out thereby what they have not mastered, review that material, and try again. In this way they can work up to a high level of proficiency in the content of the course.

Conventional examinations provide only a single opportunity to show knowledge, and diagnostic information cannot be used to improve one's grade. Our method of computer generated repeatable examinations encourages the student to use diagnostic information and to restudy material he has not initially mastered, and it thereby decreases the student's aversion of the examination process while maintaining appropriate demands for the mastery of the course content. The technique of repeatable testing, of allowing more than one chance to demonstrate competence with material, is an exceedingly attractive instructional procedure. In a single stroke it releases the professor from his conflict between being excessively demanding and transmitting important information; he can expect and demand that the student master the material because the student can work up to mastery through a series of study sessions and examinations. Similarly it releases the student from his conflict between fear and fatigue; he has the opportunity to set a humane pace of study for himself, because if that pace is insufficient to obtain a satisfying grade on the first test over a unit of material, he can increase his pace in order to succeed on subsequent tests covering that material. Work rather than worry is elicited from the student.

One should not consider the CGRT system to be a kind of Computer-Assisted Instruction,[1] since students are not in interaction with a computer. It resembles more what Cooley and Glaser[2] have termed "computer-managed instruction" since the computer is used to facilitate instructional examination processes. We strongly point out that CGRT does not suffer from the present serious technological and economic disadvantages of Computer-Assisted Instruction, but nevertheless shares many of its educational advantages.

THE CGRT PROCESS

The Computer Generated Repeatable Testing process typically consists of four steps: (1) developing pools of test items, (2) producing tests, (3) administering the tests, and (4) scoring the tests. The second and fourth steps are managed by computer, while the execution of the first and third steps is strongly influenced by the computerized nature of the process.

Developing test item pools

For each exam, the course instructor develops a pool of items (test questions) which forms the data base from which tests are prepared. This is a rather formidable step. Our experience indicates that one should have about six to ten items in the pool for each question on an exam to assure adequate variation on the individual tests. An instructor planning to give eight exams of twenty questions each should construct about fifteen hundred individual items for his course. This work, which is every bit as tedious and time consuming as it sounds, should be done prior to the first semester in which repeatable testing is to be used in the course. Fortunately, the item pools, once developed, are rather permanent, especially for the basic college undergraduate courses that are the most likely candidates for this computerized testing scheme. Only relatively minor alterations to the item pools are needed to accommodate other instructors, changes of texts, etc., that may occur in subsequent semesters. Further, textbook publishers often have compendiums of test questions for their popular texts. Reusing test questions semester after semester, or even making the entire item pool available to students, is not a disadvantage under our procedure, and in fact is likely to be distinctly advantageous!

Since items will ultimately appear on computer generated tests, the form of the items must conform to the requirements of present computer printing technology. Normally, items may consist of upper-case letters, numbers, and the usual special characters available on modern high-speed line printers. Diagrams, pictures, and other graphic aids usually cannot be printed directly, although the instructor may easily include these by providing the student with a supplementary sheet of diagrams to accompany the tests.

If the tests are to be graded manually, technology imposes no limitation on the structure of the answer to an item. The test questions may elicit objective or subjective responses from the student. On the other hand, if the instructor wishes to use mechanical grading techniques, he must provide for a *single-character* response for each item, because of restrictions imposed by the optical mark sense form readers usually available in universities. While this requirement may appear to be a severe limitation, it in fact allows considerable freedom in the form of objective test items. True-false and multiple choice items call for single character responses. Key-word, fill-in, and other forms resulting in a definite numeric or symbolic answer may easily be reduced to a single character response using the following convention: In such a question the form of the answer is indicated by a series of dots which includes one asterisk. The student will construct his symbolic or numeric answer to the question, and will record as his response on his mark sense form the single character selected by the position of the asterisk in the string of dots. For example, ..*. means code the third letter or digit of the answer, *... means the first letter or digit, and so forth. The notation .*.. appearing in a fill-in-the-blank question calling for the answer "INTEGRAL" would require the student to mark the "N" space on his answer sheet. Students describe such alphabetically or numerically coded items as being hard but fair. The student cannot answer such an item unless he has mastered the basic concepts and vocabulary. Recall is emphasized; simple recognition is subordinated.

In addition to the question part of an item, which the student sees when he takes a test, each item also has an answer part to allow machine grading and to provide information to the student after testing. The answer part of an item may contain, in addition to an answer character, any relevant information, such as the full symbolic or numeric answer, textbook page references, and other diagnostic aids for the student.

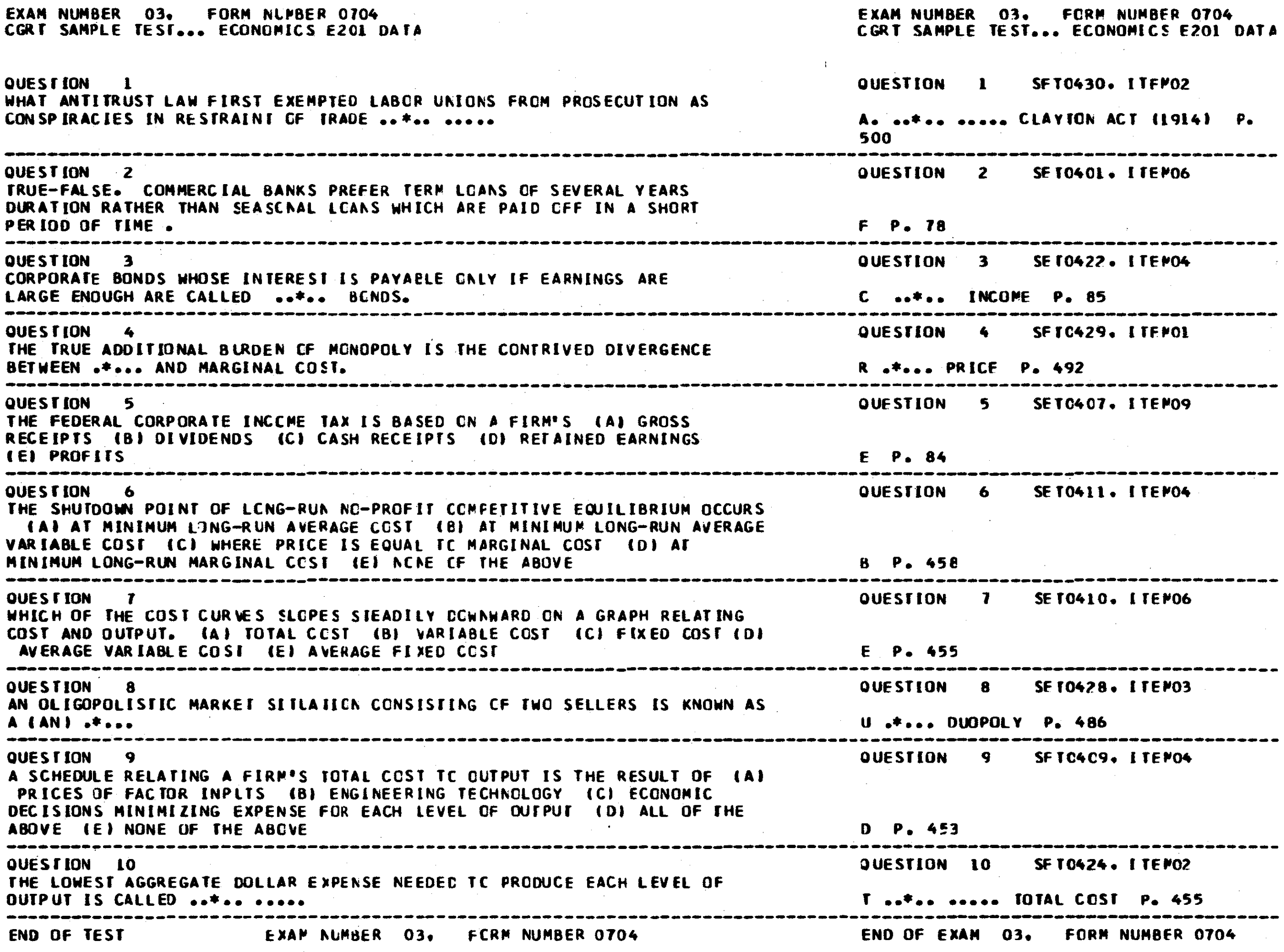
After the instructor has developed a section of his test item pool, he will have it punched onto punch cards or entered into an appropriate editable data file system. To facilitate the selection of items for an individual test and to maintain order among the large item pools the instructor classifies his items into sets, the items within a set are given distinct unit numbers, and cards or lines for the question part and the answer part of each item are numbered serially. Usually a set will consist of those items that test similar material. The use of set numbers is explained in the next section.

Producing tests

The individualized tests are generated on a digital computer using a computer program GENERATOR. This program, which is described in more detail in a later section, reads the item pool for a particular exam, checks the input data for proper sequencing and correct format, reads information describing the tests to be generated (number of tests, number of questions per test, etc.), generates and prints the individual tests, and punches a small answer summary deck for use in mechanized grading. The appearance of the tests is similar to the photo-reduced sample in Figure 1. Each test is individually numbered and has questions on the left part of the line printer page and answers on the right. The item identification numbers for each question appear in the answer part for reference. The instructor will of course separate the answer part from the question part prior to giving a test to the student.

The computer program selects items for a test by randomly choosing an item from each set. The order of choosing sets is also randomized. No item is used more than once per test. The digital computer is vital to test production, since the random item selection, formatting, and printing of large numbers of individualized tests is beyond the capacity of nonautomated operations. In the sample test in Figure 1 each item is assigned equal weight. The instructor may also assign weights (point values) to sets of items, thus allowing him to emphasize particular topics or award points based on the difficulty of items.

The computer time required to generate the tests is very small; the time required to print tests is, however,



```
EXAM NUMBER  03.   FORM NUMBER 0704                                           EXAM NUMBER  03.   FORM NUMBER 0704
CGRT SAMPLE TEST... ECONOMICS E201 DATA                                       CGRT SAMPLE TEST... ECONOMICS E201 DATA


QUESTION   1                                                                  QUESTION   1    SET0430. ITEM02
WHAT ANTITRUST LAW FIRST EXEMPTED LABOR UNIONS FROM PROSECUTION AS
CONSPIRACIES IN RESTRAINT OF TRADE ..*.. .....                                A. ..*.. ..... CLAYTON ACT (1914)  P.
                                                                              500
------------------------------------------------------------------------------------------------------------------------
QUESTION   2                                                                  QUESTION   2    SET0401. ITEM06
TRUE-FALSE.  COMMERCIAL BANKS PREFER TERM LOANS OF SEVERAL YEARS
DURATION RATHER THAN SEASONAL LOANS WHICH ARE PAID OFF IN A SHORT
PERIOD OF TIME .                                                              F  P. 78
------------------------------------------------------------------------------------------------------------------------
QUESTION   3                                                                  QUESTION   3    SET0422. ITEM04
CORPORATE BONDS WHOSE INTEREST IS PAYABLE ONLY IF EARNINGS ARE
LARGE ENOUGH ARE CALLED  ..*..  BONDS.                                        C  ..*..  INCOME  P. 85
------------------------------------------------------------------------------------------------------------------------
QUESTION   4                                                                  QUESTION   4    SET0429. ITEM01
THE TRUE ADDITIONAL BURDEN OF MONOPOLY IS THE CONTRIVED DIVERGENCE
BETWEEN .*... AND MARGINAL COST.                                              R .*... PRICE  P. 492
------------------------------------------------------------------------------------------------------------------------
QUESTION   5                                                                  QUESTION   5    SET0407. ITEM09
THE FEDERAL CORPORATE INCOME TAX IS BASED ON A FIRM'S  (A) GROSS
RECEIPTS  (B) DIVIDENDS  (C) CASH RECEIPTS  (D) RETAINED EARNINGS
(E) PROFITS                                                                   E  P. 84
------------------------------------------------------------------------------------------------------------------------
QUESTION   6                                                                  QUESTION   6    SET0411. ITEM04
THE SHUTDOWN POINT OF LONG-RUN NO-PROFIT COMPETITIVE EQUILIBRIUM OCCURS
 (A) AT MINIMUM LONG-RUN AVERAGE COST  (B) AT MINIMUM LONG-RUN AVERAGE
VARIABLE COST  (C) WHERE PRICE IS EQUAL TO MARGINAL COST  (D) AT
MINIMUM LONG-RUN MARGINAL COST  (E) NONE OF THE ABOVE                         B  P. 458
------------------------------------------------------------------------------------------------------------------------
QUESTION   7                                                                  QUESTION   7    SET0410. ITEM06
WHICH OF THE COST CURVES SLOPES STEADILY DOWNWARD ON A GRAPH RELATING
COST AND OUTPUT.  (A) TOTAL COST  (B) VARIABLE COST  (C) FIXED COST (D)
 AVERAGE VARIABLE COST  (E) AVERAGE FIXED COST                                E  P. 455
------------------------------------------------------------------------------------------------------------------------
QUESTION   8                                                                  QUESTION   8    SET0428. ITEM03
AN OLIGOPOLISTIC MARKET SITUATION CONSISTING OF TWO SELLERS IS KNOWN AS
A (AN) .*...                                                                  U .*... DUOPOLY  P. 486
------------------------------------------------------------------------------------------------------------------------
QUESTION   9                                                                  QUESTION   9    SET0409. ITEM04
A SCHEDULE RELATING A FIRM'S TOTAL COST TO OUTPUT IS THE RESULT OF  (A)
 PRICES OF FACTOR INPUTS  (B) ENGINEERING TECHNOLOGY  (C) ECONOMIC
DECISIONS MINIMIZING EXPENSE FOR EACH LEVEL OF OUTPUT  (D) ALL OF THE
ABOVE  (E) NONE OF THE ABOVE                                                  D  P. 453
------------------------------------------------------------------------------------------------------------------------
QUESTION  10                                                                  QUESTION  10    SET0424. ITEM02
THE LOWEST AGGREGATE DOLLAR EXPENSE NEEDED TO PRODUCE EACH LEVEL OF
OUTPUT IS CALLED ..*.. .....                                                  T ..*.. ..... TOTAL COST  P. 455
------------------------------------------------------------------------------------------------------------------------
END OF TEST              EXAM NUMBER  03.   FORM NUMBER 0704                  END OF EXAM  03.   FORM NUMBER 0704
```

Figure 1—Sample individualized computer generated test with answers attached

substantial. Typical times on the Indiana University CDC 3600 computer system are about four minutes of computer time (of which about 20 seconds are for item selection) to generate 1,000 three-page tests, and about three hours of printer time to print them. As we show later, the total cost per test is about 5¢. This compares well with the 5¢ cost per test for conventional exams using standard office facilities!

To avoid grief caused by possible computer delays and human errors, an instructor should submit his test production runs to the computing facility well in advance of his need. With many hundreds of students eager and ready to be examined on the course material, the instructor should risk no delays in preparing the tests. He need take no special precautions against pilfering of tests or even of listings of entire test item pools. The tests are individualized, and the item pools are large enough that the memorizing of the whole question pool is not a fruitful approach. (Indeed, as we implied earlier, a potentially useful study aid is to make the entire pool of test questions and answers available to the students prior to examination times).

Administering the tests

The instructor decides for himself how and when to test his students. He may give tests in class or at other scheduled times; or, more flexibly, he may allow his students to choose their own times for testing. A combination of in-class testing followed by opportunities for student-scheduled retesting appears to be useful. Such options depend on the instructor's preference and the availability of testing room space and personnel.

A student taking a test usually obtains an individualized test (with answer part removed) and a mark sense form and special pencil. He takes a seat in the testing area and immediately enters on his mark sense form his student identification number (social security number or other agreed-upon identifier), the exam number, and his individual test number. The student then marks his answers on his test, and for each question enters the appropriate single-character response on his mark sense form. After completing a test, the student exchanges his mark sense form for the answer part of his individual test. The mark sense form is kept by the proctor for later grading. The student, having the correct answers in hand, can immediately determine his errors, and is stimulated to improve his knowledge of weak areas. Since the tests are individualized, the student may repeat the examination at later times, within the constraints imposed by the instructor.

Scoring tests

The instructor and his assistants may of course grade tests manually if they desire. However, computerized grading of the individualized tests is usually desirable, and may be performed using the information on the student's mark sense forms. Since the student has received the answer part of his test in exchange for his filled in mark sense form, there is no necessity for undue haste in grading the tests. The instructor or his assistant will, whenever convenient, have the information on the mark sense forms transformed to punch cards on an optical mark sense form reader. This step is required to obtain a form of input acceptable to the typical academic computing facility; one can bypass this step if optical mark sense form reading equipment is attached directly to his institution's computing equipment.

Scoring of the student responses for an exam is done by computer using a program GRADER. Input to this program is the answer summary deck punched by program GENERATOR when the tests were prepared, and the student response cards derived from the mark sense forms. Output of this program is a roster of student ID's and test scores and a punch card deck of the high score for each student for this exam.

Most academic institutions have available cumulative grading computer programs. These permit exam grades to be accumulated, and aid in the eventual preparation of final grades by generating score distributions and other statistics. The card deck prepared by GRADER is for use with such cumulative grading systems.

As a followup of test scoring, we are developing an item analysis procedure for CGRT. Since the item pools tend to be reused many times, such an item analysis will aid in the detection of defective test items and will assist the instructor in polishing his item pool.

THE COMPUTING PROCESSES

The test producing program GENERATOR and the scoring program GRADER are written in Fortran. Virtually all academic computing centers have well-maintained Fortran compilers that produce a fairly good quality of object code. We have several versions of the CGRT programs: well-documented ANSI Fortran versions designed to run on all commonly-available computers, and specialized versions of GENERATOR for the CDC 3600 and for the CDC 6600. The specialized versions utilize CDC extensions of ANSI Fortran to decrease the execution time dramatically by bypassing the repetitive processing of format statements during test printing. Since GENERATOR is completely output-bound, we anticipate that many potential users of the ANSI Fortran version would wish to discuss modification of the program with their systems people to take advantage of local extensions to Fortran output facilities.

GENERATOR reads the test item pool, checks each record for consistency of identification information, and creates a condensed file of the test item questions and answers, partially formatted for output. This item file is kept in primary storage. A directory of the origins of sets and individual items is formed to provide rapid retrieval of item information for test printing. GENERATOR then reads directives for test production: an arbitrary header line for all tests, an exam number, the number of individualized tests wanted, the number of items (questions) on each test, the test number of the first test (others are numbered sequentially from the starting number), and possibly other data to select additional options. A file is written for punching which records the set number, item number, and answer character of all test items. This deck, which is typically about 25 cards, is used by GRADER to regenerate the sequence of items and answers in a given test.

Then for each test, the program selects question items and writes the test onto an output file. For each test, a pseudo-random number generator is initialized with a unique but reproducible number. Using the "random" but reproducible sequence of numbers from the random number routine, GENERATOR determines the order of questions on a test by random selection without replacement of sets followed by random selection without replacement of an item from each set until the requisite number of questions is chosen. If an item from each set is used and questions remain to be chosen, the process repeats. When selection is complete, the question

TABLE I—Cost Analysis of CGRT[a]

ITEM	COST
Punch cards for item pools (One-time expense)[b]	$ .40
Punch cards for student responses: 1000 cards	1.00
Printer paper: 3000 sheets	8.30
Mark sense forms: 1000 forms	8.80
Keypunching services for item pool punching (one-time expense)[b]	6.50
Computer charges for test production and grading: about 5 minutes @ $200 per hour[c]	16.70
High speed line printer and controller rental and maintenance: @ $1800 per month[d]	6.00 to $12.00
Total expenses	$47.70 to $53.70

AVERAGE COST PER TEST: 4.8¢ to 5.4¢

[a] for 1000 three-page tests.
[b] prorated over four semesters.
[c] Indiana University CDC 3600 system.
[d] CDC 512 printer system.

and answer text for each selected item is formatted and written. Program control then returns to prepare the next test.

GENERATOR also has several optional facilities, such as multiple copies of tests, an answer summary for the instructor, and a method of assigning point values to items to allow weighting of the items during scoring.

Since the amount of output is substantial and on-line secondary storage is limited, most people will find it convenient to write the tests as blocked records on a magnetic tape. The computer center staff may then print the tape at a convenient time.

GRADER accepts as input the item pool answer summary deck punched by GENERATOR and the student responses punched from mark sense forms. To grade a student's response to a particular test, GRADER uses the same item selection algorithm as GENERATOR to recreate the same sequence of items and answer characters. The student's score is formed as the sum of the values of each correctly answered item. The score, the test number, and the student's ID number are saved. When all student responses have been graded, GRADER sorts the ID's and test scores, and a roster is produced showing for each student his scores, highest first, and the test numbers. As a final step, a punch card summary of the roster is prepared for use in possible later cumulative grading operations.

THE ECONOMICS OF CGRT

At first glance a procedure that uses a computer for test preparation and for printing of individualized tests appears economically unsound. This is very definitely not the case. In Tables I and II, which are cost analyses for the preparation, printing, and scoring of 1000 three-page tests, we have attempted to itemize expenses in a similar manner for both CGRT and the conventional method. Therefore, the cost of a computer line printer and associated equipment has been treated as a separate entry, rather than included in general computer charges. We have assumed that such expenses as the initial keypunching of item pools are distributed over four semesters.

The analyses show that both CGRT and conventionally prepared tests cost about 5¢ per test. While the estimate for conventional exams is fairly accurate, changing some of the assumptions in the CGRT analysis may alter the estimate by perhaps up to two cents per test. Also, under repeatable testing, students tend to take more than one repeatable test over each examination unit. In any event, the cost of repeatable tests is in the same range as conventional tests. More important, the expenses of the CGRT process are a very minor item in the cost of educating the student, amounting to $.50 to $1.50 per student per course. This is inexpensive education!

SUMMARY

Computer Generated Repeatable Testing works. It has been used in numerous courses for nearly three years at Indiana University, and it is also in use at the University of Nebraska, Illinois Institute of Technology, Indiana University-Purdue University at Indianapolis, and

TABLE II—Cost Analysis of Conventionally Prepared Tests.[a]

ITEM	COST
Paper: 3000 sheets	$ 6.00
Mark sense forms: 1000 forms	8.80
Punch cards for student responses: 1000 cards	1.00
Clerical services @ $4.00 per hour:[b]	
Typing: 1½ hours	6.00
Multilithing: 2 hours	8.00
Collating and stapling: 4½ hours	18.00
Computer charges for grading: about 1 minute @ $200 per hour[c]	3.30
Total expenses	$51.10

AVERAGE COST PER TEST: 5.1¢

[a] for 1000 three-page tests.
[b] estimates supplied by Indiana University Chemistry Department.
[c] Indiana University CDC 3600 system.

other places. The method has been enthusiastically used by instructors of undergraduate courses in such varied disciplines as psychology, chemistry, computer science, economics, English, speech therapy, home economics, accounting, and education.

In general, students have been highly satisfied with the repeatable testing method. Their mood is one of alertness rather than anxiety. They are relaxed during examinations, and their morale is good. The undergraduate counselling units of Indiana University have received numerous student comments favorable to CGRT.

An unexpected result in some of the CGRT courses has been the students' excellent performance on technical material not discussed in class. Repeatable examinations appear to provide a stimulus and a way to master material typically neglected by students in conventional courses. Although we have only a little data taken under properly controlled conditions, indications from several common achievement tests given at Indiana University are that overall student achievement in repeatably tested sections is higher than in conventionally tested sections of the same course.[3] All available evidence suggests that a system of frequent and repeatable examinations provide an excellent atmosphere for scholarly activities of beginning students.

We hope that many readers will wish to try CGRT or suggest its use to their non-computer-oriented colleagues. The computer programs and ample documentation are available from Franklin Prosser.

REFERENCES

1 P SUPPES M MORNINGSTAR
Computer-assisted instruction
Science Volume 166 pp 343–350 1969

2 W C COOLEY R GLASER
The computer and individualized instruction
Science Volume 166 pp 574–582 1969

3 D JENSEN F PROSSER
Computer-generated, repeatable examinations and large class instruction
Presented at Midwestern Psychological Association Meeting Chicago Illinois 1969

R2—A natural language question-answering system*

by K. BISS, R. CHIEN and F. STAHL

University of Illinois at Urbana-Champaign
Urbana, Illinois

INTRODUCTION

A large number of systems involving computers require a high degree of man-machine interaction. In these systems the capability of the computer to process natural language information would be extremely useful as that eliminates the need for the user to learn various formal languages for the purpose of interaction.

In view of the fact that computational costs are decreasing while programming costs are increasing rapidly, computers with natural language processing capabilities will become practical for many applications. A good way to study natural language man-machine communication is through the development of question-answering systems; for it is in these systems that a well-defined natural language discourse takes place.

Research aimed at developing intelligent question-answering systems has been carried on for over a decade. For information concerning these systems we refer the reader to Simmons.[13,14] Of the more advanced question-answering systems Green and Raphael[6,7] have developed a very powerful deductive procedure and Simmons, et al.[12,15] introduced an extremely attractive representation scheme.

Reported in this paper is an intelligent natural language question-answering system called the R2 system. The system introduces a variety of new techniques and encompasses the features found in previous systems. It accepts a wide range of input sentences in English and deduces answers to input questions based upon available information. The data base chosen to demonstrate the capabilities of the system is the Illinois Drivers Manual, *Rules of the Road,* which consists of about 2000 English sentences.

This particular data base was chosen because its contents are representative of the type of facts that one normally encounters. In addition, the number of facts associated with traffic laws is large enough not to be considered trivial as verified by the fact that it is larger than any other data base currently being used for natural language question-answering systems; yet it is small enough to be handled adequately by our present computer configuration.

The internal representation in the R2 system is based upon a high-order logical calculus that permits the expression of wide range of natural language information. Input to the system undergoes syntactic and semantic analysis in order to be transformed into the internal representation scheme. A recursive goal-oriented theorem-proving algorithm is used to deduce answers to questions posed to the system.

In the second section we comment on some natural language question-answering systems by pointing out some of the features found in these systems. In the third section is described the R2 system and the high-order calculus that is used for the internal representation of natural language information. Finally, an example showing the processing of a typical question is presented.

COMMENTS ON SOME NATURAL LANGUAGE QUESTION-ANSWERING SYSTEMS

Generally, natural language question-answering systems use some formal internal representation for facts and questions in order to facilitate deductive manipulations. In a number of earlier systems the representation was based upon some type of limited relational calculi, as for example Raphael's SIR,[9] and Black's SQA.[5]

Green and Raphael[6,7] subsequently developed a system that offered the full expressiveness of the first-order predicate calculus for the representation of natural language information. The deductive procedure of this

* This work was supported by the Office of Education under Contract No. OE C-1-7-071213-4557 and by the Joint Services Electronics Program (U.S. Army, U.S. Navy, and U.S. Air Force) under contract DAAB-07-67C-0199.

system was based on an automatic theorem-proving algorithm that was first described by Robinson[10] and improved upon by Wos, et al.[16,17,18,19] and others.[1,2,8]

The use of first-order predicate calculus as a formal language for the representation of natural language information when used in conjunction with automatic theorem-proving procedures is a significant improvement over previous schemes. However, the first-order predicate calculus cannot be used to express relationships between relations, or allow variables to range over relations as well as objects.

For example, suppose it is necessary to put into the first-order language the sentences

John crossed the street after the light changed. (1)

or

A car must always yield to a pedestrian. (2)

In (1) we are unable to put the sentence into the first-order language because we have a relation, namely *after*, whose arguments are forced to be relations, namely *crossed* and *changed*, rather than some individuals. In (2) we cannot put the sentence into the first-order language because we are faced with the quantification of a variable which ranges over situations not individuals. That is, the sentence states that for all possible situations a certain condition holds (i.e., that a car must yield to a pedestrian).

Simmons, et al.[12,15] developed a system that used nested binary relations for the formal representation of natural language information. The relations they used were of the form (aRb) where a and b could in turn be nested binary relations and R could represent a complex relationship. Note that this scheme overcomes the representational problems mentioned above for the first-order predicate calculus. However, the system still lacks the capability to handle either negation or quantification.

The R2 question-answering system uses a high-order formal language for the internal representation of information and a recursive theorem-proving procedure for performing the necessary deducive operations. This system can represent relations between relations and quantification of variables ranging over rather complex structures. In addition, the goal-oriented theorem-proving procedure performs sophisticated logical deduction through the use of contextual information. In the section that follows a description of the R2 system is given.

THE R2 NATURAL LANGUAGE QUESTION-ANSWERING SYSTEM

An intelligent natural language question-answering system must be capable of performing sophisticated linguistic processing of input information in order to arrive at a well-structured internal representation upon which extensive logical processing can be performed. That is, any high quality natural language question-answering system must have:

1. an internal data structure sufficiently rich to represent the subtle semantic differences found in natural language information;
2. a method of transforming natural language into that structure; and
3. a strong deduction algorithm for manipulating the information in that structure.

The R2 question-answering system has been designed to encompass these features. A parser and a semantic converter are used to transform natural language information into the high-order language. A deduction algorithm manipulates the facts represented in the high-order language to synthesize the necessary information for the answering of questions posed to the system.

The parser (or syntactic analyzer) breaks down a sentence to show the structural relationships among the parts of the sentence. The parsing indicates what words modify other words (and which words they modify); what are the subject, object, and predicate of the sentence, etc. This type of information is essential to any system that utilizes natural language in an intelligent manner. If, for example, a sentence relating to some action is given, then it is necessary to differentiate between that which is performing the action, and that upon which the action is being performed. The syntactic analysis of that sentence indicates this type of information.

The semantic converter accepts the syntactically parsed statements (i.e., facts and questions) and determines if these parsings are semantically well formed. If the statement is well formed, it is converted into the high-order language, and if it is not, then it is returned to the parser.

The deducer receives its information from the semantic converter expressed in the high-order language. Included in this information is a formal statement of the question that the system is attempting to answer. Through a recursive goal-oriented deductive procedure the necessary implicit information is generated. This new information is then used to answer the question.

The high-order language

The high-order language used in the R2 system contains simple objects like *car, driveway, pedestrian*; more complex objects like *green car*, or *a car in a driveway*; relations like 'a car *yields* to a pedestrian'; relations between relations like 'John crossed the street *after* the light changed'; and variables that range over these various entities.

Simple type objects like *car*, *driveway*, and *pedestrian* are represented by character strings identical to their orthographic representation. The more complex objects that are modified in some way like *green car* and *a car in a driveway* are represented as the unary relations *green(car)* and *(in(driveway))(car)*, respectively, where the modifier becomes the relation symbol and the object becomes the argument of that relation. Of course, a modifier may also modify an object that is already modified. Thus, *a green car in a driveway* would become *(in(driveway))(green(car))*.

In order to appreciate this formalism think of *green* as a function whose value is equal to its argument with the additional property of being green. Thus, anything that applies to the object *car*, without qualification, also applies to the modified object *green(car)*.

A relation like 'a car *yields* to a pedestrian' is represented as an n-ary relation symbol followed by the appropriate arguments. In this case, the relation is binary and would be represented as

yield(car, pedestrian).

In general, an n-ary relation may take as arguments any objects, relations, or variables (so long as they make sense in the domain of discourse being considered). These nested relations may be connected with the logical symbols $\wedge$, $\vee$, $\sim$, and $\rightarrow$ as in other logical languages, and quantification may occur over any variables appearing in these expressions. For a formal definition of this language see Biss, et al.[3]

Using this language we can now represent the sentences which we could not handle previously. Thus, the sentence

John crossed the street after the light changed

would become

after(cross(John, street), change(light))

where *after* is, in this example, a binary relation relating *cross* and *change*. In the same manner, the sentence

A car must always yield to a pedestrian

would become

$\forall$y(y$\rightarrow$must(yield(car, pedestrian)))

where y is a variable that ranges over structures such as

(in(driveway))(car)

or

(in(crosswalk))(drunk(pedestrian)).

Question-answering in the R2 system

To best illustrate the performance of the R2 system let us look at an example. Suppose the system receives the question:

Do cars always have to yield to pedestrians? (3)

and it has at its disposal the facts:

Pedestrians not in a crosswalk must yield to cars. (4)

And

If x must yield to y

then y does not have to yield to x. (5)

from the data base already stored in the formal language.

After the question has entered the system it is transformed into the internal formal language and is posed as a theorem for the system to prove. If the theorem is proven, the answer to the question is *yes*; if the negation of the theorem is proven, the answer is *no*.

The parser starts the transformation process by noting that in (3) *cars* is the subject, *yield* the verb, *pedestrian* the object, *always* an adverb modifying the main verb, *have to* is an auxiliary, etc. The grammar used for the syntactic analysis is a modified context-free immediate constituent phrase-structure grammar.[10] The form of the output from the parser is

PREDICATE(modifiers)(SUBJECT(modifiers),

OBJECT(modifiers))

Thus, the subject and the object of the sentence are the arguments of the predicate.

The syntactic information is given to the semantic converter which then puts the sentence into the high-order language, as in (6). (Notice that the question word *do* is dropped).

always(have to(yield(car, pedestrian))) (6)

This statement is now taken to be the theorem to be proven. At this point the expression *have to* is replaced by *must* in order to normalize the text. Thus we have:

always(must(yield(car, pedestrian))). (7)

The converter must now check to make sure that (7) does not conflict with what is known about the real world. This is done by using both the semantic rules

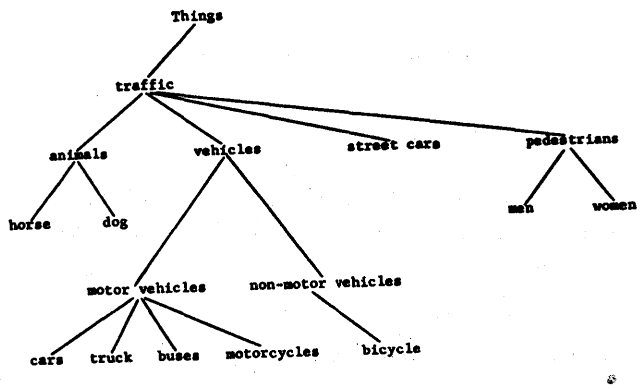


Figure 1

for this domain:

$$\text{yield(traffic, traffic)} \qquad (8)$$

$$\text{must(n-ary relation)} \qquad (9)$$

$$\text{always(n-ary relation)} \qquad (10)$$

and the tree of Figure 1 as the axioms of a system which must prove the well-formedness of (7).

The tree in Figure 1 gives the information that *cars* and *trucks* are *motor vehicles*, which are *vehicles*, which are *traffic*; and that *pedestrians* are *traffic*.

From (10) the converter knows that in order for (7) to be considered well formed it must be shown that

$$\text{must(yield(car, pedestrians))} \qquad (11)$$

is a well-formed n-ary relation. But, from (9) we know that (11) is a well formed unary relation if

$$\text{yield(car, pedestrian)} \qquad (12)$$

is a well-formed n-ary relation. From (8) we know that (12) is a well-formed binary relation if it can be shown that *car* is a type of *traffic* and *pedestrian* is also a type of *traffic*. But from the tree in Figure 1 it is known that both *cars* and *pedestrians* are *traffic*, and, therefore, the *yield* relation is well formed, consequently (11) is well formed. Thus, (7) is well formed and the semantic converter concludes that what it was given makes sense in the real world.

The last task that the semantic converter must perform is that of replacing *always* in (7) with a quantifier. Thus (7) would become

$$\forall y(y \rightarrow \text{must(yield(car, pedestrian))}) \qquad (13)$$

where y is a variable ranging over situations. The formula in (13) states that under all situations cars must yield to pedestrians, which is just what (7) says.

But (13) implies

$$\forall x_1 \forall x_2(\text{must(yield}(x_1(\text{car}), x_2(\text{pedestrian})))) \qquad (14)$$

where x_1 is a variable ranging over situations on *car*, and x_2 is a variable ranging over situations on *pedestrian*.

The deducer receives (14) and tries to prove that it is a theorem. The theorem is proven by contradiction using a recursive goal-oriented theorem-proving procedure. That is, an attempt will be made to show that the negation of (14) contradicts the information which the system has. In this particular case the system has (4) and (5) stored in the high-order language as:

$$\text{must(yield((not(in(crosswalk)))(pedestrian), car))} \qquad (15)$$

and

$$\forall x \forall y[\text{must(yield}(x, y)) \rightarrow \sim\text{must(yield}(y, x))] \qquad (16)$$

respectively.

At first, the deducer tries to prove that (14) is true by showing that (17) contradicts the relevant axioms, namely (15) and (16) in this case.

$$\sim\forall x_1 \forall x_2(\text{must(yield}(x_1(\text{car}), x_2(\text{pedestrian})))) \qquad (17)$$

The system will, of course, not be able to prove (14) since it is not true, as can be seen by looking at (15). Thus, the system eventually tries to prove that (14) is false, i.e., that (17) is a theorem. In order to prove that (17) is a theorem, we try to prove that its negation (14) contradicts the axioms (15) and (16). Now, (16) is rewritten as

$$\sim\text{must(yield}(x, y)) \vee \sim\text{must(yield}(y, x)) \qquad (18)$$

since $A \rightarrow B$ is equivalent to $\sim A \vee B$, where universal quantification has been made implicit. Then,

$$\sim\text{must(yield}(x_2(\text{pedestrian}), x_1(\text{car}))) \qquad (19)$$

follows from (14) and (18) by recursively applying the high-order resolution on them. Now, (19) resolves with (15) if we let $x_2 \equiv \text{not(in(crosswalk))}$ and $x_1 \equiv \emptyset$ (the empty substitution), generating a contradiction. Thus, (17) which means

cars do not always have to yield to pedestrians

is true. Therefore, the answer to the question

Do cars always have to yield to pedestrians?

is *no*.

Each step of the deduction may, as in this example, appear as merely the matching of two identical terms or the instantiation of some variable. It should be noted that the theorem-proving procedure can handle much more complex deductions than this. Specifically, the procedure recursively attempts to perform deduction at *every* level of nesting. For a more detailed description of this procedure along with other examples involving more complex deductions see Biss, et al.[4]

Now that the basic procedure has been illustrated for questions of the *yes/no* type, we can consider the questions of the form

$$\exists x \psi(x)?$$

i.e., 'Is there an x such that $\psi(x)$ is true?' where $\psi(x)$ is any statement involving the variable x. The deduction algorithm attempts to produce a substitution instance p such that $\psi(p)$ is true.

Suppose we ask under what conditions must a pedestrian yield to a car?

which is represented formally as

$$\exists x(x \rightarrow \text{must}(\text{yield}(\text{pedestrian}, \text{car})))$$

or equivalently as

$$\exists x_1 \exists x_2(\text{must}(\text{yield}(x_1(\text{pedestrian}), x_2(\text{car})))). \quad (20)$$

The result of the deduction procedure is that the negation of (20) contradicts the axioms. Therefore, the answer to the question is the substitution instance that was used for the variables in order to arrive at the contradictions. That is, the answer is

when

$$x_1 \equiv \text{not}(\text{in}(\text{crosswalk}))$$

$$x_2 \equiv \emptyset$$

or equivalently

when pedestrian not in the crosswalk.

CONCLUSION

The R2 system described in this paper exhibits a number of advanced features not found in existing natural language question-answering systems. It is based upon a high-order logical calculus that allows the embedding of relations, and quantification over rather complex structures thus permitting the expression of a wide range of natural language information.

Two factors, the ease of transforming natural language information into the high-order representation, and the existence of a high-powered recursive deduction algorithm, have made R2 a very powerful system indeed. The techniques developed for the system are directly applicable to many other factual English data bases without modification. These techniques may also be applied to other areas that might involve automated natural language processing, such as: computer-aided instruction, management information systems, or document retrieval systems.

REFERENCES

1 R ANDERSON W W BLEDSOE
A linear format for resolution with merging and a new technique for establishing completeness
Journal of the ACM Vol 17 No 3 pp 525–534 July 1970

2 P B ANDREWS
Resolution and merging
Journal of the ACM Vol 15 No 3 pp 367–381 July 1968

3 K O BISS R T CHIEN F A STAHL
A data structure for cognitive information retrieval
Coordinated Science Laboratory University of Illinois Urbana Illinois 1970

4 K O BISS R T CHIEN F A STAHL
Logical deduction in the R2 question-answering system
Coordinated Science Laboratory University of Illinois Urbana Illinois 1971 (to appear)

5 F S BLACK
A deductive question-answering system
Semantic Information Processing (ed M Minsky) MIT Press Cambridge Mass pp 354-402 1968

6 C C GREEN
Theorem-proving by resolution as a basis for question-answering systems
Machine Intelligence 4 (eds B. Meltzer and D Michie) Edinburgh Univ Press Edinburgh pp 151–170 1969

7 C C GREEN B RAPHAEL
The use of theorem-proving techniques in question-answering systems
Proceedings of the ACM National Conference pp 169-181 1968

8 D W LOVELAND
A linear format for resolution
University of Pittsburgh Dept of Computer Science 1968

9 B RAPHAEL
A computer program for semantic information retrieval
Sematic Information Processing (ed M Minsky) MIT Press Cambridge Mass pp 33–145 1968

10 J A ROBINSON
A machine-oriented logic based on the resolution principle
Journal of the ACM Vol 12 No 1 pp 23–41 January 1965

11 J A SCHULTZ W T BIELBY
An algorithm for the syntactic analysis in the R2 information system
Coordinated Science Laboratory University of Illinois Urbana Illinois 1970

12 R M SCHWARCZ J F BURGER R F SIMMONS
A deductive question-answerer for natural language inference
Communications of the ACM Vol 13 No 3 pp 167–183 March 1970

13 R F SIMMONS
Answering English questions by computer: A survey
Communications of the ACM Vol 8 No 1 pp 53-69 January 1965

14 R F SIMMONS
Natural language question-answering systems: 1969
Communications of the ACM Vol 13 No 1 pp 15-36 January 1970

15 R F SIMMONS J F BURGER R M SCHWARCZ
A computational model of verbal understanding
Proceedings of the Fall Joint Computer Conference Spartan Books pp 441-456 1968

16 L WOS G A ROBINSON D F CARSON
Some theorem-proving strategies and their implementation
Argonne National Laboratory Technical Memorandum No 72 Argonne Illinois 1964

17 L WOS G A ROBINSON D F CARSON
The unit preference strategy in theorem-proving
Proceedings of the Fall Joint Computer Conference Vol 26 pp 615-621 Spartan Books 1964

18 L WOS G A ROBINSON D F CARSON
Efficiency and completeness of the set of support strategy in theorem-proving
Journal of the ACM Vol 12 No 4 pp 536-541 October 1965

19 L WOS G A ROBINSON D F CARSON L SHALLA
The concept of demodulation in theorem-proving
Journal of the ACM Vol 14 No 4 pp 698-709 October 1967

Performance evaluation of direct access storage devices with a fixed head per track

by TRILOK MANOCHA, WILLIAM L. MARTIN and KARL W. STEVENS

International Business Machines Corporation
Kingston, New York

INTRODUCTION

Computer designers agree that the slow response of input and output devices is the most critical factor limiting the performance of a computer system. When a program requests an access to one of these devices, the request is put on a queue. The members of the queue are usually serviced on a first-come-first-served basis or on the basis of preassigned priorities. Devices with large capacity are able to store a number of data sets belonging to various programs in a multi-programming environment. This usually leads to a queue build-up for these devices, thereby creating a bottleneck in the system.

This paper treats devices of the type commonly used for most computer systems. They are rotating direct-access devices with a fixed head/track. Each track is divided into a number of blocks of data called sectors. Each member of the queue requests one of the blocks of data from the device. The transmission of data occurs when the required sector comes under the fixed read/write head.

One of the methods of reducing the bottleneck is to provide the device a capability to examine the queue. The device can then select the particular request that requires the minimum wait-time before its corresponding sector comes under the head. When the request is selected, the Central Processing Unit promptly initiates the selected request.

The queueing procedure is accomplished most effectively by a hardware queuer which automatically stacks the requests. The hardware limitations will allow only a fixed number of requests to be placed on this queue. Any additional requests will be handled by the software and added to the hardware queue as the earlier requests in the existing queue are satisfied.

Due to this procedure, the two parameters that affect the performance of the device most are queue size and the number of sectors per track. The most important assumption made is that each member of the queue has an equal probability of requesting any one of the sectors on a track.

Figure 1 illustrates the device structure. It shows that each track contains N sectors and a read/write head. The sector positions are aligned in such a manner that all sectors with the same sector number come under the fixed heads at the same time. Each member of the queue has a sector number associated with it. Though the head or track number is also needed to locate the actual block of data, we assume that the head-switching time is small enough to be neglected. In this way, if sector X on track Y is being transmitted and if a request in the queue requires sector $X+1$ on track Z, then the next sector transmitted will be sector $X+1$ from track Z. The switching of heads from Y to Z is done in negligible time.

THE STATE MATRIX

The integers at the left of the rows in Figure 2 represent the queue length for the device; those on top of the columns represent the number of distinct sector numbers associated with the members of the queue.

Figure 2 is designated as the state-matrix X, where X_{ij} represents the state of the device. In state X_{ij} there are j distinct sector numbers associated with i members of the queue. Each element X_{ij}, of the state-matrix represents the probability of the device being in state X_{ij}.

It is advantageous to consider the queue as consisting of small groups of members, where each group contains requests with the same sector number. Within each group, there is a particular request that will be satisfied before the other members of the group. This may be due to a first-come-first-served basis or a preassigned priority scheme. This particular request is called the

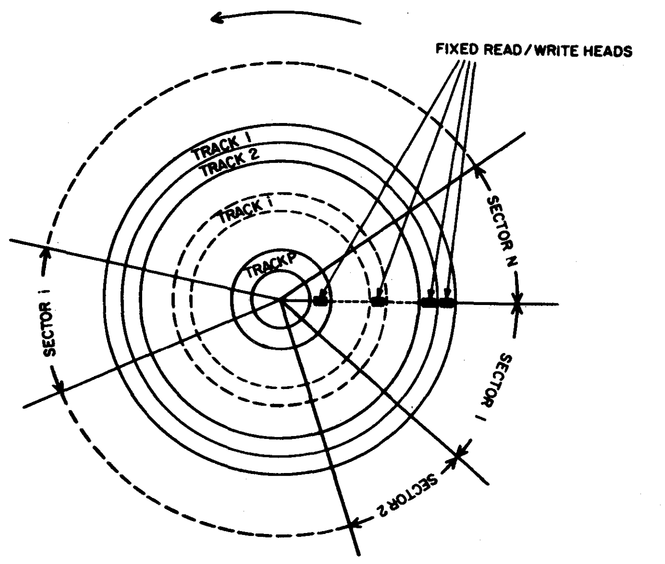


Figure 1—Device structure with distribution of tracks and sectors

representative element of the group and is the one in contention with representative elements from other groups in the queue. The number of distinct sector numbers associated with the members of the queue is equal to the number of representative elements in the queue.

Let us, at this stage, calculate the elements of the matrix if four random requests are introduced in the queue of a device with N sectors/track. We assume that N is greater than or equal to four.

When the first request is introduced, it is a representative request with probability 1. Therefore, element x_{11} is equal to 1.

Clearly, the device cannot exist in states X_{12}, X_{13}, or X_{14} since there cannot be more than one representative element in a queue of only one element. Therefore, elements X_{12}, X_{13} and X_{14} are equal to zero.

When the second request is introduced in the queue, the probability that it will occur for the same sector as the previous request is $1/N$. The probability that it will occur for a different sector is $(N-1)/N$. Therefore, elements x_{21} and x_{22} are $1/N$ and $(N-1)/N$ respectively. Since the device cannot exist in states X_{23} and X_{24}, elements x_{23} and x_{24} are equal to zero.

It is easy to see that all elements x_{ij} where $j>i$ will be zero since there cannot be more than i representative elements in a queue of i members.

When a third request is introduced, the probability that the device will jump to state X_{31} from state X_{21} is $1/N$. Since the probability of the device being in state X_{21} is $1/N$, element $x_{31}=1/N^2$.

State X_{32} can be reached from either of two previous states, state X_{21} or state X_{22}.

If the device is in state X_{21}, which means that both requests exist for the same sector, the probability that the third request occurs for a different sector is $(N-1)/N$. Therefore, the probability of reaching state X_{32} from state X_{21} is $(N-1)/N$. Or, the probability of reaching state X_{32} via state X_{21}, is $(1/N)(N-1/N)=(N-1)/N^2$.

If the device is in state X_{22}, the probability that the device will jump to state X_{32}, via state X_{22}, is $((N-1)/N)(2/N)$. Total probability of reaching state X_{32} is $(N-1)/N^2+2(N-1)/N^2=3(N-1)/N^2$.

State X_{33} can be reached only from state X_{22}. Since in state X_{22}, the two requests occur for different sectors, the probability of the third request occurring for a different sector is $(N-2)/N$.

The probability of reaching state X_{33} is

$$((N-2)/N)((N-1)/N)=(N-1)(N-2)/N^2.$$

When the fourth request is introduced, the device jumps to the states in the fourth row of the matrix.

The probability of reaching state X_{41} from state X_{31} is $1/N$. Since the probability of being in state X_{31} is $1/N^2$, element $x_{41}=1/N^3$.

The device queue has two representative elements among the four members of the queue when the device is in state X_{42}. This also means that there are two groups in the queue. There are two possible ways in which the four members can be distributed among the two groups. Three members in the first group and one member in the second group is a valid configuration. Similarly, two members in each group is also valid.

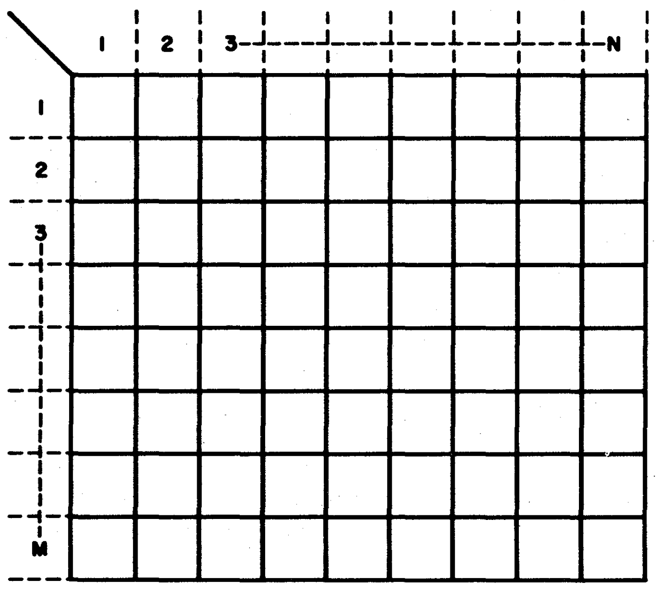


Figure 2—The state-matrix X

It is convenient at this point to break up state X_{42} into two substates. Substate X_{42}^1 requires a three-one combination for the membership of the groups and substate X_{42}^2 requires a two-two combination for the membership of the groups.

Substate X_{42}^1 can be reached from state X_{31} with probability $(N-1)/N$.

Substate X_{42}^1 can be reached from state X_{32} with probability $1/N$.

Therefore, the probability of reaching substate X_{42}^1

$$=x_{31}\cdot(N-1)/N+x_{32}\cdot 1/N=x_{42}^1$$

$$=(N-1)/N^3+3(N-1)/N^3=4(N-1)/N^3.$$

Substate X_{42}^2 can be reached from state X_{32} with probability $1/N$.

Since substate X_{42}^2 can be reached only from state X_{32}, element x_{42}^2

$$=x_{32}\cdot 1/N=3(N-1)/N^3.$$

Total probability of reaching state $X_{42}=x_{42}^1+x_{42}^2=7(N-1)/N^3$.

State X_{43} can be reached from state X_{32} or from state X_{33}. The probability of reaching state X_{43} if the device was in state $X_{32}=(N-2)/N$. From state X_{33}, the probability of reaching state $X_{43}-3/N$.

Therefore, element $x_{43}=x_{32}\cdot(N-2)/N+x_{33}\cdot 3/N$.

$$=3(N-1)(N-2)/N^3+3(N-1)(N-2)/N^3$$

$$=6(N-1)(N-2)/N^3$$

State X_{44} can be reached only from state X_{33}. The probability of reaching state X_{44} is $x_{33}\cdot(N-3)/N$.

$$=(N-1)(N-2)(N-3)/N^3.$$

For the particular case when there are six sectors/track, the state matrix is shown in Figure 3.

It follows that the number of rows in the state-matrix is equal to the queue length. The number of columns is equal to the queue length if the number of sectors/track is greater than the queue length. Otherwise, the number of columns is equal to the number of sectors/track.

When a request is removed from the queue, one of the groups loses a member. If the device was in state X_{ij}, it will jump to one of the states in the $(i-1)$th row. If the group that lost its representative element contains another member such that a new representative element can be created, the jump is to state $X_{(i-1)j}$. If the group contains no other member, the total number of representative elements in the queue is decreased by one and the jump is to state $X_{(i-1)j-1}$.

The new values of the elements of the $(i-1)$th row can be calculated if the probabilities of the jumps from the ith row are known. If state $X_{(i-1)j-1}$ can be reached from state $X_{i(j-1)}$ with probability A and from state $X_{i,j}$ with probability B, then the new value of $x_{(i-j)j-1}=A\cdot x_{i(j-1)}+B\cdot x_{i,j}$.

	1	2	3	4
1	1	0	0	0
2	1/6	5/6	0	0
3	1/36	15/36	20/36	0
4	1/216	35/216	120/216	60/216

Figure 3—The state-matrix for the case of six sectors and a queue length of four

At this stage, if another request is introduced in the queue, the elements of the ith row can be calculated from the elements of the $(i-1)$th row by a procedure similar to the one used when four random requests are introduced in the queue.

When the device operates with a constant queue length, a new request is added to the queue when an existing request gets satisfied. The device will constantly be moving from the states of the Qth row to the states of the $(Q-1)$th row and back. The move from the Qth row to the $(Q-1)$th row will occur when a request gets satisfied and the move back to the Qth row will occur when the new request joins the queue.

If the probabilities associated with these jumps are known, we can write an equation of the form

$$x_{ij}^*=x_{ij}\cdot C_j+D_j$$

where x_{ij}^* is the value of x_{ij} after one request has been satisfied and another added to the queue. This enables us to define an iterative procedure to calculate the elements of the Qth row after a large number of requests have been satisfied and an equally large number of requests added to the queue. To begin the iterative procedure, we need to know the initial values of the elements of the Qth row and the values of C_j's and D_j's. By performing enough iterations, we can calculate the steady-state values of the elements of the Qth row, if such a state exists.

The procedure mentioned in this paper will be useful if a steady-state exists and if it is independent of the initial conditions. We shall soon see that both these requirements are satisfied.

STEADY-STATE SOLUTION

We shall calculate the steady-state values for the special case of a device with six sectors/track and a constant queue of four members. After that, we shall describe a general procedure for obtaining the steady-state solution.

Let A represent the probability of being in state X_{41}, $B1$ the probability of being in substate $X_{42}{}^1$, $B2$ the probability of being in substate $X_{42}{}^2$, C the probability of being in state X_{43}, and D the probability of being in state X_{44}.

Let the initial values of $A, B1, B2, C$ and D be $\frac{1}{216}$, $\frac{20}{216}$, $\frac{15}{216}$, $\frac{120}{216}$ and $\frac{60}{216}$, respectively.

The next step is to calculate the new values of A, $B1$, $B2$, C and D after one request is satisfied and a new one joins the queue. These new values will be represented here by a superscripted asterisk.

If the device was in state X_{41}, it will necessarily jump to state X_{31} when a request is satisfied.

The probability of jumping back to X_{41} when a new request joins the queue $= \frac{1}{6}$.

The probability of reaching state X_{41} along the path $X_{41} \rightarrow X_{31} \rightarrow X_{41} - A/6$. If the device was in substate $X_{42}{}^1$, the probability that the sector for the group containing three requests will occur before the sector for the group containing one request is $\frac{1}{2}$.

Therefore, the probability that the device will jump to state X_{31} is $\frac{1}{2}$ and the probability that it will jump to state X_{32} is also $\frac{1}{2}$.

Probability of the device reaching state X_{41} along the path $X_{42}{}^1 \rightarrow X_{31} \rightarrow X_{41} = (B1 \times \frac{1}{2} \times \frac{1}{6})$.

$$A^* = (A/6) + (B1 \times \tfrac{1}{2} \times \tfrac{1}{6}).$$

Substate $X_{42}{}^1$ can be reached along five different paths. These paths and the probabilities associated with them are:

1.	$X_{41} \rightarrow X_{31} \rightarrow X_{42}{}^1$	$(A \times \frac{5}{6})$
2.	$X_{42}{}^1 \rightarrow X_{31} \rightarrow X_{42}{}^1$	$(B1 \times \frac{1}{2} \times \frac{5}{6})$
3.	$X_{42}{}^1 \rightarrow X_{32} \rightarrow X_{42}{}^1$	$(B1 \times \frac{1}{2} \times \frac{1}{6})$
4.	$X_{42}{}^2 \rightarrow X_{32} \rightarrow X_{42}{}^1$	$(B2 \times \frac{1}{6})$
5.	$X_{43} \rightarrow X_{32} \rightarrow X_{42}{}^1$	$(C \times \frac{2}{3} \times \frac{1}{6})$

Path 4 has probability $B2 \times \frac{1}{6}$ because if the device was in state $X_{42}{}^1$ and a request is satisfied, it must necessarily jump to state X_{32}. When a new request is introduced, the probability of jumping to substate $X_{42}{}^1$ is $\frac{1}{6}$.

For path 5, the value is $(C \times \frac{2}{3} \times \frac{1}{6})$ because state X_{43} represents a 2-1-1 combination and if a request is satisfied, the probability that the sector had only one request is $\frac{2}{3}$. Therefore, the probability of jumping to state X_{32} is $\frac{2}{3}$ and the probability of jumping to state X_{33} is $\frac{1}{3}$.

$$B1^* = (A \times \tfrac{5}{6}) + (B_1 \times \tfrac{1}{2} \times \tfrac{5}{6}) + (B1 \times \tfrac{1}{2} \times \tfrac{1}{6}) + (B2 \times \tfrac{1}{6}) + (C \times \tfrac{2}{3} \times \tfrac{1}{6}).$$

Substate $X_{42}{}^2$ can be reached by three different paths:

1.	$X_{42}{}^1 \rightarrow X_{32} \rightarrow X_{42}{}^2$	with prob.	$(B1 \times \frac{1}{2} \times \frac{1}{6})$
2.	$X_{42}{}^2 \rightarrow X_{32} \rightarrow X_{42}{}^2$	with prob.	$(B2 \times 1 \times \frac{1}{6})$
3.	$X_{43} \rightarrow X_{32} \rightarrow X_{42}{}^2$	with prob.	$(C \times \frac{2}{3} \times \frac{1}{6})$

$$B^* = (B1 \times \tfrac{1}{2} \times \tfrac{1}{6}) + (B2 \times 1 \times \tfrac{1}{6}) + (C \times \tfrac{2}{3} \times \tfrac{1}{6})$$

The paths to reach state X_{43} are:

1.	$X_{42}{}^1 \rightarrow X_{32} \rightarrow X_{43}$	with prob.	$(B1 \times \frac{1}{2} \times \frac{4}{6})$
2.	$X_{42}{}^2 \rightarrow X_{32} \rightarrow X_{43}$	with prob.	$(B2 \times \frac{4}{6})$
3.	$X_{43} \rightarrow X_{32} \rightarrow X_{43}$	with prob.	$(C \times \frac{2}{3} \times \frac{4}{6})$
4.	$X_{43} \rightarrow X_{33} \rightarrow X_{43}$	with prob.	$(C \times \frac{1}{3} \times \frac{3}{6})$
5.	$X_{44} \rightarrow X_{33} \rightarrow X_{43}$	with prob.	$(D \times 1 \times \frac{1}{2})$

$$C^* = (B1 \times \tfrac{1}{2} \times \tfrac{4}{6}) + (B2 \times \tfrac{4}{6}) + (C \times \tfrac{2}{3} \times \tfrac{4}{6}) + (C \times \tfrac{1}{3} \times \tfrac{3}{6}) + (D \times \tfrac{1}{2})$$

Similarly, state X_{44} can be reached by the following two paths:

$X_{43} \rightarrow X_{33} \rightarrow X_{44}$	with prob.	$(C \times \frac{1}{3} \times \frac{1}{2})$
$X_{44} \rightarrow X_{33} \rightarrow X_{44}$	with prob.	$(D \times \frac{1}{2})$

$$D^* = (C \times \tfrac{1}{3} \times \tfrac{1}{2}) + (D \times \tfrac{1}{2})$$

A^*, $B1^*$, $B2^*$, C^*, D^* represent the new values of A, $B1$, $B2$, C and D after one request has been satisfied and a new one joins the queue.

The second iteration will represent the situation after two requests have been satisfied, the third after three and so on.

A *PL/I* program carried out 100 iterations and it was found that after ten iterations, the change in the values A, B, C and D was only beyond the sixth place of decimal.

To show that the convergence to a steady-state solution is independent of the initial conditions, the program was run with initial condition $A = B1 = B2 = C = 0$, $D = 1$.

In this case, the steady-state was reached after 15 iterations.

The transient values and the duration of the transients is a function of the initial conditions.

The steady-state values were:

$$A=0.017857, \quad B=B1+B2=0.267857,$$

$$C=0.535714, \quad D=0.179571.$$

A general algorithm for obtaining the steady-state solution will now be described.

A GENERAL ALGORITHM FOR DETERMINING STEADY-STATE VALUES

Let there be N sectors/track and Q members in the queue. The state-matrix will contain Q rows. Figure 4 shows the last two rows of the matrix. If N is greater than Q, the matrix contains Q columns. Otherwise, the number of columns in the matrix is equal to N.

For each element x_{Qj} of the Qth row, we want to determine the values of C_j and D_j such that

$$x_{Qj}^{*}=X_{Qj}\cdot C_j+D_j$$

If the device is in state X_{ij}, there are j groups in a queue of i members and each group contains at least one member. Each state X_{ij} is divided into M_{ij} substates where M_{ij} is obtained as follows:

Let $(Y_1, Y_2, \ldots, Y_j)$ be an unordered j-tuple such that

$$\sum_{P=1}^{j} Y_P=i \quad \text{and} \quad Y_P>0$$

Then M_{ij} is the number of distinct j-tuples.

Each substate represents a combination by which the i members distribute themselves among j groups. Since the j-tuple was unordered, groups A, B, C containing a, b, c members respectively are represented by the same substate as groups A, B, C containing b, a, c members respectively.

Let the kth substate of state X_{ij} be represented by X_{ij}^{k}, and the probability of the device existing in substate X_{ij}^{k} be x_{ij}^{k}. The calculation of C_j's and D_j's is done on a substate level such that we are effectively calculating C_j^{k}'s and D_j^{k}'s.

Consider state X_{Qj}. We need to determine C_j and D_j such that

$$x_{Qj}^{*}=x_{Qj}\cdot C_j+D_j$$

However we shall determine all C_j^{k}'s and D_j^{k}'s such that

$$X_{Qj}^{k*}=X_{Qj}^{k}\cdot C_j^{k}+D_j^{k} \quad \text{for all} \quad X_{Qj}^{k}$$

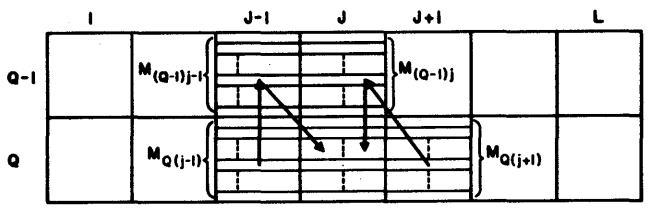


Figure 4—The last two rows of the state-matrix

and

$$X_{Qj}^{*}=\sum_{k=1}^{MQj} X_{Qj}^{k*}$$

To determine all paths by which substate X_{Qj}^{k} can be reached when a request is satisfied and another added to the queue, we shall focus our attention on states $X_{Q(j-1)}$, X_{Qj}, $X_{Q(j+1)}$, $X_{(Q-1)j-1}$ and $X_{(Q-1)j}$. Each substate of states $X_{Q(j-1)}$, X_{Qj}, $X_{Q(j+1)}$ may begin a path leading to substate X_{Qj}^{k}.

We shall first consider the substates of the state $X_{Q(j-1)}$. The device can jump to substates of state $X_{(Q-1)j-1}$ when a request is satisfied.

Let $P_{ijk}{}^{lmn}$ be the probability of jumping from substate X_{ij}^{k} to substate X_{lm}^{n}.

For the Zth substate of $X_{Q(j-1)}$ the probability of reaching substate X_{Qj}^{k} is given by

$$\sum_{Y=1}^{M(Q-1)j-1} P_{Q(j-1)Z}{}^{(Q-1)(j-1)Y}\cdot P_{(Q-1)(j-1)y}{}^{Qjk}\cdot X_{Q(j-1)}{}^{Z}$$

$$=q_{Q(j-1)Z}{}^{Qjk}$$

where $q_{Q(j-1)Z}{}^{Qjk}$ is the total probability of reaching X_{Qj}^{k} from state $X_{Q(j-1)}$. Therefore, the probability of reaching X_{Qj}^{k} from state $X_{Q(j-1)}$

$$=\sum_{Z=1}^{MQ(j-1)} q_{Q(j-1)Z}{}^{Qjk}=\gamma_1$$

The above equation covers all paths from state $X_{Q(j-1)}$ to substate X_{Qj}^{k} since all these paths must go through the state $X_{(Q-1)j-1}$.

Substate X_{Qj}^{k} can also be reached by paths originating from the substates of state X_{Qj}. The first set of paths is via state $X_{(Q-1)(j-1)}$ and the second set is via state $X_{(Q-1)j}$

$$\leftarrow q_{QjZ}{}^{Qjk}=\sum_{Y=1}^{M(Q-1)j-1} P_{QjZ}{}^{(Q-1)(j-1)Y}\cdot P_{(Q-1)(j-1)y}{}^{Qjk}$$

$$\cdot X_{Qj}{}^{Z}+\sum_{Y=1}^{M(Q-1)j} P_{QjZ}{}^{(Q-1)jY}\cdot P_{(Q-1)jy}{}^{Qjk}\cdot X_{Qj}{}^{Z}$$

Therefore, the probability of reaching substate X_{Qj}^{k}

from state X_{Qj}

$$= \sum_{Z=1}^{MQj} q_{QjZ}^{Qjk} = \gamma_2$$

For the paths originating from substate $X_{Q(j+1)}^{Z}$,

$$q_{Q(j+1)Z}^{Qjk} = \sum_{Y=1}^{M(Q-1)j} P_{Q(j+1)Z}^{(Q-1)jY} \cdot P_{(Q-1)jy}^{Qjk} \cdot X_{Q(j+1)}^{Z}$$

Total probability of reaching substate X_{Qj}^{k} from state $X_{Q(j+1)}$

$$= \sum_{Z=1}^{MQ(j+1)} q_{\ Q(j+1)Z}^{Qjk} = \gamma_3$$

Therefore, the total probability of reaching substate X_{Qj}^{k} after one request is satisfied and a new request added to the queue is $x_{Qj}^{k*} = r_1 + r_2 + r_3$.

By putting in the appropriate expression for r_1, r_2 and r_3 we can get x_{Qj}^{*} in the form $x_{Qj}^{k} \cdot C_j^k + D_j^k$. However, it is easier to calculate the r's separately and then add them to obtain x_{Qj}^{*}.

Similarly, expressions are obtained for all the substates in the Qth row. The iterations are performed on the probabilities associated with the substates.

When the steady-state is achieved, the values of x_{Qj}'s are obtained for the different values of j.

$$X_{Qj} = \sum_{k=1}^{MQj} X_{Qj}^{k}$$

$X_{Q1}, X_{Q2}, \ldots, X_{QL}$ are the steady-state values that we need, to calculate the performance of the device. In practice we find that unless the queue length is very large, there are very few substates associated with each state. For a queue length of four, only state X_{42} contained two substates. The substates for the other states were the states themselves. For a queue length of six, states X_{62} and X_{63} will contain three substates each while state X_{64} contains two substates. The rest of the states do not have any substates besides themselves.

It is unlikely that the hardware queuer will handle a very large queue during normal device operation. In any case, a simple program should be able to evaluate the steady-state values.

The next step is to determine the data rate of the device from the steady-state values.

DETERMINATION OF DATA RATE

An analysis is developed to correlate the steady-state solution obtained from the state matrix to the data rate of the device.

Let us consider a device with N sectors and a queue length of Q.

The state matrix provides the steady-state values of $x_{Q1}, x_{Q2}, \ldots, x_{QL}$ where L is equal to Q if the number of sectors is greater than the queue length. Otherwise, L is equal to the number of sectors.

The average number of sectors that the head will skip before it finds one for which a request exists will be determined.

x_{Q1} is the probability that all requests exist for the same sector. When a request is satisfied and a new one joins the queue, the head will wait till the sector for the new request comes up. In the worst case, it will have to wait for $N-1$ sectors to transfer the closest request if the new request occurs for the same sector as the rest of the members of the queue. Since the new request can exist for any sector with equal probability, the average number of sectors skipped before reading/writing occurs is $(N-1)/2$.

Consider the significance of element x_{Q2} of the state-matrix. If the two sectors for which the requests exist are S_1 and S_2 and a request for S_1 is satisfied, the head will either wait for sector S_2 or the sector S_{NEW} for which the new request exists. The decision will depend on the distance of S_2 and S_{NEW} from S_1 in the direction of rotation. The sector closest to S_1 is satisfied first.

For the sector (S_1+1) which is the sector immediately after S_1, the interval between the present position of the head and the last time it passed this sector is $(N-1)$ sectors. For sector (S_1+2), this interval is $(N-2)$ sectors and so on.

Since (S_1+1) has been exposed to new requests for the longest period, it has the highest probability of being the second distinct sector number. This means that the probability of sector (S_1+1) being S_2 is greater than the probability of (S_1+2) being S_2. Similarly, the probability of (S_1+2) being S_2 is greater than the probability of (S_1+3) being S_2.

By the above approximation in which the probability of a sector (S_1+X) being S_2 is directly proportional to $(N-X)$:

$$\frac{\text{prob } (S_1+X_1) \text{ being } S_2}{\text{prob } (S_1+X_2) \text{ being } S_2} = \frac{N-X_1}{N-X_2}$$

Therefore, probabilities of sectors (S_1+1), (S_1+2), $\ldots (S_1-1)$ being S_2 are in the ratio $N-1 : N-2 : N-3 \ldots : 1$

If $\text{SUM} = (N-1)/(Y-1)Y$, then the probability that sector (S_1+1) is the second distinct sector $= (N-1)/\text{SUM}$. Also, the probability that sector (S_1+2) is the second distinct sector $= (N-2)/\text{SUM}$ and so on.

Therefore, the representative request for the second distinct sector is 'biased' towards the sectors close to

S_1. When the new random request joins the queue, the situation is such that a 'random' and a 'biased' request are in contention to be satisfied first.

Extending the argument further, there are, two 'biased' requests and one random request when a request is satisfied in state X_{Q3} and a new one joins the queue.

In general, the device in state X_{QK} has $(K-1)$ 'biased' requests and one random request in contention when it satisfies a request and a new one joins the queue. The head will not have to skip any sectors before reading/writing if any one of these requests exists for the sector immediately following the one for which the request was satisfied.

Probability that there is no request for the next sector

$$=\left(\frac{\text{SUM}-(N-1)}{\text{SUM}}\right)\left(\frac{N-1}{N}\right)=P_0'$$

Probability the head skips zero sectors $=P_0=1-P_0'$. Let S be the sector which contained data for the request satisfied.

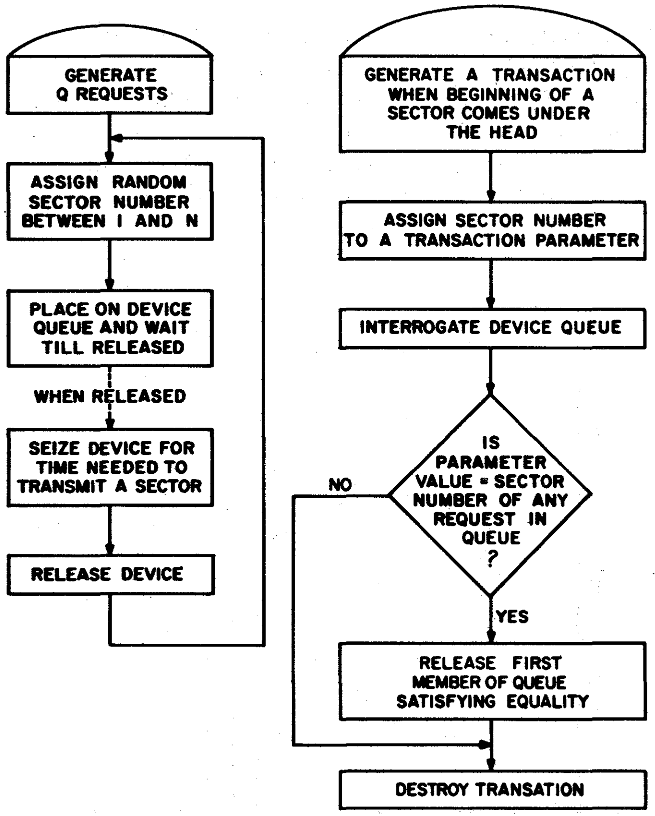

GENERATE Q REQUESTS
ASSIGN RANDOM SECTOR NUMBER BETWEEN I AND N
PLACE ON DEVICE QUEUE AND WAIT TILL RELEASED
WHEN RELEASED
SEIZE DEVICE FOR TIME NEEDED TO TRANSMIT A SECTOR
RELEASE DEVICE
GENERATE A TRANSACTION WHEN BEGINNING OF A SECTOR COMES UNDER THE HEAD
ASSIGN SECTOR NUMBER TO A TRANSACTION PARAMETER
INTERROGATE DEVICE QUEUE
IS PARAMETER VALUE = SECTOR NUMBER OF ANY REQUEST IN QUEUE ?
NO
YES
RELEASE FIRST MEMBER OF QUEUE SATISFYING EQUALITY
DESTROY TRANSATION

Figure 5—Simplified block diagram of the simulation program in GPSS/360

The head will skip one sector if no request exists for $(S+1)$ and a request exists for $(S+2)$.

Given no request for $(S+1)$, probability that there is no request for $S+2$

$$=\left(\frac{\text{SUM}-(N-1)-(N-2)}{\text{SUM}-(N-1)}\right)\left(\frac{N-2}{N-3}\right)=P_1'.$$

Given no request for $(S+1)$, the probability of at least one request for $S+2$

$$=1-P_1'$$

Probability that the head skips one sector to satisfy the next request

$$=P_0'(1-P_1')=P_1$$

Similarly, the probability of skipping M sectors is P_M and

$$P_M=P_0'P_1'\cdots P_{M-1}'(1-P_M')$$

If the device was in state X_{QU} when it satisfied a request, there are $(U-1)$ biased representative requests and one random request in contention. In this situation, the average number of sectors skipped

$$=\sum_{i=o}^{(N-1)-(U-1)} i\cdot P_i=A_U \qquad \ldots\ldots (1)$$

The upper limit of i in the summation is equal to $(N-1)-(U-1)$ because there are $(U-1)$ distinct sectors that have representative requests associated with them. In this case, the maximum number of sectors that may be skipped is equal to $(N-1)-(U-1)$

The average number of sectors skipped A_{SKIP} and

$$A_{\text{SKIP}}=\sum_{U=1}^{Q} x_{QU}\cdot A_U \quad \text{if} \quad N>Q \text{ and}$$

$$=\sum_{U=1}^{N} x_{QU}\cdot A_U \quad \text{if} \quad N\leq Q.$$

In the above formula, the x_{QU}'s are obtained from the state-matrix X and the A_U's are calculated from (1).

Data Rate of the device $=D_{\max}/(A_{\text{SKIP}}+1)$ where $D_{\max}$ is the maximum data rate of the device or the rate at which it transmits data while reading or writing records.

Another way of expressing performance of the device is in terms of percent utilization.

$$\text{Percent Utilization}=\frac{100}{A_{\text{SKIP}}+1}=\frac{\text{Data Rate. }(100)}{D_{\max}}$$

QUEUE LENGTH	SECTORS = 7		SECTORS = 10		SECTORS = 20	
	BY ANALYSIS	BY SIMULATION	BY ANALYSIS	BY SIMULATION	BY ANALYSIS	BY SIMULATION
2	41.1	41.0	31.4	30.5	17.6	17.2
3	50.9	50.1	40.6	39.7	24.3	24.0
4	58.8	57.2	47.8	46.7	30.5	29.5
5	62.6	63.6	52.7	52.3	34.7	34.4
6	66.1	66.7	56.9	56.7	38.8	38.7

Figure 6—Table of device utilization for various combinations of sector numbers and queue length

PROGRAM FOR DATA RATE EVALUATION

An experimental program was written in *PL*/1 for the evaluation of data rates. The program performs a hundred iterations to obtain a steady-state solution and the program is valid only for the case of a new request joining the queue when another request is satisfied so that the queue length remains constant.

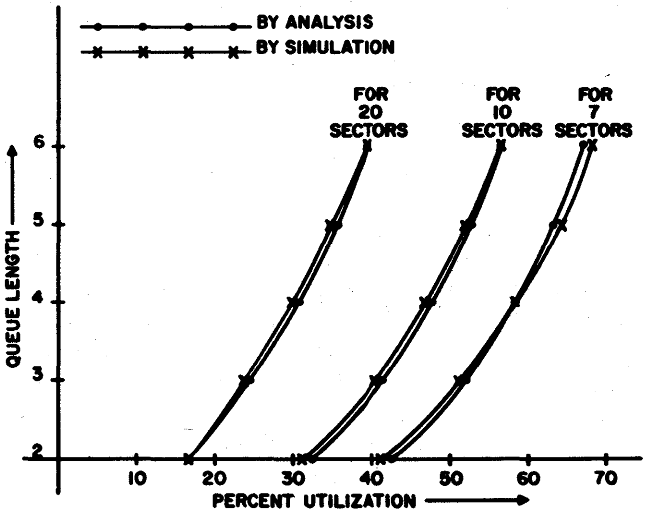


Figure 7—Comparison of results obtained by analysis and those obtained by simulation

SIMULATION

The operation of the device and its queue was simulated in GPSS/360 for various combinations of queue length and numbers of sectors. The random assignment of sector numbers was done by the use of the random number generator of GPSS/360. Figure 5 shows a block diagram of the simulation program.

RESULTS

The results obtained by analysis and by simulation for the percent utilization of the device are shown in the form of a table in Figure 6, and in the form of a graph in Figure 7.

Avg. Data Rate

$$= \frac{\%\ \text{Utilization}}{(100)\times(\text{Max. Data Rate of the Device})}$$

$$= \frac{\%\ \text{Util.}}{100\cdot D_{\max}}$$

DISCUSSION

The results obtained from the program can be used very effectively for data-rate evaluation even when the queue does not remain constant over a long period. It was mentioned that the steady-state solution is reached very rapidly, especially if the initial conditions are provided by the arrival of a cluster of random requests.

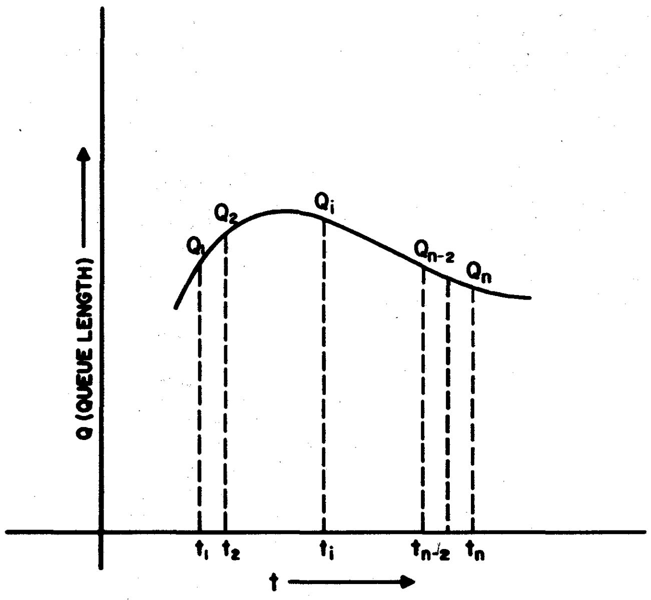


Figure 8—Variation of queue length with time

Since there is a rapid approach to the steady-state, the data rate for any queue pattern can be determined by considering the data rates at various sampled values of the queue length, as shown in Figure 8.

Let D_{R_i} be the data rate for queue length Q_i. If $D_{Ri}=f(Q_i)$ is given by the program, then, for the example considered above, a good approximation for the data rate can be obtained by applying the relation

$$\text{Data Rate} = \sum_{t=1}^{n} D_{R_t} \Big| n$$

If the sampling interval is not constant, a proper weighting factor will have to be used.

The concept of an average queue and its relationship to data rate has significance, only if the function f which relates $D_R=f(Q)$ is linear.

Since this is not the case except in distinct regions of the curve, the relationship between average queue and data rate can be very misleading.

CONCLUSIONS

The state-matrix method provides an easy and logical approach to the problem of performance evaluation. The only assumption made while arriving at the steady-state solution is that the requests are random when they join the queue. Certain added approximations were made for the correlation between the steady-state solution and the device data-rate in order to simplify the analysis. The very close correlation between the results obtained by the state-matrix method and those obtained by simulation provides substantial justification for these approximations.

This method also provides insight into the transient behavior of the queue and its dependence on initial conditions.

REFERENCES

1 W FELLER
An introduction to probability theory and its applications
John Wiley and Sons Inc Third edition 1968 New York New York

2 J ABATE H DUBNER
Optimizing the performance of a drum-like storage
IEEE Transactions on Computers Vol C-18 No 9 pp 992-997 November 1969

Drum queueing model

by S. R. ARORA

University of Minnesota and Univac Division, Sperry Rand Corporation
Minneapolis, Minnesota

and

G. P. JAIN

Univac Division, Sperry Rand Corporation
Minneapolis, Minnesota

INTRODUCTION

This paper deals with the analysis of queues at drums in a computer system. For this analysis the computer system may be viewed as being composed of two units; (1) central processing unit and (2) the auxiliary storage devices (drum subsystem). Requests for the drum subsystem originate from the central processing unit (CPU). In a multiprogramming environment, a number of jobs are concurrently active in the system. Each job may be either waiting or being serviced by one of these units. I/O requests generated by CPU may compete for the services provided by a single drum subsystem. Concurrent requests by the CPU cause queues in front of a drum subsystem. Queuing conflicts cause delays in servicing a request and reduce maximum throughput capability of the system. These conflicts may be reduced by applying one or more of the following techniques.

1. Increasing the number of channels to a subsystem.
2. Varying the number of devices to a subsystem.
3. Increasing the speed of a subsystem.
4. Better file organization.
   etc.

The purpose of this study is to develop a model for obtaining expressions for average queue size and average waiting time for various request rates and subsystem configurations. This problem is of great interest to the system designers. The proposed model will aid in determining the number of channels and also the number of drums within a drum subsystem of an otpimally balanced system.

MATH. MODEL

The model considers a drum subsystem with n drums, equipped with m transfer channels (Figure 1). Requests for the drum subsystem originate from the central processing unit. A request is always for a particular drum. The servicing of a request requires the availability of both the specific drum needed by the request and any one of the transfer channels. The following assumptions are made in this model.

ASSUMPTIONS

1. Requests follow a uniform distribution over the n drum units.
2. Both the drum and the channel are considered busy during the service time of the request.
3. Requests on any drum are serviced on a first-come-first-served basis.
4. Arrivals of requests follow a Poisson Process with a rate λ_K, where index K represents the state of the system. The state of the system is defined by the number of outstanding requests present in the drum system.
5. Service time of any request follows a negative exponential distribution with an average service time of $1/\mu$.

In order to determine the average queue size and the average waiting time, we need to know the stationary distribution of the number of pending requests in the drum system. Let p_K denote the stationary probability of having K requests in the system. From standard results on queuing with Poisson arrivals at rate λ_K and

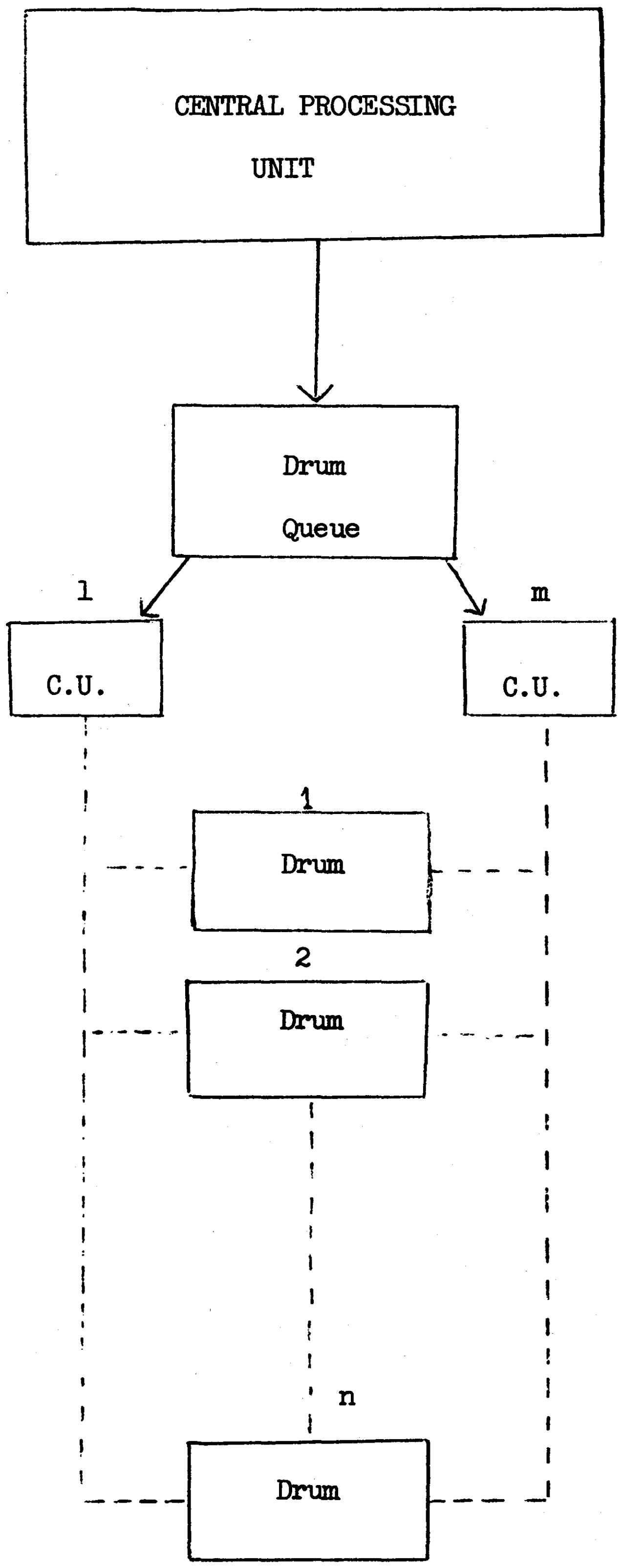


Figure 1

negative exponential service times at rate μ_K, corresponding to the state K, the expression* for p_K is given by

$$p_K = \frac{\lambda_0}{\mu_1} \times \frac{\lambda_1}{\mu_2} \dots \times \frac{\lambda_{K-1}}{\mu_K} p_0 \tag{1}$$

Different arrival patterns can be interesting. Two arrival patterns are considered in this paper. The first arrival pattern considered is given by

$$\lambda_K = \begin{cases} \lambda \text{ for } K < K_{\max}. \\ 0 \quad K \geq K_{\max}. \end{cases} \tag{2}$$

where $K_{\max}$ is a specified constant. $K_{\max}$ represents the number of concurrently active jobs in the system. When all the jobs are held over for I/O the CPU is idle and stops generating further requests. However, if the pending jobs for I/O are less than $K_{\max}$, the CPU continues to generate further requests at a rate of λ.

In order to compute the values of p_K from equation (1), we need to know λ_K and μ_K. Equation (2) specifies λ_K. We now need to develop an expression for the service rate μ_K.

μ_K is dependent upon K, m and n. The maximum number of requests that can be serviced is given by the number of requests for distinctive drums within this set. The number of channels further limit the maximum number or requests that can be serviced at any time. The minimum of the distinctive requests and number of channels will determine the number of requests that can be serviced at any time. Let $G(n, K)$ denote the number of distinct drums needed for K requests. The range of $G(n, K)$ is from 1 to minimum (n, K).

The problem of finding the distribution** of $G(n, K)$ is very similar to the well known problem of distributing K balls randomly in n boxes and finding the distribution for the number of boxes which are occupied by one or more balls. Distribution of $G(n, K)$ is given by

$$P[G(n, K) = g] = \binom{n}{n-g} \sum_{J=0}^{g} (-1)^J \binom{g}{j} \left(1 - \frac{n-g+j}{n}\right)^K \tag{3}$$

As pointed out earlier, the number of requests that can be serviced at any given time is also restricted by m, the number of channels. For $G(n, K)$ equal to g, the minimum of m and g will give the number of requests

* Elements of Queuing Theory with Applications by T. Satty, p-87, McGraw-Hill, 1961.

** An Introduction to Probability Theory and Its Applications, William Feller, p.-92, John Wiley & Sons, Inc., 1964.

that can be serviced. Define $S(m, n/K)$ as the minimum value of m and g.

$$S(m, n/K) = \min(m, g)$$

$S(m, n/K)$ represents the number of requests that can be serviced simultaneously, when K requests are in the system. Since $G(n, K)$ is a random variable, $S(m, n/K)$ is also a random variable. For abbreviation, we will denote $S(m, n/K)$ by S_K. Probability of a request being completed from the drum system during time t and $t+\Delta t$, when there are K units in the system is given by $S_K\mu\Delta t$. Probability of completing more than one service during time t and $t+\Delta t$ is a function involving terms of second and higher order of Δt, which is represented in the conventional notation of $0(\Delta t)$.

There can be K units in the system at time $t+\Delta t$ in any of the following ways.

(i) there were K units in the system at time t and no arrival or departure took place during time t and $t+\Delta t$.
(ii) there were $K-1$ units at time t and one arrival and no departure took place during time t and $t+\Delta t$
(iii) there were $K+1$ units at time t and no arrival and one departure took place during time t and $t+\Delta t$

All other cases involve more than one transition, which involves second and higher order terms of Δt.

$$\begin{aligned} p_K(t+\Delta t) = {} & p_K(t) \times \text{Prob [no arrival, no departure during } (t, t+\Delta t)] \\ & + p_{K-1}(t) \times \text{Prob [one arrival, no departure during } (t, t+\Delta t)] \\ & + p_{K+1} \times \text{Prob [no arrival, one departure during } (t, t+\Delta t)] \\ & + 0(\Delta t) \end{aligned}$$

The probability of one arrival during $(t, t+\Delta t)$ is $\lambda\Delta t$ and the probability of no arrival during $(t, t+\Delta t)$ is $(1-\lambda\Delta t)$. The probability of one departure during $(t, t+\Delta t)$ is a random variable and is given by $S_K\mu\Delta t$. The probability of no departure during $(t, t+\Delta t)$ is $(1-S_K\mu\Delta t)$.

Kolmogorov equations linking the state probabilities for times t and $t+\Delta t$ are given by

$$\begin{aligned} p_K(t+\Delta t) = {} & \sum_{S_K} p_K(t)(1-\lambda\Delta t)(1-S_K\mu\Delta t)\phi(S_K) \\ & + p_{K-1}(t)\lambda\Delta t \\ & + \sum_{S_K} p_{K+1}(t) S_{K+1}\mu\Delta t \phi(S_K) + 0(\Delta t) \end{aligned}$$

$$\text{for } K \geq 1 \quad (4)$$

and

$$p_0(t+\Delta t) = p_0(t)(1-\lambda\Delta t) + p_1(t) S_1\mu\Delta t + 0(\Delta t)$$

$$\text{for } K=0$$

where $\phi(S_K)$ denotes the density function of S_K and S_{K+1}.

The stationary probabilities are obtained from equation (4) by transposing $p_K(t)$ to the left hand side, dividing throughout by Δt, taking the limit $\Delta t \to 0$ and equating the first derivative of $p_K(t)$ to zero. Equation (4) reduces to

$$\{\lambda + Z(m, n, K)\mu\} p_K = \lambda p_{K-1} + Z(m, n, K+1)\mu p_{K+1} \qquad \text{for } K \geq 1$$

and

$$\lambda p_0 = Z(m, n, 1)\mu p_1 \qquad \text{for } K=0 \quad (5)$$

where $Z(m, n, K)$ denotes the expected value of $S(m, n/K)$

$$Z(m, n, K) = \underset{g}{\mathrm{E}}\, S(m, n/K) \quad (6)$$

$Z(m, n/K)$ represents the average number of requests that can be serviced when there are K requests in the system. Solving equation (5) recursively we obtain

$$p_K(m, n) = \left(\frac{\lambda}{\mu}\right)^K \cdot \frac{p_0(m, n)}{\prod_{j=1}^{K} Z(m, n, j)} \quad (7)$$

$$= \frac{\rho^K p_0(m, n)}{y_K} \quad (8)$$

where

$$\frac{\lambda}{\mu} = \rho$$

and

$$\prod_{j=1}^{K} Z(m, n, j) = y_K$$

$p_K(m, n)$ denotes the stationary probability of having K requests in the system.

The expected service rate when there are K requests in the system is given by $Z(m, n, K)\mu$, which is identical to the service rate μ_K in equation (1).
Since

$$p_0(m, n) + p_1(m, n) + \dots + p_{K\max}(m, n) = 1 \quad (9)$$

We obtain by substituting (8) in (9)

$$p_0(m, n)\left[1 + \frac{\rho}{y_1} + \cdots + \frac{\rho K_{\max}}{y_{K\max}}\right] = 1 \quad (10)$$

or

$$p_0(m, n) = \frac{1}{1+\frac{\rho}{y_1}+\cdots+\frac{\rho K_{\max}}{y_{K_{\max}}}} \tag{11}$$

Substituting (11) in (8) we obtain

$$p_K(m, n) = \frac{\frac{\rho^K}{y_K}}{1+\frac{\rho}{y_1}+\cdots+\frac{\rho K_{\max}}{y_{K_{\max}}}} \tag{12}$$

This gives the distribution of the number of requests in the system. The drum system becomes a bottleneck when the number of requests in the system reach $K_{\max}$, as this is the time, according to our assumptions, when the CPU stops sending any further requests until the time when the state of the system falls below $K_{\max}$. The quantity $p_{K\max}(m, n)$ will, consequently, give the probability with which the drum system becomes a bottleneck.

AVERAGE QUEUE SIZE

Expression for the average queue size of waiting requests is the difference between the expected number of requests in the system and the number of requests receiving service.

$$\text{Average queue size} = \sum_{k=0}^{K\max} Kp_K(m, n) - \sum_{k=0}^{K\max} Z(m, n, K)p_K(m, n) \tag{13}$$

AVERAGE WAITING TIME

In order to find the average waiting time we have to consider the average time required to clear the number of requests on each drum individually.

Consider the Kth request in the system. Let this request be for a drum which has $(r-1)$ requests already ahead of this request. In order to find the average waiting time of a new request we have to find the time it takes to clear $(r-1)$ requests which are ahead of this request.

With the new request the total number of requests in the system are K and the number of requests for one particular drum are r. The probability of having r requests for a particular drum given that there are K requests in the system follow a binomial distribution. Let this probability be denoted by $q(r/K)$, where

$$q(r/K) = \binom{K}{r}\left(\frac{1}{n}\right)^r\left(\frac{n-1}{n}\right)^{K-r} \tag{14}$$

From equation (6) we know $z(m, n/K)$ represents the average number of requests that can be serviced when there are K requests for n drums in the system. Therefore $[z(m, n/K)]/n$ gives the average number of requests that can be serviced from an individual drum in $1/\mu$ time.

The time to service r requests will be $n/[z(m, n/K)]$ (r/μ).

The average waiting time (AWT) of the Kth request will be given by (the service time of r requests—service time of rth request for this drum).

Thus

$$\text{AWT} = \sum_{K=0}^{K\max}\left[\sum_{r=0}^{K} \frac{n}{z(m, n/K)}\frac{r}{\mu} q(r/K)p_K(m, n)\right] - \frac{1}{\mu}(1-p_0(m, n)) \tag{15}$$

Another arrival pattern of interest is where the request rate λ_K gradually goes down as the queue gets larger.

$$\lambda_K \begin{cases} \lambda & \text{for } K=0 \\ \lambda/K^\alpha & \text{for } 0<K<K_{\max} \\ 0 & K \geq K_{\max} \end{cases}$$

The probabilities $p_K(n, m)$ for this is given by

$$p_K(n, m) = \left(\frac{\lambda}{\mu}\right)^K \frac{p_0(m, n)}{\prod_{j=1}^{K} [Z(m, n, j)]j^\alpha} = \frac{\rho^K p_0(m, n)}{y_K'}$$

where

$$\frac{\lambda}{\mu} = \rho \text{ and } \prod_{j=1}^{K} Z(m, n, j)j^\alpha = y_K' \tag{8a}$$

By substituting (8a) in (9) we obtain

$$p_0(m, n)\left[1+\frac{\rho}{y'_1}+\cdots+\frac{\rho K_{\max}}{y'_{K\max}}\right] = 1$$

$$p_0(m, n) = \frac{1}{1+\frac{\rho}{y'_1}+\cdots+\frac{\rho K_{\max}}{y'_{K\max}}} \tag{11a}$$

Substituting (11a) in (8a) we obtain

$$p_K(m, n) = \frac{\frac{\rho_K}{y'_K}}{1+\frac{\rho}{y'_1}+\cdots+\frac{\rho K_{\max}}{y'_{K\max}}} \tag{12a}$$

TABLE I

No. of drums n	λ/m $=\rho$	m=1		m=2		m=4		λ/m $=\rho$	m=1		m=2		m=4	
		AQS	AWT	AQS	AWT	AQS	AWT		AQS	AWT	AQS	AWT	AQS	AWT
2	.3	.13	11.7	.03	2.0	—	—	1.5	17.00	1545.8	2.46	116.0	—	—
4		.13	11.7	.02	.9	.01	.7		17.00	1545.8	2.06	94.0	.46	15.9
6		.13	11.7	.01	.7	.01	.4		17.00	1545.8	1.98	90.9	.27	8.8
8		.13	11.7	.01	.5	.006	.3		17.00	1545.8	1.95	88.6	.20	6.3
10		.13	11.7	.01	.3	.004	.2		17.00	1545.8	1.93	87.5	.16	4.9
2	.5	.50	45.4	.11	5.9	—	—	1.7	17.6	1597	4.36	201.5	—	—
4		.50	45.4	.06	3.0	.04	1.8		17.6	1597	3.88	176.5	.62	20.7
6		.50	45.4	.05	2.3	.02	1.1		17.6	1597	3.80	172.2	.37	11.4
8		.50	45.4	.04	2.1	.02	.8		17.6	1597	3.76	171.1	.28	8.2
10		.50	45.4	.04	2.0	.01	.6		17.6	1597	3.74	170.5	.23	6.4
2	.7	1.62	147.4	.24	12.5	—	—	1.9	17.9	1626	7.24	331.0	—	—
4		1.62	147.4	.15	7.0	.08	3.4		17.9	1626	6.80	309.0	.83	26.5
6		1.62	147.4	.13	5.9	.05	2.0		17.9	1626	6.73	306.0	.50	14.7
8		1.62	147.4	.12	5.5	.04	1.4		17.9	1626	6.70	305.0	.38	10.5
10		1.62	147.4	.12	5.2	.03	1.1		17.9	1626	6.68	304.5	.32	8.8
2	.9	5.53	502.9	.45	23.1	—	—	2.1			10.3	471.6	—	—
4		5.53	502.9	.31	14.4	.14	5.7				10.1	460.2	1.09	33.2
6		5.53	502.9	.28	12.7	.08	3.2				10.0	458.0	.66	18.6
8		5.53	502.9	.26	12.0	.06	2.3				10.0	457.2	.52	13.7
10		5.53	502.9	.25	11.7	.05	1.7				10.0	456.8	.44	11.0
2	1.1	12.3	1117.9	.81	39.8	—	—	3.5			16.67	757	—	—
4		12.3	1117.9	.59	27.2	.22	8.4				16.67	757	6.00	145
6		12.3	1117.9	.54	24.9	.13	4.7				16.67	757	4.59	107
8		12.3	1117.9	.52	23.9	.10	3.6				16.67	757	4.23	96
10		12.3	1117.9	.51	23.2	.08	2.6				16.67	757	4.06	93
2	1.3	15.75	1433.0	1.4	67.3	—	—	4.1			17.05	774		
4		15.75	1433.0	1.1	50.3	.32	11.9				17.05	774	9.5	221
6		15.75	1433.0	1.0	47.2	.19	6.6				17.05	774	8.4	192
8		15.75	1433.0	1.0	46.0	.14	5.2				17.05	774	8.2	187
10		15.75	1433.0	1.0	45.0	.11	3.4				17.05	774	8.1	184

Formulas for Average Queue Size and Average Waiting Time are given by equations (13) and (15) respectively.

ANALYSIS

Theoretically, if the requests are generated faster than these can be serviced the queue may approach infinity but in a computer system the number of requests that can be made to the drum subsystem are restricted. The number of requests are limited by the number of active jobs and the amount of parallel I/O activity within each job. For this hypothetical example it is assumed that $\mu = .011$ and the maximum number of requests (K) in the system are 20, i.e., for $K = 20$ the system stops generating any more requests. It is further assumed that the system is generating requests at the rate of λ.

Table I presents the mean queue distribution and waiting time distribution for various combinations of subsystem request rates, and subsystem configurations when the CPU request rates remain independent of queue length.

Some of these results are presented in Figures 2, 3 and 4.

One can observe from Figure 1 that in a single channel system the queue size and the waiting time are not affected by change in the number of drums on a subsystem. But as ρ approaches 1 the queue size and the waiting time get extremely large.

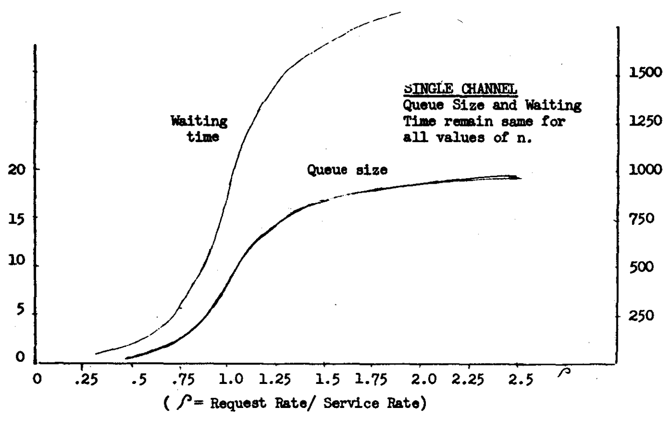


Figure 2

Addition of drums to a dual channel system reduces both the queue size and the waiting time but for more than four drums to a system the queue size and waiting time do not change significantly. (See Figures 2 and 3.)

Another observation which can be made from Table I is that the addition of an extra channel to a single channel system doubles the throughput rate. Also, the queue size is slightly decreased.

SUMMARY

Many systems and processes in use today are quite complex. Queuing problems occur from time to time when there are sufficient requests or severe irregularity in the system. Multiprogramming environment may place a heavy load on the auxiliary storage devices and thus cause queues. Therefore it is important to design an auxiliary storage system for the desired response time.

Standard queuing formulas cannot be used to find the queue size and waiting times for a drum system because in a drum system the queue size is restricted and the request rate goes down as the queue gets larger. With these two unique features in mind an analytical queuing model is developed to provide estimates of queue size and waiting times. The output of this model enables the system designer to identify drum system limiting factors and allow the determination of system sensitivities.

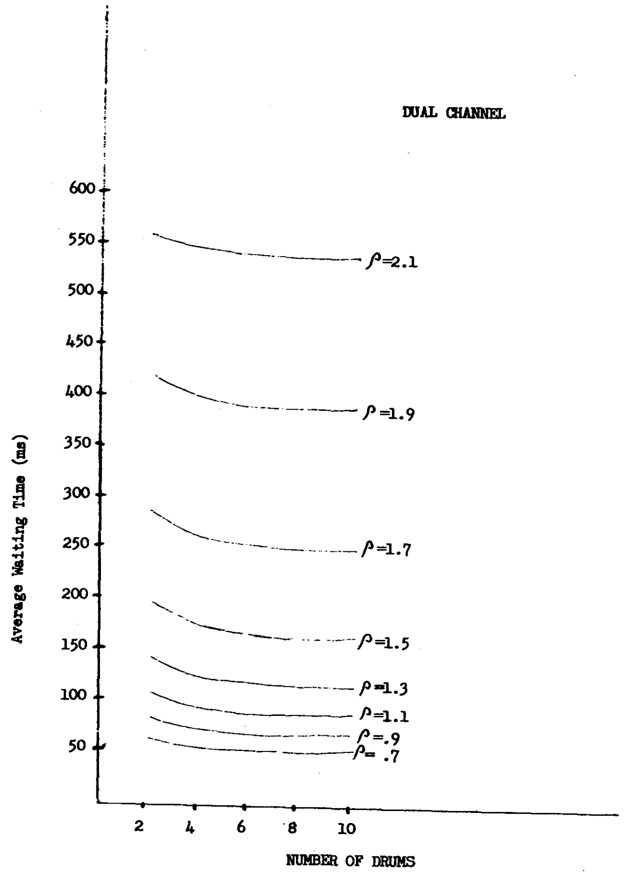


Figure 4

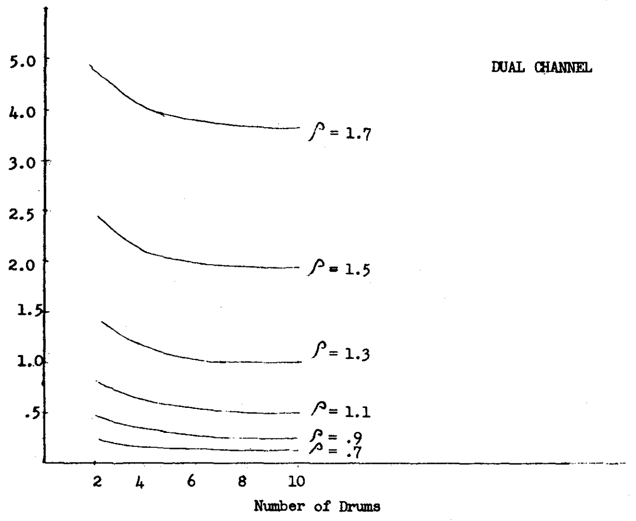


Figure 3

Storage hierarchy systems

by HARRY KATZAN, JR.

Pratt Institute
Brooklyn, New York

INTRODUCTION

The announcement by IBM on June 30, 1970 of System/370 models 155 and 165 with buffer/core storage facilities revived an interest in storage hierarchy systems that was created initially by the System/360 models 85 and 195. However, the notion of a storage hierarchy is not new and has been used previously with regard to storage management by an operating system[1,2] and for efficient use of mass storage devices. In fact, storage hierarchy, in one form or another, has been used in a variety of systems. The purpose of this paper is to survey the various storage hierarchy systems. As such, the paper presents a state-of-the-art analysis of this important area of computer science.

The material is covered from three points of view. The first is *addressable storage* and includes buffer/core systems and large capacity storage (LCS). Also included in the LCS discussion are storage management in an LCS environment and the special case where LCS is not directly addressable and must be accessed through a computer instruction or storage channel. The second topic involves *storage management* and relates to the manner in which addressable storage is managed by an operating system. Covered here are overlay schemes, relocation methods, and virtual memory. The concepts are applied to multiprogramming and time-sharing environments. The final section presents a brief discussion of *data organization and management.* Included here is a discussion of hierarchical data management, data base organization, and the migration of infrequently used information.

STORAGE HIERARCHY CONCEPTS

The purpose of a storage system is to hold information, but the simplicity of the concept ends there. A variety of storage mechanisms exists ranging from high-speed circuitry to low-speed tape or cards. Each type, obviously, has a relative cost/performance index that is used to correlate a storage medium with the function it is to perform. In many cases, it is necessary to synthesize a storage system from two or more types of storage to meet cost objectives and functional requirements. The most simple example of the latter concept is the conventional computing system, depicted in Figure 1, with a relatively small high-speed core storage for programs and data and mass storage consisting of direct-access devices such as magnetic disk or magnetic drum for storing large amounts of information for relatively short periods of time and for use as an overflow device for high-speed core storage. In this case, back-up storage usually consists of tape or cards. The concepts have been extended, obviously, in the high-performance computing systems of today.

The notion of storage hierarchy is important for the following reasons:

1. It has been possible to speed up the functioning of the arithmetic and logical units of a computer to such an extent that performance is dependent upon storage speeds.
2. High speed computer storage is particularly cost sensitive.
3. Programmers are frequently required to prepare programs for computers with storage that is considerably smaller in size than that which is actually needed. Thus, programs must be structured accordingly, a process that limits the functional effectiveness of the programmers and the computing system.
4. Most installations have accumulated and continue to accumulate large amounts of information.

The purpose of a storage hierarchy system is to increase the overall functional effectiveness of a computing system. In this context, functional effectiveness can take

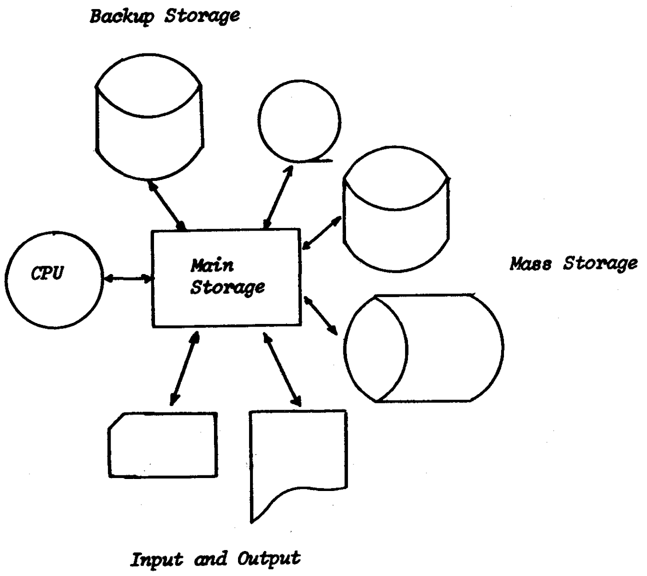


Figure 1—Storage hierarchy of a conventional computing system (simplified)

one of several forms:

1. Increasing the performance of the central processing unit.
2. Increasing total system throughput.
3. Providing the user with a functional capability that is otherwise not inherent in the individual components themselves. (For example, a virtual memory provides a single-level store larger than actual high-speed storage.)
4. Making large amounts of data available within reasonably short periods of time.

In all cases, the objective of a storage hierarchy system is to maximize the frequency that the faster device is referenced so that access speeds approach those of the faster devices at a cost that approaches the slower devices. A storage hierarchy system is usually characterized as follows:

1. Fast and slow storage devices are organized into blocks (of information).
2. The first reference to a particular block requires that the entire block is fetched from the slow device and stored in the fast device. Subsequent fetches to that block require only that fast storage be accessed.
3. The fast storage (usually referred to as a buffer) is capable of storing several blocks which are replaced on a dynamic basis using an appropriate replacement algorithm.

Because storage devices have radically different characteristics and the expected pattern of storage references is difficult to determine beforehand, the methods for designing an optimum hierarchy are not well-defined and often require simulation and empirical analyses. Gecsei, Slutz, and Traiger[3] have described the evaluation of a variety of replacement algorithms for buffer/core storage hierarchy systems. Denning[27] surveys replacement techniques for virtual memory and paging.

ADDRESSABLE STORAGE

Historical perspective

One of the problems in designing a high speed computing system is that core storage is slower than the arithmetic and logical units because of the memory paradox, depicted in Figure 2. Although attempts have been made in the past to compensate for the difference in CPU and storage access time by overlap, local storage buffering, and interleaving, the results were only partially successful and demanded that changes be made to the architecture of the conventional computing system. The first attempt was in the IBM System/360 Model 85 (see Liptay[4]). The Model 85 uses a small fast storage buffer integrated into the CPU called the *cache**. The cache is not addressable by a program and is transparent to the programmer. The cache is used to

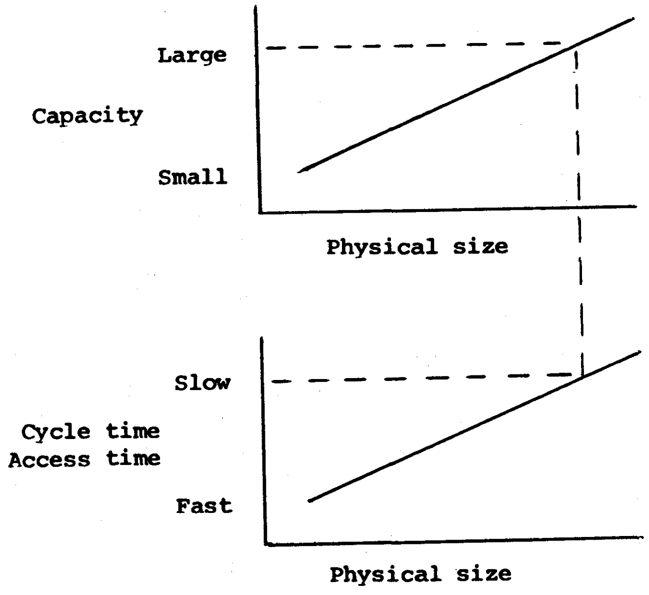


Figure 2—The classical memory paradox

* In computing, a buffer is used frequently to interface two components that operate at different speeds. The most familiar example is the I/O buffer used by many programming systems to compensate for relatively slow I/O speeds.

hold sectors of main storage that are being used and is maintained dynamically by the CPU.

In the Model 85, the cache and main storage are divided into sectors, each containing a fixed number of storage blocks. When a block of storage is referenced for the first time by the CPU, it is moved to the cache. Subsequent references to that block are made to the cache without having to access the slower main storage. Figure 3 depicts a schematic of the Model 85. Detailed information on the Model 85 and the cache organization is available from Liptay[4] and Conti, Gibson, and Pitkowsky.[5] The concept was successful and was used with the System/360 Model 195[6,7] and the System/370 Models 155[8] and 165.[9] The buffer system used with the Model 165 is described in the next section as a concrete example of the organization and maintenance of a buffer/core storage system.

Organization and maintenance of a buffer/core system

The buffer/core storage system in the IBM System/370 Model 165 contains an 8K high-speed buffer storage unit** to increase internal performance. Data flow in the Model 165 is depicted in Figure 4 with a relative access time of 9 to 1 in favor of the buffer over core storage. The CPU can obtain eight bytes from the buffer in 160 nanoseconds compared with a time of 1.44 microseconds when the same amount of data must be obtained from core storage.

When a request for data is made by the CPU, the buffer storage control (Figure 5) searches an address array of the buffer's contents to determine if the data

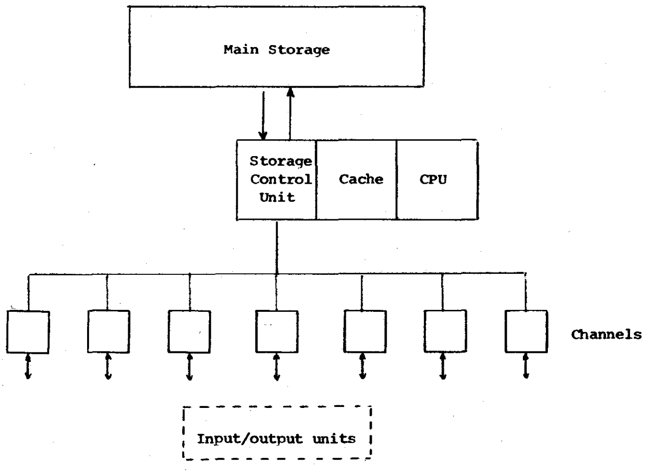


Figure 3—Schematic of the Model 85

** An additional 8K (bytes) of buffer storage can be included with an optional buffer expansion feature.

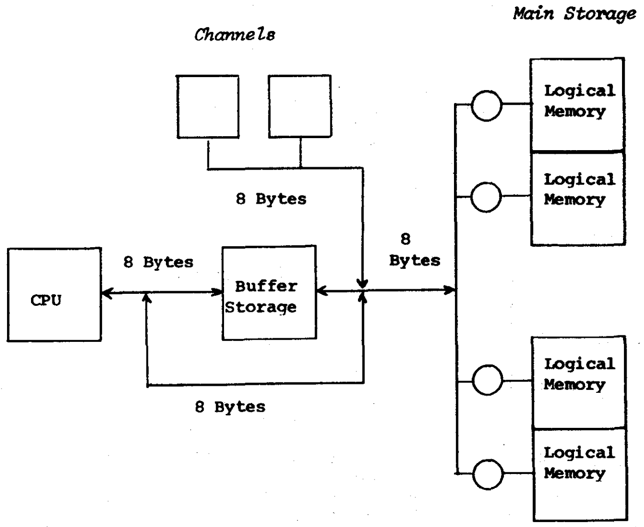


Figure 4—Data flow in the Model 165

requested is in the buffer. If it is, the data is sent to the CPU without a core storage reference. If the requested data is not in the buffer, it is obtained from core storage and sent to the CPU. The data is also assigned a buffer location and stored in the buffer. When data is stored by the CPU, both the buffer and core storage are updated if the corresponding location contains data that is currently in the buffer. The data channels always reference core storage. However, if a data channel stores data in a core storage location that contains data in the buffer, then the buffer is also updated accordingly.

The buffer and core storage are both divided into blocks of 32 bytes, as shown in Figure 6. The 8K buffer contains 64 columns, each containing 4 blocks—a total of 256 distinct blocks of information. (The 16K buffer contains 128 columns and 4 rows for a total of 512 blocks.) Each buffer block contains 32 *consecutive* bytes from core storage, which is also regarded (logically) as containing 64 (or 128) columns with the number of blocks in each column being dependent upon the size of the core storage. Any of the blocks in a core storage column can be placed in one of the four blocks in the corresponding buffer column.

The buffer storage is managed with an address array (Figure 6) and a replacement array, which is described later. An element of the address array consists of the 14 low order bits of the core storage address of the 32 bytes of data contained in the corresponding position in buffer storage. When the CPU references core storage, the address array is used to determine if the requested data is in the buffer.

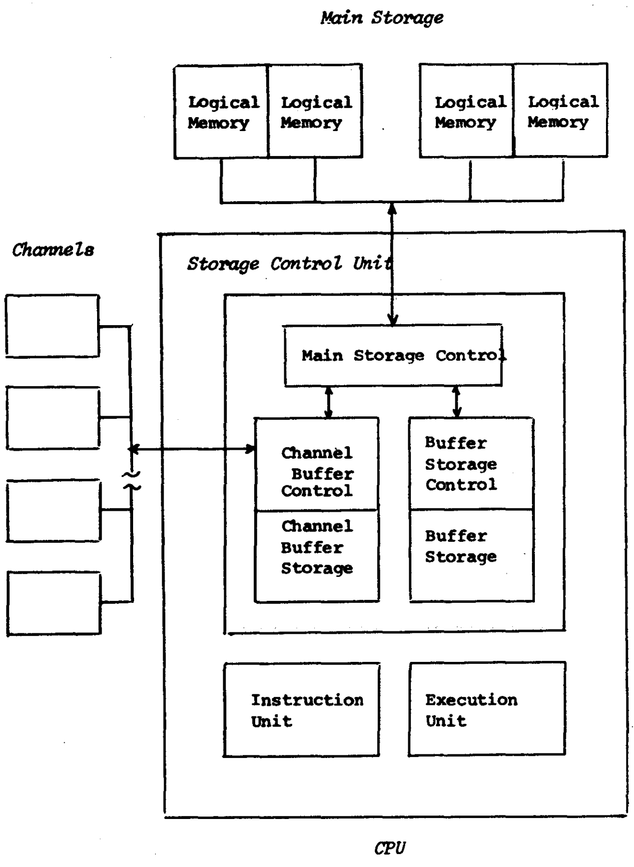


Figure 5—Model 165 storage hierarchy system

The System/370 computer computes a 32-bit effective address* but uses only the low-order 24 bits of it, denoted by bits 8 through 31 in Figure 7. Bits 8 through 21 are used for comparison with address array entries; bits 21-26 (20-26 with a 16K buffer) are used to select a column of the address array; bits 27 and 28 are used to address a double-word (i.e., 8 bytes) within a block; and bits 29-31 are used for displacement within a double word. Thus, an effective address is compared with at most four address array entries; therefore, a column (of 32-byte blocks) of core storage contends for a position in a 4-row column of the buffer.

Each column in the buffer is associated with a 4-entry replacement array which controls buffer replacement activity. When a buffer block is referenced, it is put at the top of the list for its particular column. When a block must be reassigned, the entry at the bottom of the list is selected for replacement.

The buffer storage system operates as follows:

1. When the CPU requests a unit of data, bits 21-26 of the effective address are used to select a (buffer) column of the address array.
2. Bits 8-21 of the effective address are compared with each of the four entries in that column of the address array.
3. If an equal compare is made, a buffer address (Figure 8) is calculated from the address compare and the selected bits of the effective address; the double-word indicated by bits 27 and 28 of the effective address is sent to the CPU. Core storage is not referenced and the referenced buffer block is put at the top of its column activity list in the replacement array.
4. If the block addressed is not found in the corresponding column of the address array, then the data is fetched from core storage, sent to the CPU, and stored in the buffer for future data requests. The buffer block on the bottom of the

* That is, *base* plus *index* plus *displacement.*

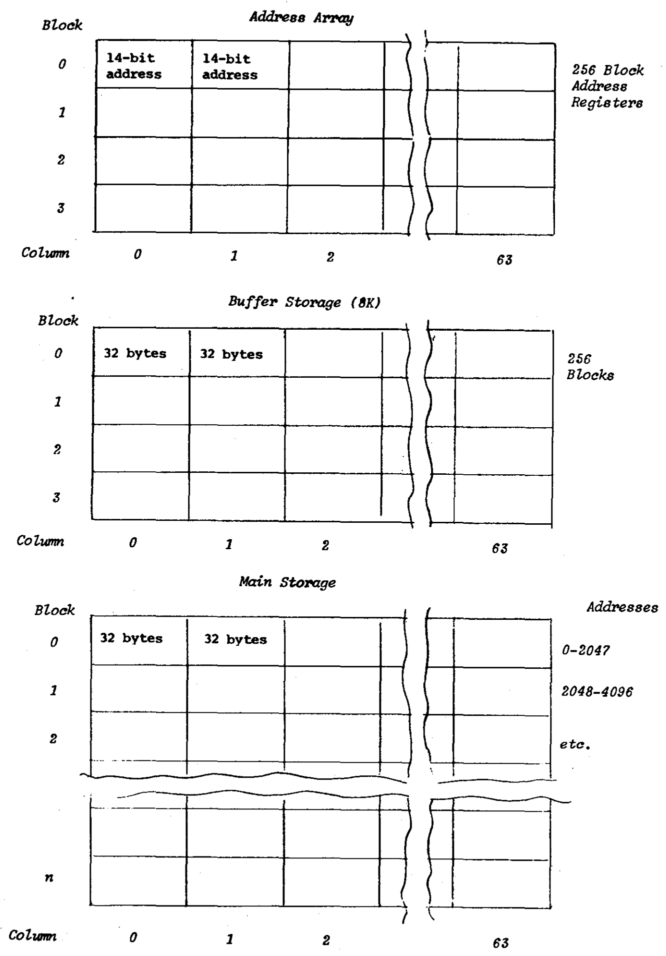


Figure 6—Buffer storage organization (8K)

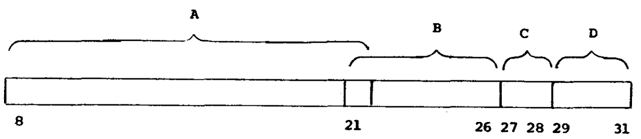
replacement array activity list for that column is assigned the new block of data.

Clearly, the net result of a buffer/core system is to reduce the number of references to core storage making the CPU less dependent upon storage access times. Thus, core storage can be larger even though increased size implies longer cables and a greater access time. Other benefits also exist. The time spent in error checking circuitry and CPU degradation due to core storage cycle stealing from the CPU by high priority data channels become less critical.

Large capacity storage

Large capacity storage* (LCS) is high speed addressable bulk storage. LCS is usually combined with CPU main storage to form a hierarchy of storage directly addressable by the CPU. Addresses in LCS are regarded as an extension to main storage—although the cycle time of LCS is considerably greater. For example, an LCS with a memory cycle time of 8 microseconds is not unusual. However, interleaving is frequently used to reduce the effective cycle time. The CPU can fetch instructions or data directly from LCS, but performance is degraded when executing out of LCS. For some applications, it is efficient to move information from LCS to main storage before it is used. In other cases, the overhead of moving the information is not justified.

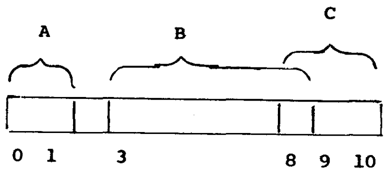
LCS has been used in a variety of interesting ways. Lauer[10] describes the use of LCS at Carnegie-Mellon University as a swapping device, instead of drum, for their 360/67 time-sharing system. The use of LCS was motivated for three reasons: (1) the effective rate at which the system can deliver pages is increased; (2) response-time is decreased since LCS has no rotational delay; and (3) less main core is needed for system operation. For this application, a storage channel, which behaves exactly like an I/O data channel, was used and



A - Address compare
B - Reference buffer column
C - Reference double word
D - Reference byte within double word

Figure 7—Main storage address

* Also known as large core storage.



A - Row in buffer column (results from address compare)
B - Buffer column
C - Reference double word

Figure 8—Buffer storage address

permitted CPU processing and core-to-core transfers to take place simultaneously. Lauer also points out the possibility of executing out of LCS directly by setting page tables to refer directly to LCS.

Freeman[11] relates an equally interesting technique used at the Triangle Universities Computation Center. At TUCC on a 360/75 system, LCS is used for checkpoint/restarts, for a technique called "hyperdisk," and for a variety of other useful functions. Using LCS for checkpoints increased the frequency with which they could be taken and provided added reliability features. Freeman also found that on a 360/75 system, only LCS was a sufficiently fast "source and sink" for system I/O to give satisfactory CPU utilization. Hyperdisk—i.e., a combination of LCS and disk for overflow—effectively allowed system-type programs such as compilers, assemblers, and access routines to reside in core storage instead of being divided into multi-overlay structures.

Fikes, Lauer, and Vareha[12] describe extensions to the Carnegie-Mellon time-sharing system that effectively combine a storage hierarchy of main storage, LCS, disk, and data cell. The C-MU system is further characterized by two levels of executable storage and an increased number of direct data paths from direct-access storage to the CPU. In general, the overhead problem in their system is attacked in the following way. LCS is used as a swapping device and an extension of executable core. Associated with each page of virtual memory is an indicator denoting whether that page is to be accessed by the CPU while residing in main storage or while residing in LCS. Pages that are executed in LCS use a disk as a swapping device and migrate back and forth between LCS and disk. Pages that are executed in main storage used LCS as a swapping device and migrate back and forth between main storage and LCS. The criteria used for determining where a page is executed is relatively simple (partly because of constraints on the implementation effort). All shared pages are executed out of main storage and all non-shared pages are executed out of LCS. In a university environment of small jobs with much assembly and compilation, this effectively amounts to executing system code out of main storage and user code out of LCS. It is also reported that Chen and Hsieh[13] have developed a more optimal algorithm for making the allocation between main storage and LCS.

Not all large capacity storage systems need be directly addressable or accessible by a storage channel. The CDC 7600,[14,15] for example, includes a small core memory (SCM) and a large core memory (LCM) which functions as a storage hierarchy system for programs and data. SCM is used for I/O buffers, for system overlays and tables, and for user programs. LCM is used for permanent system residence, for job swapping, and for input and output files. Only programs in SCM can be executed by the CPU and information is moved between SCM and LCM with block transfer instructions.

Buffer/core summary

Collectively, buffer/core systems and large capacity storage serve the same basic purpose—i.e., to increase the frequency with which accesses are made to the fast storage in a storage hierarchy system. Internal CPU performance is the key issue and in most cases, the hierarchy of storage is transparent to the applications programmer. The methodology seeks to optimize the *overall* functioning of the system, within cost/performance constraints, by applying a functional organization to the hierarchies of storage, taking into consideration the manner in which information is accessed in a general-purpose computing system.

STORAGE MANAGEMENT

The overlay problem

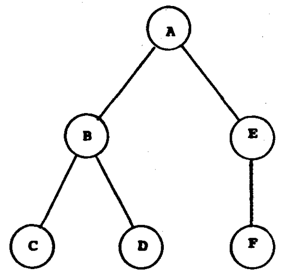
The effective utilization of main storage is important from the standpoint of program development and for computer system performance. Inefficiencies in both categories have characteristically been resolved with the reply, "If we had more core, we could solve the problem." Although the solution is indeed partly correct, equally significant improvements have been made through advanced storage management techniques.

For the programmer, the problem becomes significant when his program is too large for available main storage. The main storage limit may be a physical hardware constraint or may be a logical constraint imposed to increase the level of multiprogramming (see Johnson and Martinson[16]). A program overlay scheme is frequently used (see Lanzano[17]) in which main storage is shared by a hierarchy of program segments. Figure 9 depicts a program tree and a storage trace for a simple overlay.

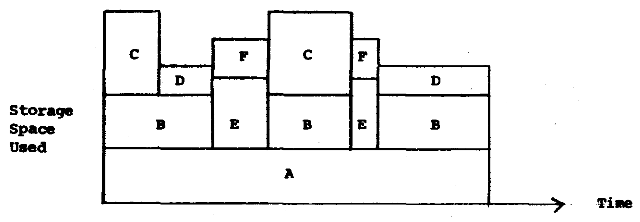
Overlay facilities are perhaps the most elementary form of storage hierarchy for programs and are sometimes used to make addressable storage appear larger than physical or logical storage by storing unused program segments on auxiliary storage such as tape or disk. Several obvious disadvantages exist:

1. Relocation and linking are usually performed before execution is begun.
2. Overlay deck setup requires that the programmer overtly specify the segments that are overlaid.
3. Programs cannot be relocated—as is frequently required in multiprogramming or time-sharing systems.
4. Storage utilization is inefficient in that enough space must be allocated for the largest core load at any point in time (see Figure 9).

Although program overlay techniques are in widespread use today, they severely limit the programmer and the designers of general-purpose operating systems and impose unnecessary throughput limits on computing systems. The reader has probably guessed what is considered to be the state-of-the-art solution to many of the above problems—virtual memory. However, the concept first went through several stages of evolution.



(A) Program tree.



(B) Storage trace.

Figure 9—Simple overlay system

Virtual memory

Because of time-sharing operational considerations, the need for relocatability was as important in the evolution of virtual memory as was the overlay problem. The first evolutionary step was the *relocation register* used by Kinslow[18] at IBM and by Corbato,[19] *et al.*, with the CTSS system at M.I.T. With the relocation register method (Figure 10), the user's program is loaded into main storage and the address at which it is loaded is placed in the relocation register. During execution, the contents of the relocation register are added to each effective address to arrive at the appropriate main storage location. Although use of a relocation register helps solve the relocation problem, it does not solve the overlay problem since the user's program must still reside entirely in main storage.

The next step in the development of a virtual memory system was to consider main storage as a collection of fixed-size pages, such as 4096 bytes as used in TSS/360[20] or 1024 words as used in MULTICS.[21] The user is permitted to construct his program as though he has a large address space; however, each effective address referenced by the system goes through an address mapping, as shown in Figures 11 and 12 (see Randell and Kuehner[22]). In most cases, only high order bits of an address need be translated and addresses within a page can go unchanged. This technique allows pages from different programs to reside in main storage while unused pages are stored on a direct-access device or in large capacity storage. Pages can easily be relocated by adjusting entries in a table of addresses, called a *page table*. The advantages of virtual memory are obvious (see Johnson and Martinson[16]):

1. Adjacent virtual memory pages need not occupy adjacent main storage areas.
2. Not all of a user's program need be in storage simultaneously and can be retrieved when needed (called *demand paging*).
3. The user is permitted a large address space for programs and for data.
4. The operating system can manage storage dynamically.

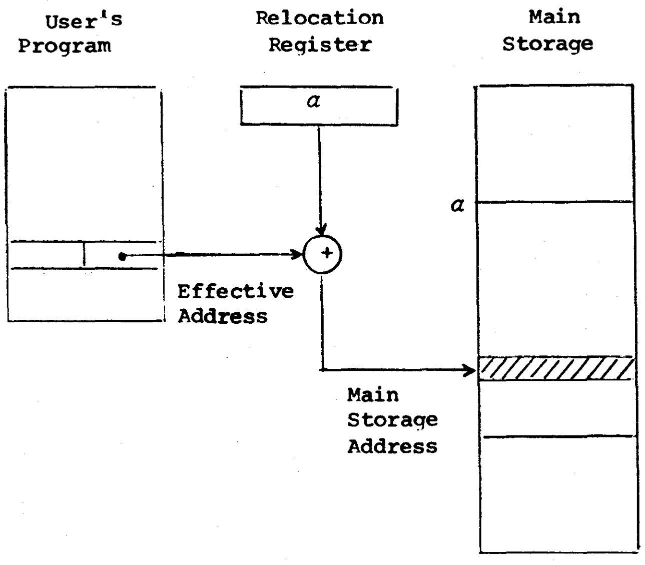


Figure 10—Use of a relocation register

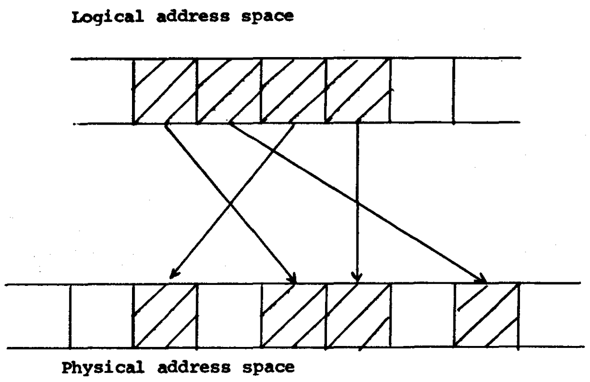


Figure 11—Address mapping

The final step in the evolution of virtual memory achieves efficiency and versatility in storage management by defining two-level page tables and by segmenting a symbolic address into segment number, page number, and displacement within a page (see Dennis[23] and Arden, *et al.*[24]). Two-level page tables (see Figures 13 and 14) decrease page table size and allow programs to be shared by defining a shared segment of pages. The process of moving pages in and out of main storage is known as *paging* and has also been described by Flores[25] and Oppenheimer and Weizer.[26] Virtual memory and paging has been the subject of much analysis; Denning[27] gives a good summary of research in that area.

Thus, virtual memory provides a storage hierarchy system described as follows:

1. A large address space as seen by the programmer for the preparation and execution of programs.
2. Main storage containing the needed pages for one or more programs being multiprogrammed.
3. An auxiliary paging device such as drum for program residence.
4. A backup paging device to which infrequently used pages can be migrated from drum storage.

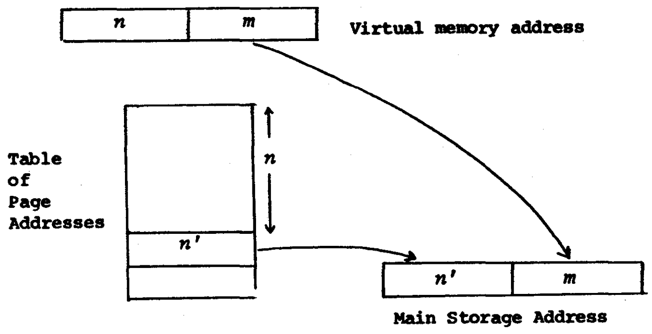


Figure 12—Address translation (one level page tables)

The virtual memory storage hierarchy is depicted in Figure 15.

Uses of virtual memory

Unlike buffer/core systems, virtual memory storage hierarchy systems need not be transparent to the user. Virtual memory has also been used* with small computers (see Christensen and Hause[28]) to give the programmer a large address space through a combination of main storage and direct access storage devices. In this instance, the user is provided with *expanded functional capability* at the cost of reduced throughput and a moresophisticated operating system.

Virtual memory has also been used to increase system throughput by improving upon conventional storage management techniques. The Boeing Company has modified OS/360 to utilize the virtual memory features of the IBM 360/67. The methodology involves overcommitting main storage by declaring more job partitions than the computing system can support and by mapping those partitions through the dynamic address translation feature of the Model 67[29,30] onto main storage and magnetic drum. The implementation, called OS/67 and depicted in Figure 16, increases performance because of the ability of the system to assign unused main storage to other virtual partitions when small programs are executing.

DATA ORGANIZATION AND MANAGEMENT

Hierarchy systems for data organization and management require a unique mixture of hardware features, software facilities, and user applications and, at this point in time, are not as well-defined as those previously surveyed in this paper. However, the basic objective is the same—to increase the frequency with which needed information can be retrieved from the faster device of a storage hierarchy. Hierarchical data organization also provides for efficiency of storage by reducing the amount of redundant information.

Hierarchical data management

The subject of hierarchical data management has been studied from different points of view by different authors. Opler[31] was concerned with the general storage assignment problem and defined four attributes of direct-access device types: flow rate, capacity, waiting time, and access time. On the basis of these attributes, he developed a hierarchy of main storage, large capacity storage, drum, disk, and magnetic strip transport systems and discussed the problems of effectively using these devices. This was the first definitive description of hierarchical data management systems.

In another important paper, Madnick and Alsop[32] discuss the levels of hierarchy in a modular file option. In this approach, the file is considered the most important unit of organization and emphasis is placed upon the logical file system as seen by the user. Madnick and Alsop define six hierarchical levels as follows:

1. Input/output control system,
2. Device strategy modules,
3. File organization strategy modules,
4. Basic file system,

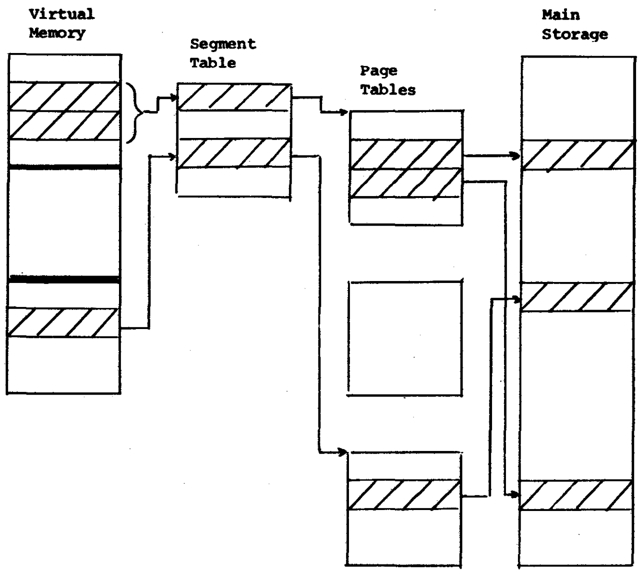


Figure 13—Two level page tables

* That is, in addition to the classical virtual memory systems such as TSS/360 or MULTICS that permit virtual memory addresses up to the addressing capability of the hardware.

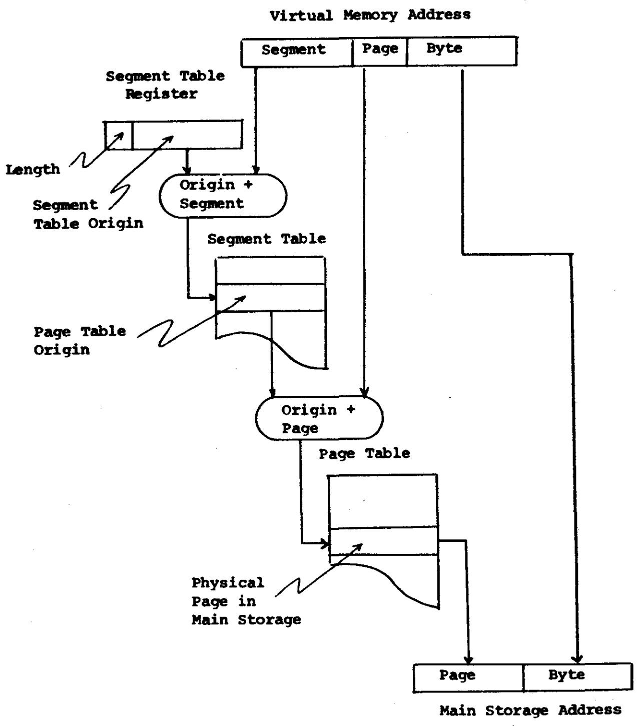


Figure 14—Segmented addressing

5. Logical file system, and
6. Access methods and user interface.

At the opposite extreme to file structures are data management systems[33, 34] in which emphasis is placed on the access of individual data items. Dodd[35] surveys systems of this sort and describes structural relationships between discrete data.

Data base organization

Inefficiencies in both the storage and retrieval of information led several researchers (see, for example, Mealy[36] and Chapin[37]) to explore the applications of

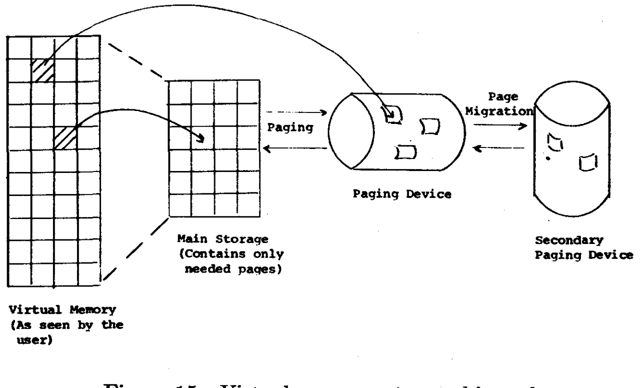

Paging
Page Migration
Paging Device
Main Storage (Contains only needed pages)
Secondary Paging Device
Virtual Memory (As seen by the user)

Figure 15—Virtual memory storage hierarchy

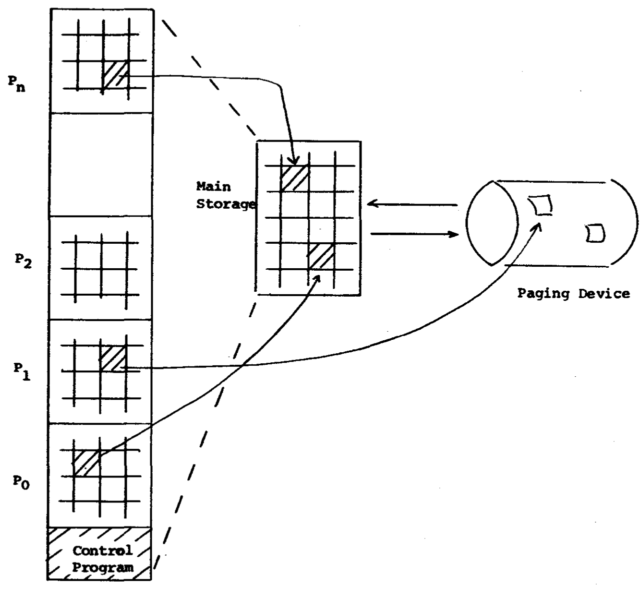


Figure 16—Virtual memory partitioning in OS/67 (conceptual overview)

data and to recognize the distinction between data access and data organization. This work combined with the increased information needs of our society (see Aron[38]) eventually led to the data base technology and data structures that we know today.

Presently, it is customary to distinguish between data structure and storage structure (Figure 17) in which *information* is meaning assigned to *data* and recorded on a *storage* device. Logical data, as viewed by the user, is used independently of its physical storage—which can be organized to facilitate access and conserve space. Thus, the logical data structures depicted in Figure 18 can be stored as the hierarchical data structure, shown in Figure 19. Although the examples are indeed oversimplified, they are indicative of the kind of information systems that are required for the interrogation and maintenance of large centralized information files. The advantages of hierarchical data structures can be listed as follows:[39]

1. Elimination of *redundant* data.
2. Use of the same data by all users of the system (*consistency*).
3. Program independence from physical storage devices.

As such, hierarchical data structures are essential for effective data base technology. The design and structure of data base systems are also described by Blier and Vorhaus[40] and the CODASYL data base reports.[41]

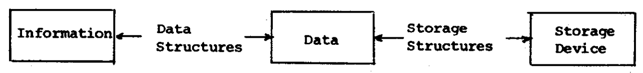


Figure 17—Distinction between information, data, and storage

Data set migration and backup storage

No survey of storage hierarchy systems would be complete without mention of the migration of data sets (files) from secondary or on-line storage to backup storage. The problem is particularly significant in a utility-class time sharing system such as TSS/360 or MULTICS in which file storage is considered to be on-line (or public) and may be viewed as infinite in extent by the user. Obviously, on-line storage is not infinite and information stored therein must be managed carefully. Files that are no longer needed must be purged from the system and infrequently used information must be moved to backup storage—a process frequently referred to as *data migration.*

The MULTICS system at M.I.T. (see Daley and Neumann[42]) combines a file backup system with a weekly dumping procedure that effectively migrates infrequently used information. First, a copy of all files created or modified by a user during a terminal session are duplicated on magnetic tape when he signs off. These tapes, termed *incremental dump tapes,* are replaced periodically and provide a means of reconstructing file storage in case of catastrophic system failure. Another tape dump is made periodically of files used in that period. The latter process facilitates the reconstruction of on-line storage and provides a means of eliminating unused files. A file directory entry remains in the system indefinitely unless it is explicitly deleted by the user. When a user desires to retrieve a file from backup storage, the correct set of dump tapes are retrieved and mounted as requested by the file retrieval procedure in MULTICS.

The expanded use of on-line storage in the TSS/360 system at IBM Research created a need for a storage hierarchy of on-line direct-access volumes, off-line direct access volumes, and tape volumes (see Considine and Weis[43]). The authors developed a set of migration commands that effectively copy the contents of public storage while migrating, to archival storage, data sets that fail a test of currency. The user is also provided with commands for determining which of his data sets have been migrated and other commands for retrieving them from secondary storage. The authors conclude their paper with an interesting comment to the effect

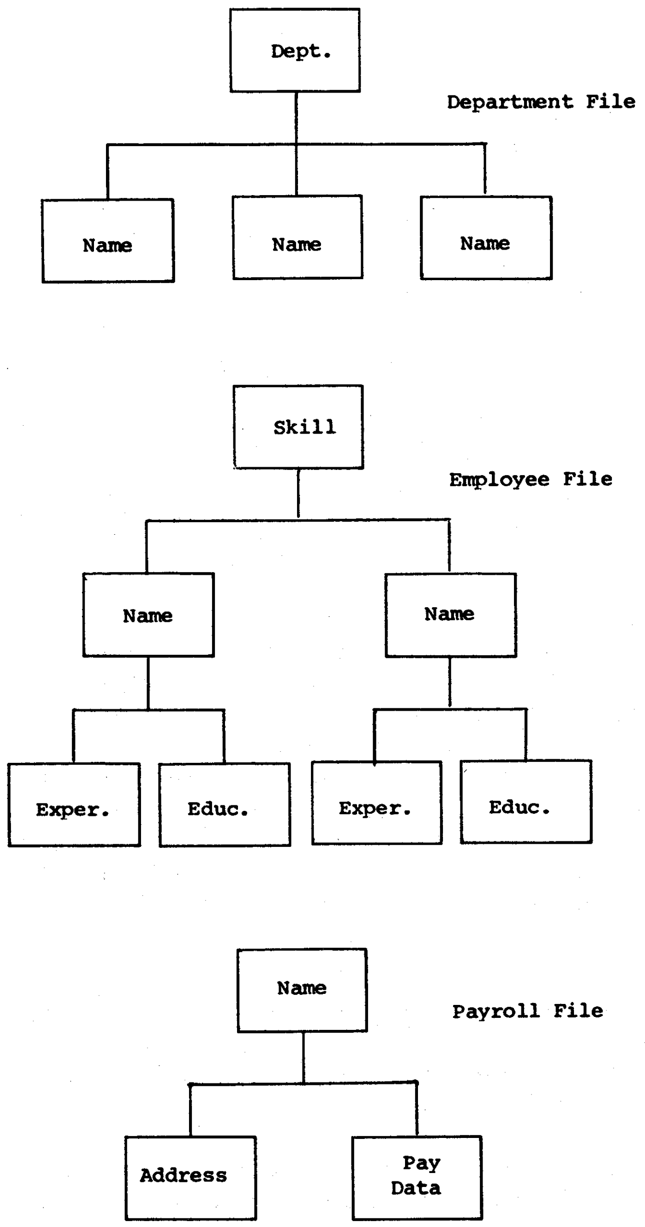


Figure 18—Logical data structures

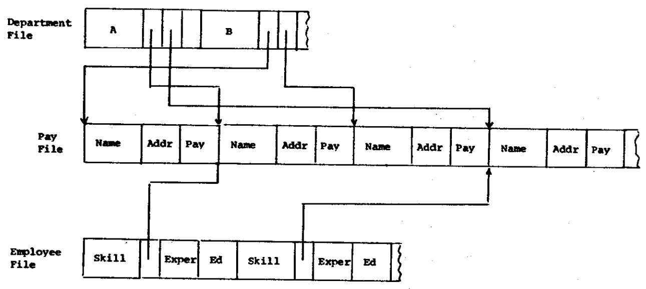


Figure 19—Physical data structures

that the amount of space gained by moving data sets to archival storage more than pays for the effort involved and that most of the data moved to archival storage have stayed there.

SUMMARY

Storage hierarchy systems are a means of improving the cost/performance ratio of storage components by combining elements with different characteristics to meet the needs of a given class of applications. In general, the process involves the organization of data on fast and slower devices into manageable units and by transferring the most frequently used data to the faster device on a dynamic basis.

Storage hierarchy systems fall into 3 general categories:

1. Buffer/core systems for addressable main storage that are transparent to the programmer.
2. Storage management systems such as virtual memory that are used by the programmer.
3. Data organization and management systems that permit large amounts of information to be stored and retrieved efficiently.

Collectively, storage hierarchy systems present a new approach to the storage aspects of computer technology wherein performance improvements are made through system design and organization in conjunction with improvements in the components themselves.

REFERENCES

1 H KATZAN
Advanced programming: Programming and operating systems
Van Nostrand Reinhold Co 1970

2 H KATZAN
Operating systems architecture
Proceedings of the Spring Joint Computer Conference 1970

3 J GECSEI D R SLUTZ I L TRAIGER
Evaluation techniques for storage hierarchies
IBM Systems Journal Volume 9 Number 2 1970

4 J S LIPTAY
Structural aspects of the System/360 Model 85: II The cache
IBM Systems Journal Volume 7 Number 1 1968

5 C J CONTI D H GIBSON S H PITKOWSKY
Structural aspects of the System/360 Model 85: I General organization
IBM Systems Journal Volume 7 Number 1 1968

6 J O MURPHEY R M WADE
The IBM 360/195
Datamation April 1970

7 *IBM System/360 Model 195 functional characteristics*
IBM Corporation Form A22-6943 August 1969

8 *A guide to the IBM/370 Model 155*
IBM Corporation Form GC20-1729-0 1970

9 *A guide to the IBM System/370 Model 165*
IBM Corporation Form GC20-1730 1970

10 H C LAUER
Bulk core in a 360/67 time-sharing system
Proceedings of the Fall Joint Computer Conference 1967

11 D N FREEMAN
A storage-hierarchy system for batch processing
Proceedings of the Spring Joint Computer Conference 1968

12 R E FIKES H C LAUER A L VAREHA
Steps toward a general-purpose time-sharing system using large capacity core storage and TSS/360
Proceedings of the 1968 ACM National Conference

13 Y C CHEN S C HSIEH
Selective transfer analysis
IBM Research Report RC 1926 October 25 1967

14 L I DINNERSTEIN
The CDC 7600—a giant in our time
Data Processing Magazine May 1969

15 T H ELROD
The CDC 7600 and SCOPE 76
Datamation April 1970

16 O W JOHNSON J R MARTINSON
Virtual memory in time sharing System/360
TSS/360 Compendium
IBM Data Processing Division 1969

17 B C LANZANO
Loader standardization for overlay programs
Communications of the ACM Vol 12 No 10 October 1969

18 H A KINSLOW
The time-sharing monitor system
Proceedings of the Fall Joint Computer Conference 1964

19 F J CORBATO et al
The compatible time-sharing system
The MIT Press Cambridge Massachusetts 1963

20 A S LETT W L KONIGSFORD
TSS/360: A time-shared operating system
Proceedings of the Fall Joint Computer Conference 1968

21 F J CORBATO V A VYSSOTSKY
Introduction and overview of the MULTICS system
Proceedings of the Fall Joint Computer Conference 1965

22 B RANDELL C J KUEHNER
Dynamic storage allocation systems
Communications of the ACM Vol 11 No 5 May 1968

23 J B DENNIS
Segmentation and the design of multiprogrammed computer systems
Journal of the ACM Vol 12 No 4 October 1965

24 B W ARDEN B A GALLER T C O'BRIEN F H WESTERVELT
Program and addressing structure in a time-sharing environment
Journal of the ACM Vol 13 No 1 January 1966

25 I FLORES
Virtual memory and paging
Datamation August 1967—Part I
Datamation September 1967—Part II

26 G OPPENHEIMER N WEIZER
Resource management for a medium scale time-sharing operating system
Communications of the ACM Vol 11 No 5 May 1968

27 P J DENNING
Virtual memory
Computing Surveys Vol 2 No 3 September 1970

28 C CHRISTENSEN A D-HAUSE
A multiprogramming, virtual memory system for a small computer
Proceedings of the Spring Joint Computer Conference 1970

29 W T COMFORT
A computing system design for user service
Proceedings of the Fall Joint Computer Conference 1965

30 C T GIBSON
Time-sharing in the IBM system/360: Model 67
Proceedings of the Spring Joint Computer Conference 1966

31 A OPLER
Dynamic flow of programs and data through hierarchical storage
Proceedings of the IFIP Congress 1965

32 S E MADNICK J W ALSOP
A modular approach to file system design
Proceedings of the Spring Joint Computer Conference 1970

33 P J DIXON J SABLE
DM-1—a generalized data management system
Proceedings of the Spring Joint Computer Conference 1967

34 D B NELSON R A PICK K B ANDREWS
GIM-1—a generalized information management language and computer system
Proceedings of the Spring Joint Computer Conference 1967

35 G G DODD
Elements of data management systems
Computing Surveys Vol 1 No 2 June 1969

36 G H MEALY
Another look at data
Proceedings of the Fall Joint Computer Conference 1967

37 N CHAPIN
A deeper look at data
Proceedings of the ACM National Conference 1968

38 J D ARON
Information systems in perspective
Computing Surveys Vol 1 No 4 December 1969

39 *Information Management System/360*
IBM Corporation Form GH20-0765 March 1970

40 R E BLIER A H VORHAUS
File organization in the SDC time-shared data management system (TDMS)
System Development Corporation SP-2907 August 1968

41 CODASYL Reports
(1) *A survey of generalized data base management systems*
(2) *CODASYL data base task group*
Available from the ACM

42 R C DALEY P G NEUMANN
A general-purpose file system for secondary storage
Proceedings of the Fall Joint Computer Conference 1965

43 J P CONSIDINE A H WEISS
Establishment and maintenance of a storage hierarchy for an on-line data base under TSS/360
Proceedings of the Fall Joint Computer Conference 1969

Optimal sizing, loading and re-loading in a multi-level memory hierarchy system

by S. R. ARORA

University of Minnesota and UNIVAC Division, Sperry Rand Corporation
Roseville, Minnesota

and

A. GALLO

UNIVAC Division, Sperry Rand Corporation
Roseville, Minnesota

Starting with the appearance of the third generation computers the demand for memory storage devices has increased. This increased demand has been for both the main storage and auxiliary storage devices. The major reasons for this increase were the problems of larger data processing tasks and the introduction of multi-programming and time-sharing.

The increase of memory storage space cannot be accomplished economically without reducing the speeds of the memories. System designers have been trying to exploit the heterogeneous nature of programs and data files so that files that are least frequently accessed are stored in slower memory modules and files with high activity are loaded in faster memory modules. This has been done more or less on heuristic basis.

This problem became more acute with the introduction of the directly addressable bulk. In this system organization the processor executes instructions and data fetches directly from the bulk core, which operates at a lower speed than the main memory. The proper allocation of the data and program segments in different memory modules became very critical in the overall performance.

This paper studies the effects of loading and re-loading of program and data segments in various directly addressable memory modules. It seeks optimal rules of loading and re-loading. It also studies the problem of optimal sizing of different memory modules.

We first select the criterion of effectiveness with respect to the problem of memory allocation and memory sizing decisions. We define the central processor unit and the directly addressable memory levels as a sub-system of the whole computer system and take response time as the performance measure of this sub-system. Considering the central processor as a single server, the response time of the sub-system is the serial sum of the instruction execution times of the various program modules making up a benchmark. We will call the individual programs and data segments "objects." The collection of all objects is our benchmark. We characterize an object through the following parameters.

j = number identifying an object

K = total number of objects in the benchmark

$$j = 1, 2, \ldots, K$$

v_j = size of the jth object

p_j = total number of references made to object j during the total benchmark run time

i_j = average number of instructions or data fetches per reference to object j

R = total run time of the selected benchmark

As stated earlier, an object can be a set of instructions or a set of data fetches. It can belong to the operating system or to the user's library. Some of the objects will show high level of activity, which means that the processor will loop through a small set of instructions many times during one reference. Typically, loop bound programs are of this type. In other cases out of a large quantity of instructions or data only a few will be executed during one reference. Data banks and decision bound programs typically belong to this category.

OPTIMAL LOADING RULE IN VARIOUS DIRECTLY ADDRESSABLE MEMORY LEVELS

As stated earlier, our first objective is to determine how to load the given K objects in memory levels with different cycle times so that the total processing time of these objects is minimized.

Define

$$h_j = \frac{p_j i_j}{v_j}\ ; \quad j=1, 2, \ldots, K \tag{1}$$

Then the optimal loading rule is that the objects are loaded, starting from the fastest memory level to the slowest in descending order of the index h_j.

Theorem

The total processing time R for processing K objects in an N level memory system with sizes s_n is minimized if objects are loaded within these levels, starting with the fastest memory in descending order of the quantity h_j.

Proof

Consider two objects j and j' with sizes v_j and v_j' and indexes h_j and h_j' such that $h_j' > h_j$. Consider two adjacent memory levels n and $n+1$, with speeds t_n and t_{n+1}. Let object j be currently loaded in level n and object j' in level $n+1$, which is contrary to the optimal loading rule. Let R' be the total response time with the current loading and R be the total response time if objects j and j' were to interchange their levels over the space v, where $v = \min(v_j, v_j')$. Values of R and R' are given by

$$R' = L + t_n p_j i_j \cdot \frac{v}{v_j} + t_{n+1} p_j' i_j' \frac{v}{v_j'} \tag{2}$$

$$R = L + t_{n+1} p_j i_j \frac{v}{v_j} + t_n p_j' i_j' \cdot \frac{v}{v_j'} \tag{3}$$

where L is the response time of objects other than j and j'. The ratios v/v_j, v/v_j', denote the fractions of the objects j and j' which are affected by the above interchange over the space v.

Substituting (1) in (2) and (3), we obtain

$$R' = L + t_n h_j v + t_{n+1} h_j' \cdot v \tag{4}$$

$$R = L + t_{n+1} h_j v + t_n h_j' \cdot v \tag{5}$$

Subtracting (4) from (5) we obtain

$$\begin{aligned} R - R' &= (t_{n+1} - t_n) h_j v - (t_{n+1} - t_n) h_j' v \\ &= (t_{n+1} - t_n)(h_j - h_j') \cdot v \end{aligned} \tag{6}$$

Since $(t_{n+1} - t_n)$ is positive and $(h_j - h_j')$ is negative by assumptions, $R - R'$ is negative. In other words the total response time has been reduced by the above interchange of objects j and j'. Continuing similar interchange over other objects, we will converge to the optimal loading rule, which will minimize the total response time. Arranging all the objects in the descending order of index h_j makes the loading rule optimal for any N level memory system of any sizes, including the particular case when the $\{s_n\} \equiv \{v_j\}$.

The heterogeneity of the objects makes it economical to execute them from memory levels with different speeds. Let us define the following parameters for our memory hierarchy system.

t_n = instruction execution time per instruction from the nth memory level
c_n = cost per unit space per unit time for the nth memory level

Assume that memories are numbered according to their speeds, the fastest is assigned number 1 and the slowest is assigned number N. It is clear that a memory hierarchy with different speeds and different costs for various memory levels is meaningful only if they satisfy the following relationships:

$$t_1 < t_2 < \cdots < t_N$$
and
$$c_1 > c_2 > \cdots > c_N \tag{7}$$

ACTIVITY PROFILE CURVE

We define the activity level of program j to be equal to $p_j i_j$, which is also equal to $v_j h_j$ according to equation (1). The activity profile curve establishes a relationship between the cumulative activity level over the collection of objects and the cumulative memory space demand for these objects, when objects are assumed to be numbered according to the index h_j. The object with the highest value of the index is numbered 1 and the one with the lowest value is numbered K. Let $F(s)$ in Figure 1 denote the activity profile curve.

$F(s)$ and s are defined as below:

$$F(s) = \sum_{j=1}^{k} p_j i_j$$

$$s = \sum_{j=1}^{k} v_j \tag{8}$$

where k is a subset of the objects K. In particular point A on the s-axis and point B on the $F(s)$ axis are given by

$$A=\sum_{j=1}^{K} v_j$$

$$B=\sum_{j=1}^{K} p_j i_j \tag{9}$$

If we normalize $\sum_{j=1}^{k} p_j i_j / \sum_{j=1}^{K} p_j i_j$, then we have a cumulative distribution curve. Let $F'(s)$ denote the normalized function. The shape of the curve $F'(s)$ can be obtained by curve fitting techniques from the actual file data. Exponential curve given by $F(s)=1-e^{-\alpha s}$ was observed to fit several benchmarks, where the parameter characterizes the particular shape of the function for the given benchmark.

OPTIMAL SIZING OF VARIOUS ADDRESSABLE MEMORY LEVELS

The response time R for the given benchmark in the specified sub-system of the processor and directly addressable memories is given by

$$R=\sum_{n=1}^{K}\sum_{j=1}^{K} t_n p_j i_j x_{n,j} \tag{10}$$

where $0 \leq x_{n,j} \leq 1$. It takes the value 0, if the object is not loaded in the nth level, a value of 1, if it is completely loaded and a value $0 < x_{n,j} < 1$, if it is fractionally loaded in the nth level.

Let $s_1, \ldots, s_n, \ldots, s_N$ denote the sizes for the N memory levels. Assuming that loading of the objects follows the optimal loading rule, the response time R

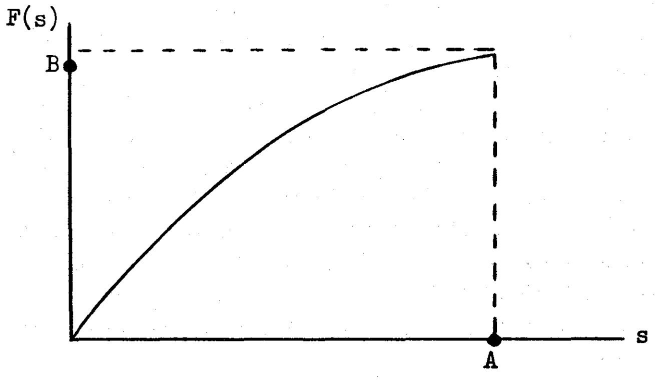


Figure 1—Activity profile curve for the given benchmark

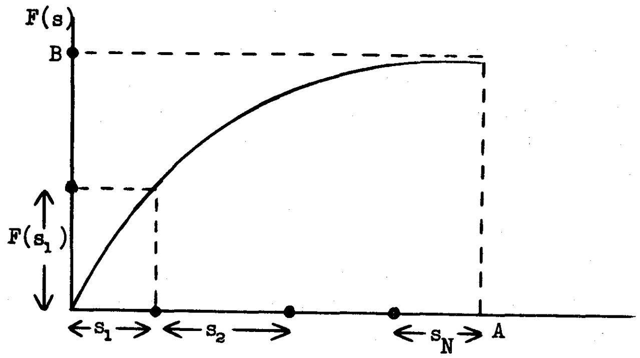


Figure 2

may be written as

$$R=t_1 F(s_1)+t_2[F(s_1+s_2)-F(s_1)]+\cdots$$
$$+t_N\left[F\left(\sum_{n=1}^{N} s_n\right)-F\left(\sum_{n=1}^{N-1} s_n\right)\right] \tag{11}$$

Equation (11) is derived from equation (10) by substituting $F(s)$ for $\sum_{j=1}^{k} p_j i_j$ as defined in equation (8).

The cost of processing the benchmark is given by the following equation.

$$Z=R\left(\sum_{n=1}^{N} c_n s_n+G\right) \tag{12}$$

where G is the unit time rental cost of the processor. Z measures the total cost incurred by the sub-system for running the benchmark. The quantity

$$\left(\sum_{n=1}^{N} c_n s_n+G\right)$$

denotes the unit time rental cost of the sub-system and this rental cost is paid for the total response time R. The objective is to minimize Z subject to the constraint that the response time has to be smaller than a certain fixed number. Mathematically, the problem of optimal sizing may be stated as

Find $s_n \geq 0$ for $n=1, \ldots, N$

which minimize

$$Z=R\left(\sum_{n=1}^{N} c_n s_n+G\right) \tag{13}$$

subject to

$$R \leq R_0 \tag{14}$$

Minimization may be achieved by defining the Lagrange Function Z' and applying Kuhn Tucker conditions. We will simplify our problem by initially ignoring the non-negativity constraint on s_n and also by treating (14) as an equality constraint. If on solving the first order conditions, it is found that some of the s_n values are negative, such solutions will be unfeasible. The boundary solutions, where some of the s_n values were zero, will be examined systematically according to the algorithm described below. The Lagrange function Z' is given by

$$Z' = R\left(\sum_{n=1}^{N} c_n s_n + G\right) - \lambda(R - R_0) \qquad (15)$$

If Z' is minimized at $(s_1^*, \ldots, s_n^*, \ldots, s_N^*)$ the following conditions must be satisfied at the optimal point.

$$\frac{\partial Z'}{\partial s_n} = \frac{\partial R}{\partial s_n}\left(\sum_{n=1}^{N} c_n s_n + G\right) - \lambda \frac{\partial R}{\partial s_n} + Rc_n = 0$$

$$= \frac{\partial R}{\partial s_n}\left(\sum_{n=1}^{N} c_n s_n + G - \lambda\right) + Rc_n = 0 \qquad (16)$$

$$\text{for} \quad n = 1, 2, \ldots, N$$

$\partial R/\partial s_n$ defines the rate of change in the total response time R with respect to a change in s_n, keeping the size of other levels unchanged. Subtracting equation (16) with index $(n-1)$ from the same equation with index (n), we obtain

$$\left(\frac{\partial R}{\partial s_n} - \frac{\partial R}{\partial s_{n-1}}\right)\left(\sum_{n=1}^{N} c_n s_n + G - \lambda\right) + R(c_n - c_{n-1}) = 0 \qquad (17)$$

$$\text{for} \quad n = 2, 3, \ldots, N$$

On re-arranging we obtain

$$\frac{\left(\dfrac{\partial R}{\partial s_n} - \dfrac{\partial R}{\partial s_{n-1}}\right)}{c_{n-1} - c_n} = \frac{R}{\displaystyle\sum_{n=1}^{N} c_n s_n + G - \lambda} \qquad (18)$$

$$\text{for} \quad n = 2, 3, \ldots, N$$

By differentiating R from equation (11) with respect to s_n and s_{n-1} and subtracting the two derivatives we obtain

$$\left(\frac{\partial R}{\partial s_n} - \frac{\partial R}{\partial s_{n-1}}\right) = (t_n - t_{n-1}) f(s_1 + s_2 + \cdots + s_{n-1}) \qquad (19)$$

where $f(s)$ denotes the derivative of the function $F(s)$.

Substituting (19) in (18), we obtain

$$\frac{(t_n - t_{n-1})}{(c_{n-1} - c_n)} f(s_1 + s_2 + \cdots + s_{n-1}) = \frac{R}{\displaystyle\sum_{n=1}^{N} c_n s_n + G - \lambda}$$

$$= \text{constant} \qquad (20)$$

$$\text{for} \quad n = 2, 3, \ldots, N$$

From equation (20) we obtain $(N-2)$ equations of the form

$$\frac{(t_n - t_{n-1})}{(c_{n-1} - c_n)} f(s_1 + s_2 + \cdots + s_{n-1})$$

$$= \left(\frac{t_{n+1} - t_n}{c_n - c_{n+1}}\right) f(s_1 + s_2 + \cdots + s_n) \qquad (21)$$

We also have to satisfy the following two conditions, which provide us with the additional two equations needed to solve the optimal values of s_n.

$$s_1 + s_2 + \cdots + s_N = \sum_{J=1}^{K} v_j \qquad (22)$$

$$R = R_0 \qquad (23)$$

SEQUENCE OF ADMISSIBLE MEMORIES

It may be recalled that while defining the activity profile curve, objects were numbered in descending order of the index h_j. Consequently the derivative $f(s)$ of the activity profile curve $F(s)$ is a monotonic non-increasing function of s. From equation (21) we conclude that the sequence $\{(t_n - t_{n-1})/(c_{n-1} - c_n)\}$ for all n is monotonic non-decreasing. Any memory level which does not comply with this rule is a non-admissible memory level. Memory levels of this type can be eliminated from search for optimal solutions.

ALGORITHM FOR SOLVING OPTIMAL VALUES OF MEMORY SIZES

Optimal solution of the memory sizes has to satisfy equations (21) through (23). After the non-admissible memories have been eliminated, the sequence $\{(t_n - t_{n-1})/(c_{n-1} - c_n)\}$ for the admissible memories is monotonic non-decreasing. Also if objects are arranged in the descending order of the index h_j, the derivative $f(s)$ of the activity profile curve $F(s)$ is a monotonic non-increasing function of s.

The following algorithm will systematically watch for the compliance of equations (21) through (23). Initially load all the objects in the slowest memory level N, and test whether the response time computed with equation (11) satisfies the specified response time constraint given in equation (23). If this constraint is satisfied, then the optimal solution has already been found, otherwise include the next faster memory level $N-1$.

We have two unknown values which we can solve by applying equations (22) and (23). If equation (23) indicates that the response time constraint cannot be

TABLE I—Airline Reservation Benchmark

j	id	h_j	$p_j i_j$	$\sum p_j i_j$	v_j	$\sum v_j$
1	P	8.66	18300	18300	2112	2112
2	P	6.14	1560	19860	254	2366
3	P	.8125	1560	21420	1920	4286
4	P	.781	300	21720	384	4670
5	P	.60	3600	25320	6000	10670
6	P	.60	1920	27240	3200	13870
7	P	.558	1536	28776	2752	16622
8	P	.441	112	28888	254	16876
9	P	.361	300	29188	832	17708
10	P	.320	480	29668	1500	19208
11	P	.291	317	29985	1088	20296
12	P	.260	1560	31545	6000	26296
13	P	.226	780	32325	384	26680
14	P	.223	780	33105	3500	30180
15	P	.223	780	33885	3500	33680
16	P	.148	95	33980	640	34320
17	P	.140	252	34232	1800	36120
18	P	.118	375	34607	3168	39288
19	D	15.0×18^{-4}	510	35117	340000	379288
20	D	8.35×12^{-4}	431.7	35548.7	516800	896088
21	D	6.06×10^{-4}	.2	35548.9	330	896418
22	D	2.65×10^{-4}	.56	35549.46	2112	898530
23	D	2.22×10^{-4}	28	35577.46	126000	1024530
24	D	1.47×10^{-4}	41.97	35619.43	285360	1309890
25	D	1.3×10^{-4}	3.0	35622.43	21500	1331390
26	D	$.972\times10^{-4}$	5.25	35627.68	54012	1385402
27	D	$.62\times10^{-4}$	3.0	35630.68	48000	1433402
28	D	$.53\times10^{-4}$	.07	35630.75	1320	1434722
29	D	$.456\times10^{-4}$	27.36	35658.11	600000	2034722
30	D	$.433\times10^{-4}$	4.9	35633.01	113154	2147876
31	D	$.38\times10^{-4}$	5.4	35668.41	143160	2291236
32	D	$.20\times10^{-4}$	.02	35668.43	1000	2292236
33	D	$.185\times10^{-4}$	7.99	35676.42	432000	2724236
34	D	$.175\times10^{-4}$	105.00	35781.42	6000000	8724236
35	D	$.167\times10^{-4}$	1.5	35782.92	90024	8814260
36	D	$.159\times10^{-4}$	.07	35782.99	4400	1818660
37	D	$.059\times10^{-4}$	45.20	35828.19	7630000	1644860
38	D	$.0415\times10^{-4}$	.49	35828.68	118060	16566720
39	D	$.033\times10^{-4}$	.1	35828.78	30030	16596750
40	D	$.030\times10^{-4}$	4.04	35832.82	1333352	17930102
41	D	$.0249\times10^{-4}$	.05	35832.87	20064	17950066
42	D	$.0207\times10^{-4}$	1.0	35833.87	482600	18432666
43	D	$.016\times10^{-4}$	.8	35834.67	264000	18696666
44	D	$.00413\times10^{-4}$	.01	35834.68	21102	18797768
45	D	$.0016\times10^{-4}$	.01	35834.69	60060	18777828
46	D	$.0014\times10^{-4}$	.02	35834.71	151032	18928860
47	D	$.00012\times10^{-4}$	.0003	35834.713	240042	19168902

satisfied, the next faster memory has to be included and now three equations with three unknowns have to be solved; equations (23), (22) and out of the set of equations given in (21) we include the following:

$$\frac{t_{N-1}-t_{N-2}}{c_{N-2}-c_{N-1}}f(s_1, s_2\cdots s_{N-2})=\frac{t_N-t_{N-1}}{c_{N-1}-c_N}f(s_1, s_2\cdots s_{N-1})$$

If the response time constraint cannot be satisfied, we include the next faster memory and bring in one more equation from the set of equations given in (21). The first time a feasible solution exists, we try to obtain the optimal solution. If we want to carry on the optimization with the technical constraints of the smallest size of the memory modules, an adjustment can be made so that $R\leq R_0$ should be observed. If the response time constraint cannot be satisfied, including the last available memory level, then no feasible solution exists.

If the solution to the involved equations violates the non-negativity constraint on some s_n in the set of included memory levels, then all combinations of various s_n being equal to zero have to be tested for the optimal solution.

Example

In this example we are taking an airline reservation system. The memory allocation for the objects in an airline reservation system is reasonably static. The granularity of the program and data files recognizes 47 objects with a total memory requirement of 19.2M words. Suppose we decided to buy 2M words of directly addressable memory levels. We have to make the selection out of three available memory levels. The speeds of these memory levels are

$$t_1 = .5\ \mu s \qquad t_2 = .75\ \mu s \qquad t_3 = 1.5\ \mu s$$

The proportional costs of the respective memory levels are

$$c_1 = 7 \qquad c_2 = 5 \qquad c_3 = 1$$

Table I contains the file elements arranged in the descending order of the index h_j, along with their parameters h_j, $p_j i_j$, $\sum p_j i_j$, v_j and $\sum v_j$. The second column of the table identifies the objects; P stands for programs and D stands for data.

For our purposes we fitted an exponential curve over the first 29 objects for which the directly addressable memory is being considered. The equation of the exponential curve is given by

$$F(s) = 1 - e^{-(\alpha_0 + \alpha_1 s)}$$

The least square estimates for α_0 and α_1 are

$$\hat{\alpha}_0 = 1.858$$

$$\hat{\alpha}_1 = .24166 \times 10^{-5}$$

It may be verified that all three memory levels are admissible.

Consider the three cases when $R \leq 30$ msec., $R \leq 20$ msec. and $R \leq 10$ msec. respectively.

For $R = 30$ assume we have s_3 memory level only. Multiplying the total activity with the instruction execution time of the 3rd memory level we obtain 53.5 msec. which does not satisfy the response time constraint.

Including s_2 we solve equations (22) and (23) and obtain

$$s_2 = 1973704$$

$$s_3 = 26296$$

For $R = 20$ the response time constraint cannot be satisfied with memory levels s_3 and s_2 only. s_1 also has to be included. Solving equations (21), (22) and (23) we obtain

$$s_1 = 265660$$

$$s_2 = 168250$$

$$s_3 = 1566090$$

For $R = 10$ no feasible solution exists. This can be seen also when we multiply the total activity level by the instruction execution time obtained with the fastest memory level only. This is 17.83 msec. This is the best response time which can be obtained by buying the fastest memory level only, which is the most expensive system.

DYNAMIC ALLOCATION

In the real life one benchmark may not be representative of the working conditions of a computer system. It might be desirable to observe several benchmarks. The collection of objects in different benchmarks might overlap. This is particularly evident for the operating systems, which will be accessed by all the benchmarks. Under these conditions it may be meaningful to re-locate some of the objects. The criterion of re-location is that if the improvement in response time is greater than the re-location overhead, including the work required to do time-to-time checking for carrying on necessary re-location, then re-location is desirable. Define the following notations.

$K_{r,n}$ = set of objects accepted in memory level n at decision time r for the time period $(r, r+1)$ (Figure 3).

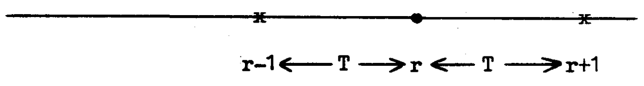


Figure 3

$K_{r,n}^*$ = set of objects that should be loaded in level n at time period r, if optimal loading rule was followed.

$$K_{r,n} = \begin{cases} K_{r-1,n}, & \text{when no re-allocation is decided at time } r \\ K_{r,n}^*, & \text{when re-allocation is decided at time } r \end{cases}$$

$J_{n,m}$ = set of objects that have to be transferred from level n to level m, if re-allocation is decided at time r.

D_r = difference in response time between re-allocating and not re-allocating.

E = fixed switching overhead incurred at every re-allocation.

ΔE = additional switching overhead to implement the checking algorithm.

G_r = the frequency of testing whether a re-allocation should take place. If the re-allocation is pre-set externally, $\Delta E = 0$.

The difference in response time can be expressed as

$$D_r = \sum_{n=1}^{N} \sum_{j \epsilon K_{r,n}} t_n p_j(r) i_j - \sum_{n=1}^{N} \sum_{j \epsilon K_{r,n}*} t_n p_j(r) i_j - \sum_{m=1}^{N} \sum_{n=1}^{N} \sum_{j \epsilon J_{n,m}} t_{n,m} v_j - (E + \Delta E \cdot G_r) \quad (24)$$

Our decision rule for re-allocation is the following:

$$K_{r,n} = \begin{cases} K_{r-1,n} & \text{if } D_r < 0 \\ K_{r,n}* & \text{if } D_r \geq 0 \end{cases}$$

The time interval to check the necessity of re-loading the objects is denoted by T. It is a function of the environments and can be estimated on the basis of actual program behavior. There can be situations when T is large. It is realistic to assume that for larger T, it is more probable that we will have the case where $D_r \geq 0$.

IMPLEMENTATION

The users who were first to install the directly addressable memory hierarchies have spent considerable amount of effort to tune up their system.[2,3] The Carnegie-Mellon University has written over a period of two years, five algorithms for loading objects to improve their system performance.[2]

Some of the advantages of directly addressable memory hierarchies are heuristically obvious. List processing tasks are the classical examples, which system designers used to mention, while advocating a directly addressable slow memory in addition to a high speed primary memory level. Other heuristic suggestions for execution from slow memory levels are the IO bound programs, communication buffers (including IO symbionts), interactive (time sharing) jobs, etc.

While many of these heuristic suggestions would not contradict the findings of this study, it is appropriate to point out some of the pitfalls. Take, for example, the level of *IO* boundedness for loading criterion. Define the level of IO boundedness

$$f_j = \frac{(t_{\text{CPU}})_j}{(t_{\text{IO}})_j}$$

where $(t_{\text{CPU}})_j$ is the total processing time of object j, $(t_{\text{IO}})_j$ is the total time spent in processing IO requests for object j.

f_j is a close approximation to $p_j i_j$, the activity level of the jth object for a given benchmark run time as defined earlier in this study. However, the index f_j ignores v_j, the size of the jth object, thus any loading rule based on the ranking of the objects in the descending order f_j contradicts the optimal loading rule proved in this study, consequently it is not an optimum loading rule.

IBM Research Group has spent considerable amount of work to study the efficient use of multi-level memory hierarchies.[1,3,7] Some of the conclusions of the IBM studies regarding the 360 systems are the following: (a) have in the slow memory a staging area for swapping in the objects, (b) transfer the objects for processing from the slow memory to the fast memory on a selective basis. The selection rule for transfer is to make a transfer if

$$\frac{i_j}{v_j} > b$$

where i_j is the activity level of object j
v_j is the volume of the jth object
b is a carefully calculated threshold value

The activity level is defined prior to the use of the program. An IBM research paper[1] describes a preprocessor, which rearranges the object codes of a program according to their activity level for making selective transfers, when the object is scheduled for processing. The idea of selective transfer is in line with the idea of dynamic re-location described in this paper. Our paper does not treat the problem of staging, which is treated in the literature in connection with replacement algorithms.[6,8]

The present paper suggests memory allocation on the basis of *a priori* observed average characteristics of job mixes. Statistics known with respect to the operating systems give evidence that the program behavior of the operating systems is usually steady for a large diversity of job mixes, therefore the optimal loading rule and optimal sizing of memory levels can be easily observed for the operating systems. For user environments with large variability in the object characteristics, an implementation of the optimal loading rule on a dynamic basis could have little overhead, because only the ranking of one index has to be observed from time to time.

REFERENCES

1 Y C CHEN S C HSIEH
Selective transfer analysis
IBM Research

2 A L VEREHA R M RUTLEDGE M M GOLD
Strategies for structuring two level memories in a paging environment
Carnegie-Mellon University Research Paper

3 C V RAMAMOORTHY K M CHANDY
Optimization of memory hierarchies in multiprogrammed systems
ACM Journal July 1970

4 F BASKETT J C BROWNE W M RAIKE
The management of a multi-level non-paged memory system
AFIPS Spring Joint Computer Conference 1970

5 S R ARORA A GALLO
Optimal loading, reloading, memory sizing in a system with multi-level memory hierarchy
UNIVAC Tech Memo 01EN 108

6 P DENNING
Thrashing: Its causes and prevention
AFIPS March 1969

7 D N FREEMAN
A storage hierarchy system for batch processing
AFIPS March 1968

8 L A BELADY
A study of replacement algorithms for a virtual storage computer
IBM Systems Journal Vol 5 No 2 1966

The TABLON mass storage network

by RICHARD B. GENTILE and JOSEPH R. LUCAS, JR.

United States Department of Defense
Washington, D. C.

INTRODUCTION

This paper documents an approach by the Department of Defense for solving the massive volume digital tape storage problem. It describes a computer network called TABLON which provides several trillion (10^{12}) bits of on-line storage simultaneously to a number of dissimilar computer systems. Such a network has potential application to any company, major institution, or publicly supported organization with the following situations:

—they currently employ a number of large scale, third-generation computer systems,
—their data base is large enough to cause physical storage and/or access problems,
—their data accumulation is growing faster than their ability to pay for devices to contain it, and
—they have a strong desire to consolidate all their computerized information into one central location

The need for more and more on-line storage has long been a recognized problem. What is perhaps not so well recognized, is the desire of some large installations (and some sharing minded smaller ones) to consolidate their data base, automate it, and establish a centralized facility upon which all can feed equally. The associated problems are formidable. Containing the data base on-line is probably the most serious, with distribution, timely access, and file management problems close behind. And if these problems could be overcome, the different characteristics of today's computer systems would surely stop them the very first time they tried to exchange a file between two different processors.

For one element of the Department of Defense, this problem existed. A number of large magnetic tape libraries were "centrally located" within the same general area, but not central enough to reduce the enormous resources involved with constant movement of data to, from, and between libraries and computers. Multiple copies, manual intervention, antiquated accounting and administration, and a general inability to support an otherwise fully automated multi-computer complex were equal burdens.

TABLON then, was developed under two primary objectives. First, it was to permit near immediate accessibility and automatic distribution of any portion of the total data base to any of several dissimilar computer systems. Second, it was to reduce the cost, space, and administration from that currently required to support an extremely large magnetic tape library.

The TABLON system is a network of dedicated computers, special hardware components, and mass storage devices. It permits large scale third-generation computer systems (called User-CPU's) to store and retrieve data on a file, or file segment, basis. Storage and retrieval is automatic, via hard wired direct data transmission, upon exchange of commands between a User-CPU and the TABLON network. TABLON also solves the problem of providing dissimilar computer systems on-line file management access to the total data base.

Obviously TABLON is expensive in terms of initial dollar outlay. A half-dozen components in the million dollar range can be easily identified, not counting engineering costs for hardware development. And then there is software, done by DoD personnel, but still an expensive resource allocation. However, there are many organizations today whose collective inventory shows large disk and drum installations of 20 or more, some closer to 50. Multiply that number by a couple of hundred thousand dollars each (on a procurement basis) and one finds a multi-million dollar inventory for on-line storage alone.

For the Department of Defense the TABLON network represents cost avoidance as opposed to cost reduction. Recovery of cost for the storage devices will be achieved during the life of these equipments through a reduction in magnetic tape procurements. Cost re-

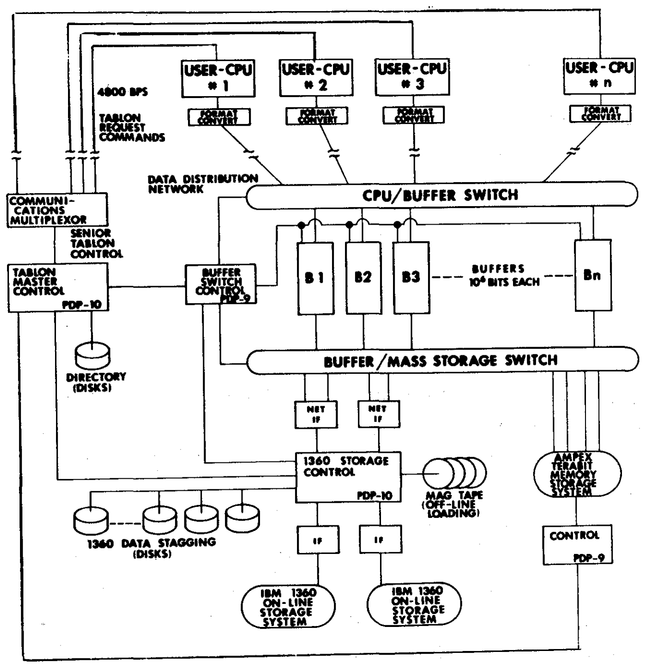

USER-CPU # 1
USER-CPU # 2
USER-CPU # 3
USER-CPU # n
4800 BPS
TABLON REQUEST COMMANDS
FORMAT CONVERT
DATA DISTRIBUTION NETWORK
CPU/BUFFER SWITCH
COMMUNI-CATIONS MULTIPLEXOR
SENIOR TABLON CONTROL
TABLON MASTER CONTROL PDP-10
BUFFER SWITCH CONTROL PDP-9
B1
B2
B3
BUFFERS 10⁶ BITS EACH
Bn
DIRECTORY (DISKS)
BUFFER/MASS STORAGE SWITCH
NET IF
1360 STORAGE CONTROL PDP-10
MAG TAPE (OFF-LINE LOADING)
AMPEX TERABIT MEMORY STORAGE SYSTEM
1360 DATA STAGGING (DISKS)
IF
CONTROL PDP-9
IBM 1360 ON-LINE STORAGE SYSTEM

Figure 1—Simplified TABLON diagram

covery for other components will be realized from a significant reduction in future storage peripherals (disks, drums) installed, plus an improvement in library maintenance and management. Moreover, TABLON's capacity for growth in the years to come gives it a forecasted useful life of ten years, bringing the per year cost well below current and forecasted annual disk and drum expenditures.

What follows is a summary description of the TABLON network, with primary emphasis on the authors' area of personal involvement in overall concept and management, and in the functional level hardware design and system integration.

OPERATIONAL DESCRIPTION

Today, when processing is completed at a User-CPU, output is written onto one or more reels of magnetic tape; the tape(s) being demounted and manually transported to a tape library. Using TABLON, instead of writing data onto digital tape(s), the User-CPU writes that same data directly into TABLON. To the User-CPU, the content, format, and other characteristics of the data are the same for TABLON as for magnetic tape. On recall by the (same or dissimilar) User-CPU, TABLON returns the data record by record, and in the proper character set, as does magnetic tape. Initially, TABLON will not permit retrieval of partial data files without first reading and discarding those data records located at the file beginning. Eventually, TABLON will permit storage and retrieval of logical file segments, but still on a file unit basis. TABLON is not a while-you-wait inquiry system; it is a digital data file (file segment) storage and retrieval network.

The attached computers

The attached computers, called User-CPU's, are, and will be typical stand-alone systems with their normal complement of peripherals. Security, and a reluctance to identify future procurements prevent the listing of actual computers by type and model. However, TABLON imposes no restrictions; thus any or all of the large scale third-generation systems are equally likely candidates.

A time-sharing computer, a batch processor, and an information storage and retrieval system, all of different manufacture, are already attached as User-CPU's. TABLON accommodates their different modes of operation equally, requiring only a software interface and communications package be added to the standard operating system of each.

The User-CPU's are not centrally located. Some are within a 3000 foot distance, located in the same building as the TABLON network. Others, a few miles away, are dependent on the "reliability" of public telephone circuits for long distance communications. This article will not discuss the methods, equipments, or problems associated with communications between the User-CPU locations and the central TABLON site.

TABLON data flow

Figure 1 shows a simplified diagram of the TABLON network. Its components will be referenced as we trace the storage and retrieval of a data file from a User-CPU to TABLON and back to the User-CPU.

A User-CPU has a data file it wishes to store into TABLON. The file is complete (all processing on it has terminated) and resides on a drum or disk within the User-CPU. A message is constructed by the User-CPU specifying parameters about the file. These include file name, classification, size, owner, who else can have access, plus an indicator as to whether the file is character based (and thus should be converted to ASCII) or binary (no conversion). This message is sent to TABLON where appropriate entries are made for the file in the index catalog. Concurrently, TABLON checks for availability of a 10^6 bit core Buffer and connects that Buffer to the requesting User-CPU. When cataloging is complete and the available Buffer connected, TABLON sends a message back to the User-CPU informing it that transfer of data may begin.

The User-CPU calls the first records from its drum or disk and writes these records out to the TABLON Buffer. At this point the Buffer is his alone, to be written at a one megabit per second rate, but at whatever size logical records and at whatever interrecord intervals are characteristic of the User-CPU. The Buffer remains attached to the User-CPU until it has transferred about 128,000 ASCII characters, or otherwise filled or finished with the 10^6 available bits. Termination of the Buffer load operation is signaled by a message from the User-CPU to TABLON, whereupon another Buffer may be made available to the User-CPU for transfer of the next file segment while TABLON processes the previous segment.

During the filling of Buffers by the User-CPU, the interface unit located in the User-CPU's area is making all necessary conversions to the ASCII character set and is padding, as necessary, each record of data such that each logical record falls on a Buffer word boundary. Both these operations will be undone when the data is later recalled for reading.

Once a Buffer has been filled by the User-CPU, the data is moved immediately into one of the mass storage systems. Since the characteristics of the IBM 1360 Photo-Digital system differ greatly from those of the AMPEX Terabit Memory system, the method of handling data from the Buffers is different. A technical summary of both these devices is provided later in this paper.

If data is destined for the AMPEX Terabit Memory (TBM*), the Buffer is switched to one of the TBM's two available write channels. The TABLON executive instructs the Terabit Memory to take the entire Buffer load and write it onto a specified block of high density magnetic tape. The TBM system then takes over and writes the Buffer load onto tape at a six megabit rate.

If data is destined for the IBM 1360 system, the entire file must first be accumulated on the PDP-10 disk packs. This file staging requirement is due to the photographic storage media used by the 1360 system and to the very slow writing rate (30,000 characters per second) of the 1360 recorder unit. To accomplish this, the Buffer is switched to a special data channel (NETIF) to permit reading the Buffer contents into PDP-10 memory, then out to the disk. The entire data file, Buffer load after Buffer load, is accumulated on the PDP-10 disk prior to sending any data to the 1360 storage system. By first staging the entire file on disk, TABLON is able to insure writing the full data file on the same or consecutive physical storage elements within the 1360. Intermediate data staging also enables a better packing of files to achieve the highest possible utilization of the storage media since once developed, the media cannot be added to later.

At the conclusion of file transfer between the User-CPU and TABLON, sign-off messages are exchanged and TABLON's catalogs updated. It is not necessary for data files to reach either storage system in order to be considered complete. Once all data is accumulated on the PDP-10 disk, it is considered entrusted to TABLON, and is available for immediate recall by the same or another User-CPU.

Retrieval of a data file is the reverse of storage. The User-CPU calls TABLON and provides the name and access authority for the file desired. TABLON verifies access and searches its index to find where the file has been stored. Once located, the mass storage device containing the file is instructed to access that file and prepare it for reading. Then, the storage device is connected to a Buffer and the first Buffer load of data transferred. The PDP-10 then connects the appropriate Buffer to the User-CPU's dedicated line and notifies the User-CPU of data availability. Reading by the User-CPU is begun, with the TABLON interface again converting (as necessary) from ASCII to the User-CPU's internal character set, and padding or omitting blanks to insure that each subsequent data record begins at a full word boundary within the User-CPU. The process is continued until the entire data file, or the specified number of file subsets, is passed.

Commonality across User-CPU's

One of the most interesting features of TABLON is its ability to permit retrieval of data by a User-CPU other than the one which stored the file; providing such capability presented a twofold problem. First, internal character sets used by different computers are not standard. Second, each computer assumes that successive records read enter on full word memory boundaries.

TABLON chose to implement hardware solutions to both problems, adding a feature to the TABLON interface unit called the Formatter/Converter. The Converter is a relatively simple device consisting of a read only memory, present to the internal character set of each type of User-CPU. At User-CPU option, data can be specified as binary (no ASCII conversion) or character, wherein the Converter will automatically do a read only table lookup and forward the ASCII configuration to TABLON for storage.

The Formatter portion of the Formatter/Converter

* AMPEX Corporation Trademark.

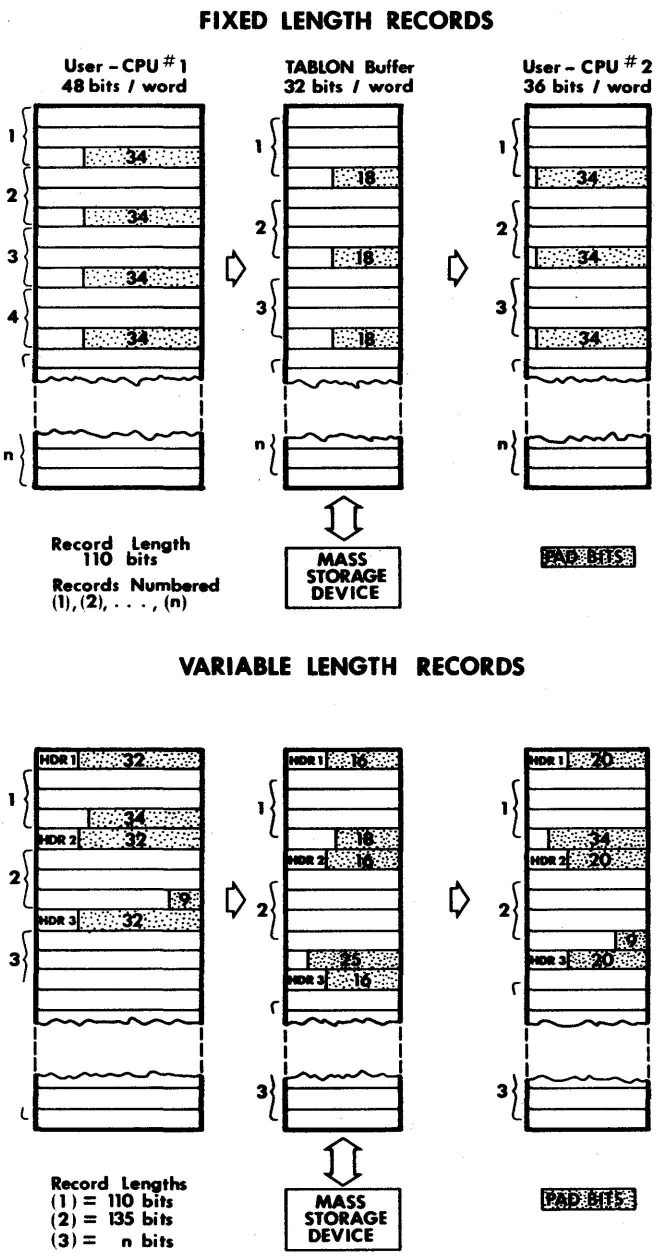


Figure 2—Formatter unit operation

unit is somewhat more complicated and is described more fully in the technical summary. Suffice it to say, the Formatter adjusts all input records from the User-CPU's into a standard TABLON length. During output back to the User-CPU's, excess bits are added on, or stripped off, so as to make the next sequential data record fall on a full word boundary in the User-CPU. This action is shown in Figure 2. No manual intervention is required in either case. Each interface unit is tailored to the type and model of User-CPU attached.

Anticipated performance—photo digital

The IBM General Purpose Simulation System (GPSS/360) was used to resolve design tradeoffs and simulate anticipated performance of the Photo Digital Systems within the TABLON System. Only anticipated performance figures are documented here.

A run was made which simulated 15 consecutive hours of operation, including 4 hours of time which had only one Photo Digital in operation, completing 858 jobs to and from TABLON. A "job" means a request at the User-CPU to store/fetch one complete file. The average job was completed in 5 minutes and 41 seconds, including the time (3 minutes and 4 seconds average) it waited in the User-CPU queue before actually getting to TABLON for processing. The range of job completion times shows 468 jobs completed in 30 seconds or less, another 72 within the next 30 seconds, 80 percent of the jobs within 5 minutes. There was no appreciable difference between the average time required to complete a read job and the time required to complete a write job.

The chance of a data file being able to go directly to TABLON without delay (when needed/furnished by a User-CPU) is over 85 percent. During the 15 simulated hours, one User-CPU had as many as 13 jobs within its own queue awaiting a turn for TABLON. The average backlog across all User-CPU's was 3.1 jobs, but the maximum did build as high as 17 at one time. Yet, there was only one job in the queue at the end of 15 hours. Over three billion characters were sent to and from TABLON during the 15 hours simulated, an average of 57,000 characters per second full time.

Jobs arrived at the User-CPU's for processing by TABLON on an average of one per minute. Any User-CPU was equally likely to get the next job, including the same User-CPU that processed the last job. The average file size was 3,360,000. Some files were as small as 2,000 characters and some as large as 100 million characters, with 95 percent of the files less than one full reel of magnetic tape (20 million characters).

*TABLON request language**

The TABLON Operating System has the fundamental task of translating high level file segment storage and retrieval requests into the detailed control sequences necessary to optimize data throughput time in a multi-User-CPU environment.

* The Language and Command names used here are not necessarily the ones that are used in the system, however, the functions are the same. Moreover, some of these functions will not be available when the system first becomes operational.

TABLON communicates with each User-CPU operating system, not with individual User programs. However, depending on the particular environment under which a User-CPU operates, an applications programmer may effectively be in direct control of his data transfer. Although the initial system will permit only one outstanding command per User-CPU, eventually TABLON will permit each User-CPU to have several data movement requests outstanding. Moreover, each of these requests may be associated with different files.

Storing data files (STORE)

To store a file, the User-CPU sends an initial STORE command to TABLON with parameters that identify the User, specify the FILE NAME, the mass memory system to use, and an estimate of the FILE SIZE. TABLON first ascertains that the request is from a valid User, that the storage allocation of the particular User will not be exceeded by this STORE, and that the FILE NAME is unique for this User. If the file is to be stored in the AMPEX Terabit Memory, a search of the MAP is made to locate the first contiguous storage area available that is just large enough to accommodate the data file. If the file is to be stored in the IBM 1360 System, a similar MAP search is made to determine which storage module will be selected for storage. Eventually, a catalog entry is constructed that will contain all the descriptors for this file. The actual transfer of the file then proceeds as described previously in this paper. Provisions have been made in the software structure to allow for non-contiguous file storage, clearly a necessity in the case of extremely large files.

Retrieving data files (FETCH)

Retrieval of the data files (or file segments) occurs in a similar fashion. The User may specify the starting point and length of data within a particular file to be transferred. For obvious throughput reasons, the system in general will not allow the User to FETCH less than one Buffer (10^6 bits) load unless, of course, the file length is less than 10^6 bits. On the other hand, random access allows the User to fetch only that portion of the data file he needs, thereby increasing job throughput at the User-CPU.

File sharing (PERMIT/REVOKE)

A series of DIRECTORY operations allows a file owner to permit other Users (independent of User-CPU's) access to one or more of his files. The permittee may be granted read only or read/modify access at the owner's option. Several authorized Users may have the same file open for reading simultaneously. The REVOKE command is used to terminate file access by one or more permittees.

File modification (APPEND/REPLACE)

The User may modify his file by either appending (APPEND function) or replacing (REPLACE function). The APPEND function is straightforward as long as provisions are made in the catalog entry to allow for non-contiguous segments. The REPLACE function is only meaningful in the TBM case and gives the User the ability to update a single block of this file. Obviously, only one User may have access when a file is open for modification.

File deletion (DELETE)

A file owner deletes one of his files by the execution of the DELETE command. In addition to making appropriate catalog modifications in the DELETE case, the SCP also modifies the MAP to make the area holding the deleted file available for storage of new data.

Reporting file activity

An important part of the design is the inclusion of "soft" meters to measure and record the file activity of User groups. These statistics are maintained in the appropriate catalog entries during the day and combined with the permanent statistical accounting file at the close of each day. As a consequence, managers have significantly more information on which to base their decisions than is possible in a manual tape library facility.

TECHNICAL DESCRIPTION

Figure 1 shows a simplified diagram of the TABLON system, consisting of the following major components; a pair of PDP-10 computers for overall control of the network and for management of the data base index; a Data Distribution Network used as the exchange mechanism for passing data between TABLON and the attached User-CPU's; a set of sophisticated interface units (one at each User-CPU site) for real time conversion to the ASCII character set and for the automatic formatting of data records across different

CPU word sizes; an AMPEX Terabit Memory for massive high speed storage of active data sets; and two IBM 1360 Photo-Digital storage systems for the archiving of "forever kept" data.

PDP-10 computer systems

A pair of interconnected PDP-10 processors are employed as controllers for the overall TABLON network. During normal operations, one PDP-10 is designated the master controller for the entire network, the other serves as controller to the pair of IBM 1360 storage systems. The master controller function includes communication with attached User-CPU's, directory management, and translating commands from the User-CPU's into action sequences to be executed by other network sub-components. The second PDP-10, acting as 1360 controller, performs all data handling and manipulation necessary between the Buffers and the 1360 storage units. This includes the scheduling of files into 1360 storage from the data staging disks and reporting back to data management final addresses of the stored file.

Redundancy and backup

Not shown in Figure 1 is the interconnection between PDP-10's. All twelve memory modules (196K words) are accessible by both processors. Also, each processor has its own I/O buss to which are attached those peripherals associated with its primary function. However, each processor has access via a programmable switch to the other's I/O buss, and thus all peripherals of the entire TABLON network. In this way, either PDP-10 processor can be the master, and can when necessary, perform the functions of both processors simultaneously. In either case, the processors themselves do not get involved with the handling of data *per se*, and therefore can continue responsive operation when controlling the entire TABLON network alone.

Software executive

Software for the PDP-10's was uniquely developed, consisting of a portion (15 percent) of the basic PDP-10 executive program, but mostly (85 percent) a special software package supplied through the talents of DoD. Major subroutines include the User-CPU communications module, commands and responses to the Data Distribution Network, directory management module, and the high level storage/retrieval commands to the 1360 controller (PDP-10) and the Terabit Memory controller (PDP-9). No less significant are software modules for statistics, on-line maintenance diagnostics, and off-line loading and unloading of data to and from magnetic tape.

Communications data multiplexor (CDM)

The CDM provides for concurrent transmission of variable length messages between the PDP-10 and each of the User-CPU's. Data flow over each line is bit serial at 4800 bps. To allow for maximum design flexibility, a PDP-8/L computer was used to implement the control logic. This approach minimizes disturbances to the PDP-10 by interrupting only when an entire message has been received rather than for each single character. It also permits the CDM to relieve the PDP-10 of considerable error analysis and to implement retransmission requests based on its own analysis.

Data distribution network

To attain respectable data throughput rates within the TABLON network, it is necessary to transmit or receive data from many User-CPU's simultaneously. Since this objective cannot be fully realized by buffering data transfers through the PDP-10 computer itself, a separate Data Distribution Network was necessary. The Data Distribution Network is composed of the following components:

Switch controller

The Switch Controller translates high level commands from the PDP-10's into a series of detailed hardware sequences. These commands take the form of Buffer to User-CPU or Buffer to Mass Storage Channel switching functions, Buffer clears, Buffer mode setting, and network status collection. Control lines between the Buffer and the Switch Controller permit appropriate initialization of the Buffer Logic, and provides an interrupt capability to the Switch Controller when the Buffer encounters an error condition. The Switch Controller also monitors key voltage levels in both the Switch and Buffer components in the idle state so that failures may be detected and corrected as early as possible. The PDP-10 receives a generated interrupt signal in such cases. The Switch Controller includes a PDP-9 for control logic implementation and easy communications with the PDP-10's.

Buffers

The Buffers are standard 32K word, 32 bits/word, 1 μsec core memory systems with special control logic added to permit the Buffer to operate as a slave to either a User-CPU interface or a Mass Storage Channel. Data is transferred between the Buffer and User-CPU at a 1 megabit serial rate, and between the Buffer and Mass Storage System at a 1 megabyte (8 bit byte) burst rate.

CPU/BUF and BUF/Mass storage switch

The CPU/BUF switch, under control of the Switch Controller, enables any User-CPU to communicate with any Buffer. Once a Buffer has been connected to a given User-CPU, no other User-CPU or Mass Storage System has access. The BUF/MSS switch, also under control of the Switch Controller, is very similar logically to the CPU/BUF switch. It allows any one of the Mass Storage Channels to communicate with any Buffer. Any Buffer not assigned is placed in a disconnect state.

User-CPU interfaces

Each User-CPU is supplied an interface unit which links the User-CPU via a 1 megabit dedicated line to the Data Distribution Network. Hardware wise, the User-CPU thinks it is communicating with a standard peripheral and to the extent possible, the same should be true for the software operating system. However, the TABLON concept is sufficiently revolutionary that software modifications at each User-CPU were unavoidable.

Formatter/converter units

The other fundamental functions of the User-CPU interface are logical record formatting and character code conversion. The Formatter/Converter unit within each interface performs these functions after being preconditioned by the User-CPU. Otherwise, the data is transferred as a binary stream only. The Formatter function is to force the first byte of each logical record to start at the beginning of a User-CPU word, even if the User-CPU is not the one that generated the file. Pad bits are added as necessary to achieve this result as shown in Figure 2 for both fixed and variable length records. The Converter function is applicable in the character transfer mode. In this mode, the standard User-CPU character (6, 7, or 8 bit) is converted to an ASCII code. Thus, all non-binary files in TABLON are stored in standard ASCII format.

AMPEX Terabit memory system

The AMPEX Terabit Memory (TBM) system uses the video technique of rotating heads and transverse recording to store large volumes of digital data on a very few reels of standard magnetic video tape. This system provides random access capability through extremely high speed searching (1000 ips) in combination with a reusable, update in place, magnetic storage media.

The Terabit Memory is being developed jointly by the Department of Defense and the AMPEX Corporation, under a task beginning in 1966. A small scale engineering model, containing at least one of each major TBM component has been constructed and is undergoing tests at AMPEX's Sunnyvale, California, plant. Concurrently, construction is under way for a trillion bit version to be delivered to the Government in early 1972.

When it arrives, the Terabit Memory system will provide the TABLON network with its most responsive storage element. Having at least two full duplex high speed data channels from the Data Distribution Network, Buffers may be emptied and filled by TBM at a combined rate of 24 megabits per second. As the active element, all files susceptible to change, all files anticipating updates, and all files of a non-archival nature will reside on Terabit Memory tapes. When established that a file has become stabilized and can be archived, it will be transferred to the 1360 storage system for permanent retention. The bulk of activity between TABLON and its attached User-CPU's will involve storage and retrieval of data files, or file segments, contained on the TBM storage device.

Terabit memory operation

The TBM system is capable of executing N+1 commands in parallel, where N is the number of transport controllers. The (N+1)st command is an immediate command such as ABORT, STOP, REQUEST STATUS, WHAT ABOUT, etc., that does not require a transport controller for its execution. The remaining commands are any combination of SEARCH, ERASE, READ, or WRITE operations.

Normally, a TBM search operation will be performed during the time a system Buffer is being loaded by a User-CPU. Upon completion of both of these functions and the switching of the Buffer to a TBM Write Channel, the PDP-10 forwards a WRITE command to the

TBM Controller specifying the transport controller, transport, channel, and block address to be used for the storage of that particular buffer load of data. Except in special test or maintenance modes, only one block of data is transferred for each READ or WRITE command. For the test, maintenance, and tape pre-test modes, a special exerciser (tester) is built into each data channel unit. In this mode, any number of blocks may be specified with each READ or WRITE operation.

Each TBM block consists of roughly 10^6 bits, the minimum amount of data that may be read or written. Approximately 167 msec is required to transmit that amount of data between the TBM and the Buffer.

The average demand access time to any TBM data block is of the order of 15 seconds. However, since most files will be contiguously stored, this time will be applicable only for the first block of a file. Less than 1 second is required to position the tape for reading of each of the remaining blocks in the file. This assumes, of course, that the transport is not needed for some higher priority data retrieval prior to completion of that particular file transfer.

The TBM system does not queue commands. By performing the queuing function in the PDP-10, TABLON retains the flexibility to effectively adapt itself to the various modes of User-CPU operation.

Technical summary of terabit memory

The Terabit Memory system is composed of four major components; the magnetic tape Transport, the Transport Controller, the Digital Data Channel, and the TBM Controller. The relation of these four major components to one another is shown in Figure 3. A summary level description follows; readers are referred to Reference 6 for further and more detailed information.

TBM Controller: Controlling the entire system is a PDP-9 computer called the TBM Controller. Its function is to provide the command interface between the PDP-10 and the various major components of TBM. Commands are received in the form of a single instruction consisting of an operation code (search, read, write, erase, etc.), a transport and block address, and the number of consecutive blocks to be operated on. Instructions are then issued to the Switches, Transport Controllers, and Digital Data Channels directly. Consecutive commands will be accepted by the TBM Controller and executed simultaneously wherever possible since operation of the major components are independent actions.

Transport Controller: Operation of each tape Transport is governed by an independent module called the Transport Controller. It receives packaged commands from the TBM Controller (PDP-9) and executes them against any one of the 64 maximum Transports. Under independent supervision of the Transport Controller, a Transport is issued instructions for prepositioning in anticipation of reading or writing. Servo and addressing electronics, plus a small control computer (NOVA), are used to implement the control functions. The NOVA translates high-level function commands (read, write, erase, search, etc.) into detailed control sequences as required.

Tape Transport: The function of the Transport is to move the tape at high speed (1000 ips) for searching, to accelerate and decelerate the tape, and to move the tape accurately during data transfer. A diagram of the Transport is shown in Figure 4. Only those functions necessary to move the tape (reels, capstans, vacuum chambers) and which cannot be provided by the Transport Controller or Data Channel are incorporated into the Transport.

Digital Data Channel: Once a tape is prepositioned, the Transport Controller and Digital Data Channel begin actual reading or writing of data from/to the video tape. On writing, the channel accepts a digital input which is then frequency-modulated and written in that form onto tape. On reading, the FM data is demodulated and reconstructed into digital format. Read and write channels are independent, permitting simultaneous reading and writing of data onto separate transports. At an 800 inch per second head speed, the data rate is six megabits per second per channel.

Switching Modules: The function of the switches is to attach any Transport Controller to any tape Transport, and to attach any Digital Data Channel to any tape Transport. Switch modules are housed physically at the base of each Transport module.

Tape Characteristics: The storage media is standard 10.5 inch reels of magnetic video recording tape. One reel contains about 48,000 inches of tape, providing 43,000 usable inches for writing data. At one million bits per linear inch (approx.) of tape, there are 43 billion (10^9) bits of data storage per tape reel. Useful life will be of the order of 1200 read or write passes to any block or its neighbor. Bad blocks can be deleted from service without affecting neighboring blocks.

Tape Format: The format of the tape consists of transverse data tracks containing the FM modulated signal, plus three digital tracks (Address, Control, and Tally) along the edges of the tape. The Control track contains one pulse per once-around of the recording head, the Address track contains block addresses 0 through 43,000, and the Tally track records the activity of each block such as number of accesses, last operation performed, and unique ID number. Each data track contains about 12,000 frequency-modulated bits. Tracks are 3.5 mils wide, separated by 1.8 mils, with 168 tracks containing 10^6 usable data bits in roughly one linear inch of tape. Each 10^6 data bits form one addressable block and represents the smallest segment of data which can be read or written in a single action. Since the length of each data block is determined by prerecording a longitudinal Address track during tape preparation, the amount of data per block could be varied without major change to the TBM system. Data blocks can be individually erased and rerecorded (erasing is prerequisite to recording) without affect on neighboring blocks. See Figure 5.

Recording Head to Tape Relation: A rotating video head drum with eight record/reproduce transducers performs contact across the digital data portion of tape. Head to tape rotation speed is 800 inches per second against a linear tape speed of five inches per

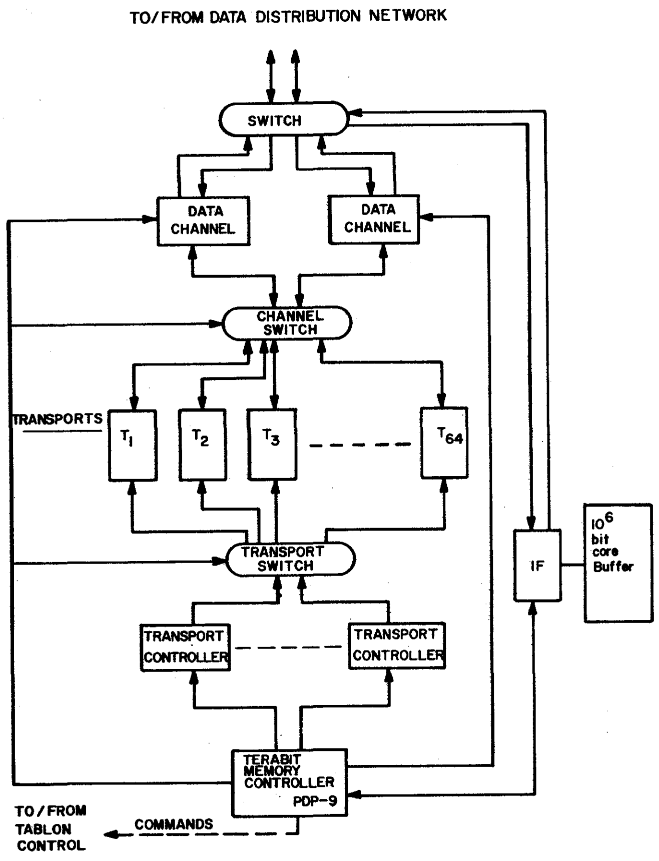
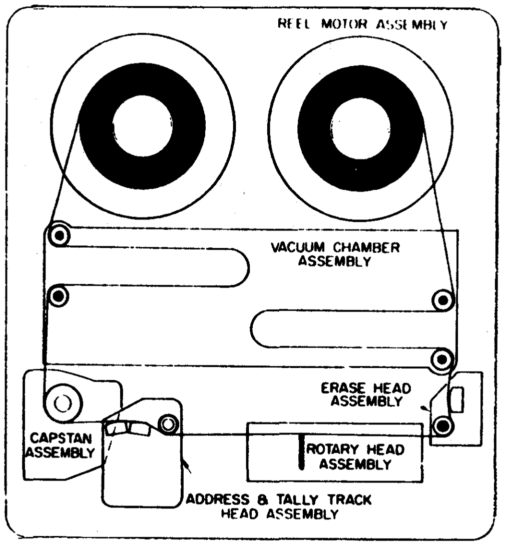
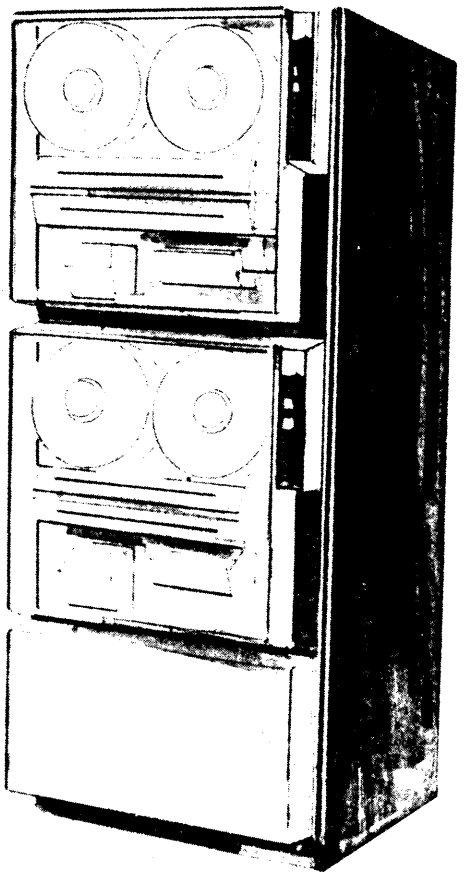


Figure 3—AMPEX terabit memory



TRANSPORT TOP PLATE



TRANSPORT MODULE

Figure 4—TBM top plate and transport module

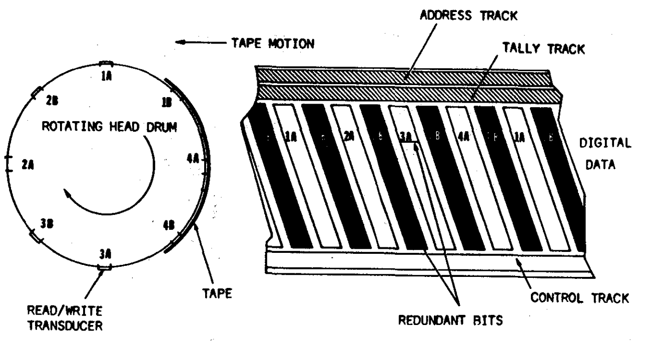


Figure 5—TBM tape format

second during reading and writing of data. This produces transverse recordings nearly perpendicular to the length of tape. The tape itself extends slightly more than 90 degrees of the circumference of the rotating head drum. With the eight heads separated by 45 degrees each, a minimum of two heads are in contact with the tape at all times. By providing the same signal to these same two heads simultaneously, all the information is recorded twice, i.e., 100 percent redundancy recording. This recording technique permits a packing density of 1.4 million bits per square inch total, or 700,000 effective data bits including redundancy.

Error Detection and Correction: After demarking (a tape pretest function) out all known areas of bad tape, error rates of less than 1 in 10^{10} data bits are expected. This performance will be achieved by 100 percent redundancy recording, plus an error detection and correction scheme described below.

A code word consists of 955 data bits and 37 EDC bits, for a total length of 992 bits. A single bit error within a code word (the most common TBM error) is detected and corrected with certainty. Beyond single bit error correction, the code is designed to detect, but not correct, all bursts with very high probability. For example, the probability that a burst error will occur when reading a TBM block is 2.54×10^{-4} (empirically determined). The probability that a burst will not be detected is 7.2×10^{-9}. The product of these two probabilities is 1.9×10^{-14} which represents the probability of an undetected burst error. Assuming correction of single bit errors and allowing 1 TBM block reread for every 2,000 BLOCKS read, a system error rate of better than 1 in 10^{12} is fully expected.

IBM 1360 photo-digital system

The two IBM 1360 storage systems provide TABLON with an extermely large volume storage reservoir for data files of lesser activity. The 1360 system uses photographic film (called the film CHIP) to store data in permanent, non-updatable fashion. Thirty-two film chips are housed in a small plastic box (called a CELL) and transported pneumatically between storage modules and read-write stations. Off-line storage of the film chips within their cell containers is also provided. For TABLON then, the 1360 systems will house the permanent, hopefully seldom updated, archival segment of the total data base. Files stored in the 1360's will be larger, their access slower (they may even be off-line), and their life more permanent than the active files contained within the AMPEX Terabit memory.

1360 system operation

As described earlier, the entire data file is assembled on the PDP-10 staging disks prior to entry to the 1360 storage systems. Writing to the 1360 is a block (18,450 8-bit characters) at a time from the PDP-10, forwarded in burst mode across the channel to the Data Controller. The channel is freed for reading between blocks. Concurrently, a frame of unexposed film is positioned for writing by the Recorder. The Data Controller feeds the entire block to the Record station where it is written directly onto the film chip with a modulated electron beam. This process is followed by automatic on-line development of the film chip, concluded by placing the fully dried chip into a waiting cell.

Cells remain at the Record/Develop Station until such time as the chip containing the last data segment arrives. At this point the cell is pneumatically removed from the Record/Develop Station and brought to one of the two Read Stations foı readback check. Using a special read-no-transfer function within the 1360, the entire data file is read with the PDP-10 being notified of any unrecoverable errors. Assuming none, this data file is then released from the PDP-10 staging disk. If errors are detected, then the PDP-10 must initiate a rewrite of the data to 1360 to insure a valid copy. Once determined that all data from the recorded film chip can be recovered, no further deterioration of data quality is expected.

Each Read Station permits the presence of two cells, thereby enabling exchange of chips between cells. On the completion of read-back verification, the chips involved with this latest file can be physically inserted to another cell if desired for a more logical storage arrangement. It should be restated that all decisions (and therefore instructions at the 1360) are generated by the PDP-10 as external controller to the 1360 systems. Each point to point movement of a cell, each chip pick, each read or write action, requires one or more individual commands from the PDP-10 to the 1360 system.

Technical summary of 1360 system

The following very brief technical summary is provided for the readers' immediate education. References 1 to 5 are recommended for further information on the IBM 1360 system:

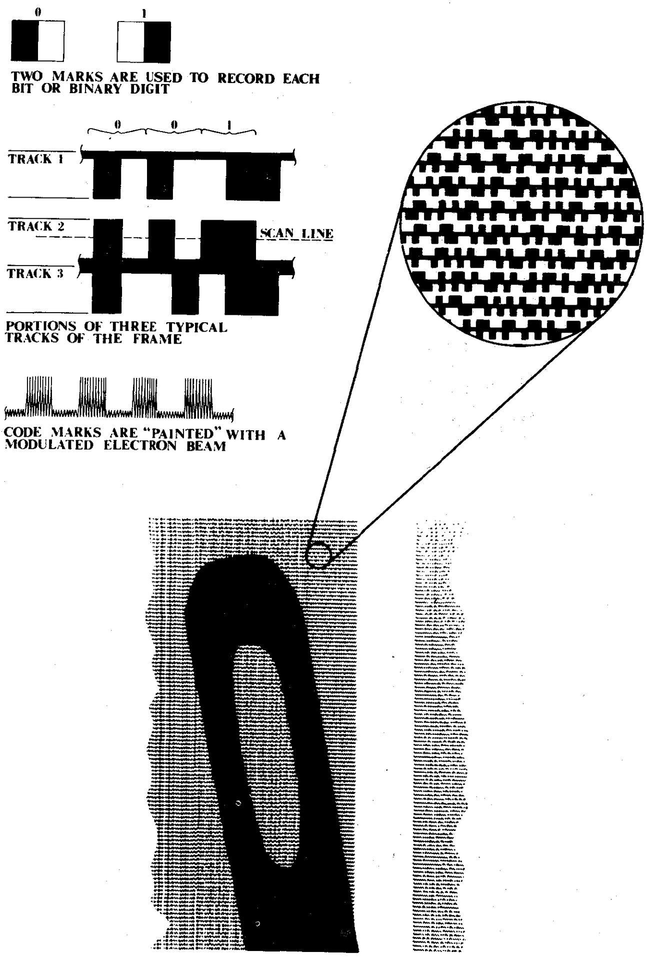
Film Chip: The smallest physical unit of storage in the 1360 system is the film chip, a fine grain silver halide emulsion, about 1.4×2.8 inches in size. Data is formatted on the chip in 32 frames, each containing 18,450 eight bit characters, arranged in 492 lines with 37½ characters per line. Each line contains 420 bits, of which 300 are data bits, 66 are error correction bits, 12 identify frame and line numbers, and the remainder are for synchronization. The coding of a bit is shown in Figure 6. Physical frame size is .277×.267 inches, containing 206,640 bits total (147,600 data only) for a density of 2×10^6 bits per square inch. Total chip capacity is 590,400 ASCII characters.

Storage Cell: The cell body is approximately 3.0× 1.6×1.1 inches, and contains 32 chips, each individually addressable. Figure 7 shows the orientation of the chips within the cell. The cell capacity with 32 film chips is 18,892,800 ASCII characters, or about the same as a fully packed reel of magnetic computer tape.

Control Processor: A stored program Processor is used to provide internal control of the 1360 system. This Processor operates at the selenoid-sensor level, determines the various functions which the system performs and provides diagnostic and recovery capabilities. Specifically, error detection is achieved by hardware, but error correction is implemented by a hybrid combination of hardware and Control Processor software.

File Modules: The basic File Module (the first module to be installed) stores 2250 cells. "Add-on" modules can be attached, each providing storage for 4500 additional cells. Cells are stored in individual compartments within movable trays; selection is achieved by moving the tray(s), to align bottomless compartments into a vertical shaft above the desired cell. Vacuum is applied and the selected cell drawn from its storage position. Access is a few seconds slower to "add-on" modules, but will be negligible during overlapped operation.

Record/Develop Station: This unit records the film, chip, then completely and automatically develops, fixes washes, and dries the film and returns it to a waiting


0
1
TWO MARKS ARE USED TO RECORD EACH BIT OR BINARY DIGIT
0
0
1
TRACK 1
TRACK 2
SCAN LINE
TRACK 3
PORTIONS OF THREE TYPICAL TRACKS OF THE FRAME
CODE MARKS ARE "PAINTED" WITH A MODULATED ELECTRON BEAM

IBM 1360 BIT CODING

Figure 6—IBM 1360 bit configuration

cell. The Recorder throughput rate is 30,000 characters per second, the recording cycle takes about 18.5 seconds per chip minimum. Development keeps pace with the recorder by processing chips simultaneously at each of eight developer stations. Complete processing of an individual chip takes about 180 seconds.

Reader Stations: The Reader opens the cell, extracts the addressed chip, and positions it in front of the cathode ray tube flying spot scanner. Associated electronics servo the flying spot onto a line of data and scan the recorded bit patterns. Information is scanned from the chip at an instantaneous rate of 2.5 megabits per second, with automatic line pair looping performed if the PDP-10 does not accept data promptly or if error correction procedures are in effect.

Data Controller: The Data Controller serves as inter-

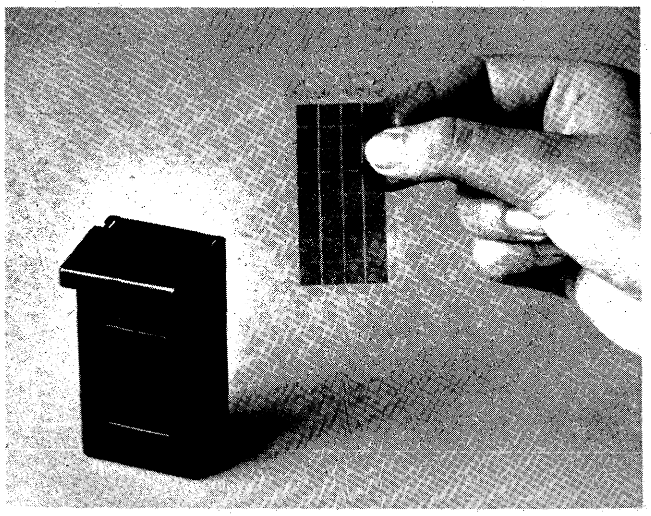


Figure 7—IBM 1360 storage cell and chip

face between the 1360 and the PDP-10, handling messages, responses, and data transmissions in both directions. It accumulates bursts of data from the PDP-10 until 18,450 characters (one full frame) have been received. The Data Controller then initiates the Recorder and forwards the entire frame of data for writing. On reading, the Data Controller accepts characters from the Reader Station and prepares them for transmission to the PDP-10. Error detection, and some correction procedures are also applied by the Data Controller if necessary.

Error Detection and Correction: The 1360 system uses a powerful error detection and correction technique. Information bits in each line are divided into six-bit segments; the code employed can correct errors in an five segments in the line of 378 bits. Multiple levels of correction are available, making use of the 1360 Control Processor if necessary to mathematically reconstruct the original data.

SUMMARY

The TABLON system is not a myth, or paper proposal. All of the hardware components are delivered and installed, except for the AMPEX Terabit memory system now under construction and scheduled for installation by early 1972. Work on hardware interfacing, and software development, is proceeding full bore with most User-CPU's having already exchanged test data with TABLON. Partial operation begins in early 1971, permitting exchange of data between on-line mass storage and standard 10.5 inch magnetic tape. By late 1971, additional User-CPU's will be attached and making daily use of TABLON for the storage and retrieval of their data files.

The TABLON network is a first approach, to be continually developed and refined as operational time is accumulated. There will be many problems in bringing a network of this size to operational status; TABLON will surely stumble, and perhaps even fall, before the payoff begins. However, the TABLON concept has definite wide ranging application. As newer mass storage devices are developed, they may internally resemble the TABLON organization. As long distance communications become more reliable, and data bases become larger at remote locations, the TABLON concept for shared access will prove both economical and feasible for other installations. In fact, shared storage in the massive capacity range may be the only answer.

ACKNOWLEDGMENTS

The authors wish to acknowledge the many outstanding technical contributions provided by others associated with the TABLON program; specifically the entire team dedicated to the design and implementation of the hardware and software systems, and those responsible for maintenance and reliability of the overall network. Acknowledgment is due also to the following contractors for their enthusiastic support: the AMPEX Corporation, the Digital Equipment Corporation, the IBM Corporation, and Westinghouse Electric.

REFERENCES

1 J D KUEHLER H R KERBY
A photo-digital mass storage system
Proceedings of the Fall Joint Computer Conference 1966 pp 735–742

2 R M FURMAN
IBM 1360 photo-digital storage system
Technical Report IBM Systems Development Division TR 02.427 May 1968

3 D P GUSTLIN D D PRENTICE
Dynamic recovery techniques guarantee system reliability
Proceedings of the Fall Joint Computer Conference 1968 pp 1389–1397

4 J B OLDHAM R T CHIEN D T TANG
Error detection and correction in a photo-digital memory system
IBM Journal of Research and Development Vol 12 No 6 pp 421–492 November 1968

5 R L GRIFFITH
Data recovery in a photo-digital storage system
IBM Journal of Research and Development Vol 13 No 4 July 1969

6 S DAMRON J R LUCAS J MILLER E SALBU M WILDMAN
A random access terabit magnetic memory
Proceedings of the Fall Joint Computer Conference 1968 pp 1381–1387

A structure for systems that plan abstractly

by WALTER JACOBS

The American University
Washington, D.C.

INTRODUCTION

The advantages of introducing planning procedures in systems that deal with complex tasks are generally recognized. Here the special benefits of using *abstraction* in the planning process will be considered, and a structure particularly well suited for abstract planning, the *purposive system*, will be described. An example will be presented to show how abstract planning operates under the purposive system; this is the program PERCY, which simulates the behavior of a nest-building insect.

There are superficial points of likeness between PERCY and L. Friedman's ADROIT (Friedman 1967, 1969). ADROIT is also a simulation of a nest-building insect, but PERCY does much more, using a considerably shorter and simpler program. The value of the purposive system structure is clearly shown by this example.

The concept of abstraction in planning is explained in the second section as the making of decisions without exploring the course of actions involved in executing these decisions. Instead of such exploration, a set of goals intermediate to the principal task are evaluated. The example of PERCY is described in the third section. It shows how the selection of intermediate goals can be separated from the translation into action of the corresponding decisions.

The fourth section discusses the hierarchical structure that makes the separation possible. The process of abstraction takes place in the course of communication between adjacent levels of the hierarchy. The operations of the purposive system are explained in the fifth section; they are few in number, simple, and independent of the task. The system's information about the task is contained in its *knowledge*, organized in a data structure of standard format. A summary describing the results achieved in the PERCY application is given in the concluding section.

DECISIONS, PLANNING AND ABSTRACTION

Abstraction in the sense used here is applicable to a system whose tasks require continual interaction with its environment. For such a system, the term *action* refers to an external commitment—an element of the system's observable behavior. A *decision*, on the other hand, is an internal operation. The term implies that a number of alternatives are evaluated and a selection is made. The decision then determines, directly or through some intervening process, what action or set of actions will follow.

It is useful to consider some examples. The nest-building insect ADROIT mentioned above provides an example where decisions as just defined are not made. Sequences of actions that accomplish desired objectives are controlled by a hierarchy of routines called Release Mechanisms and Selector Mechanisms. But these sequences are simply triggered by conditions in the environment; no evaluation of alternatives takes place.

In A. Samuel's checker-playing program (Samuel, 1959), and in game-playing programs generally, each move made by the program is an action. There is one decision for each action, and evaluation of the alternatives considered in reaching the decision is done by analyzing possible sequences of moves to follow the one being considered.

The SRI robot "Shakey" follows a pattern quite different from either of these (Nilsson, 1969). It uses a problem-solving approach, in which the entire sequence of actions to handle the task is worked out on an internally represented model of the environment before the first action is taken. Thus, there is only one decision per instance of a task, except where a mismatch between the environment and its model causes the planned solution to break down during execution. Again, as in the game-playing programs, analysis involves exploring possible sequences of actions.

The kind of decision making to be described does not

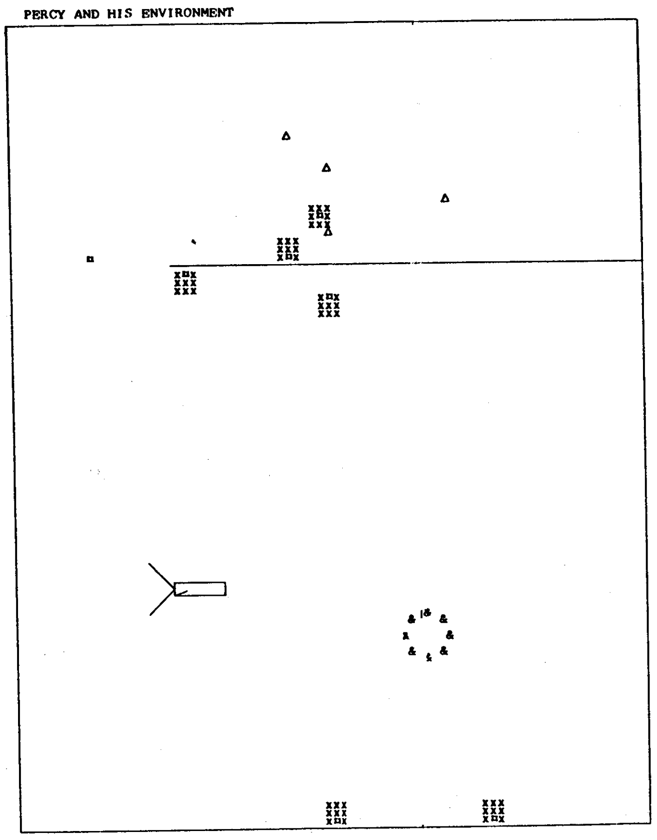


Figure 1

concern itself with the specific actions that may result from the decision that is selected. Rather, it considers intermediate *goals,* or way stations on the road to completion of the task. In *abstract planning,* a decision selects a goal among those that are available at the time the decision is faced, and this selection sets in motion a course of action that will arrive at some *outcome*—usually one of several that are possible. At this point a new decision must be made. Thus, there are a number of decisions taken in completing a task, and a very much larger number of actions.

Evaluation of goals is based on prior experience with similar decisions rather than on analysis of a course of action. The likelihood of each outcome is estimated, as well as the *satisfaction* to be gained in arriving at the outcome. Thus, an estimate of the expected satisfaction from selection of a goal is obtained, and these expectations are used in reaching a decision.

Past experience can be used because a decision remains in force till an outcome occurs. Which outcome takes place depends on what is met in the environment. In abstract planning, there is an assumption that the probabilities are stable.

Thus, abstract planning frees the system from the burden of analyzing the actual process of trying to realize a desired outcome. It must be used when the system is not able to explore in an internal model the explicit consequences of a course of action.

It is worth emphasizing that in most man-machine systems, abstract planning is the man's role in the process. The introduction of it into machine systems is a step—though far from the final one—toward having these systems handle their tasks in a way that deserves to be called "intelligent."

PERCY—AN EXAMPLE OF AN ABSTRACT PLANNING SYSTEM

The environment in which PERCY operates is pictured in Figure 1. There is a nest location, with eight individual sites at which material is to be placed in order to complete the nest-building task. There are a number of clumps of material (indicated by x's), a part of the upper area where food (small triangles) is hunted, and some obstacles to be avoided—in particular, the wall that separates the upper and lower areas. In addition, there are landmarks (small squares) which are visible at a distance, and are used by PERCY in reaching spots from which it can see its main targets.

PERCY's task (there is only one in the present version) is to build its nest. To do this calls for eight trips. On each trip, one piece of material is located, picked up, carried back to the nest, and placed at an empty site. On some of the trips, PERCY will also get food.

PERCY uses abstract planning for its trips, and makes decisions that try to maintain a high level of satisfaction. But its satisfaction will drop if it devotes too small a part of its time to the search for food, and gets hungry as a result. Satisfaction goes down also when it spends too much time getting food, and progress on the nest lags. Thus, in order to have good performance, the decisions it faces at various stages of the task must bring about a proper allocation of its time between getting food and building its nest. PERCY must rely on abstract planning because it has no internal model of the environment on which to explore in detail the consequences of a course of action.

Its observable behavior consists of a series of elementary actions. Each of these is produced in a *cycle of behavior,* which takes up a brief moment of PERCY's lifetime. A cycle begins with the receipt of a stimulus from the environment, continues by determining the desired response, and ends with the action that forms the response.

In the nest-building task, PERCY uses only five elementary actions. They are:

(i) a short movement in the direction of perceived target,
(ii) a short side-step to avoid an obstacle,
(iii) a turn through a limited angle,
(iv) a movement partly extending its arm,
(v) a movement partly retracting its arm.

To accomplish even a limited objective, a number of these elementary actions must be strung together. To reach an outcome of a selected goal may require a large number of such actions in proper sequence, all initiated by a single decision. An example of such a sequence from a run of PERCY's task will be given shortly.

Decisions are called for only at certain stages of PERCY's task. One such stage occurs when PERCY has just completed a trip by adding material to the nest, and is about to start another trip. At this point, it must decide whether food or material should be sought first. A principal factor in this decision is the amount of time that has elapsed since it last had food.

Assigning priority to food accounts for one of the goals open at this stage. There are three others, for if PERCY wants to look for material first, there are three sections of the environment it may head for: the upper area shown in Figure 1, and the lower and upper sections of the other area. (PERCY is unable to distinguish between the two landmarks in the same section; it simply heads for the first landmark it sees in the chosen section.)

To specify the goal, it is sufficient to name the classes of targets that are of interest during the time that goal is pursued, and to identify which of them is the principal target. For example, among the four goals just described, the one that gives priority to food is specified as follows:

Food is the principal target, the door landmark and material are alternate targets.

The "door landmark" is the one at the entrance to the upper area; it must be used because PERCY cannot see any target when the wall is in between.

Each goal has several outcomes. The one just described has three outcomes possible: Food will be obtained, or material will be spotted on the way, or something "illegal" may take place. This last outcome occurs when the environment fails to behave as called for in PERCY's program. For example, if the door landmark cannot be located, PERCY will go on searching till it reaches a point where it gives up the task. If no way out of the impasse were available, PERCY would continue to search until it "died."

PERCY uses prior experience to estimate, for each goal considered, the probability that each outcome will occur and the length of time that will be needed to reach that outcome if it does occur. Taking account of its current state of hunger and progress on the nest, together with the estimated time that will elapse, it further estimates what its satisfaction will be should that outcome happen. Applying its probability estimates, it then arrives at an estimate of expected satisfaction for the goal being evaluated. Finally, it decides on the goal with the highest expectation.

Nowhere in this process of evaluation does it give consideration to the actions that will follow its decision. Its evaluations deal only with measures that are quite detached from its environmental interactions. In spite of this, its decisions give rise to coherent and purposeful series of actions that can go on for many cycles of behavior before a new decision is required.

Figure 2 (showing the output of the program as plotted on an SC 4060 printer) depicts the greater part of one of PERCY's trips. PERCY's movements during this trip are marked by a trail of dots, which show the successive positions of PERCY's turning center. The interval between two dots when PERCY is headed toward a target indicates how far it moves in a single action. The direction in which it is facing is shown by an

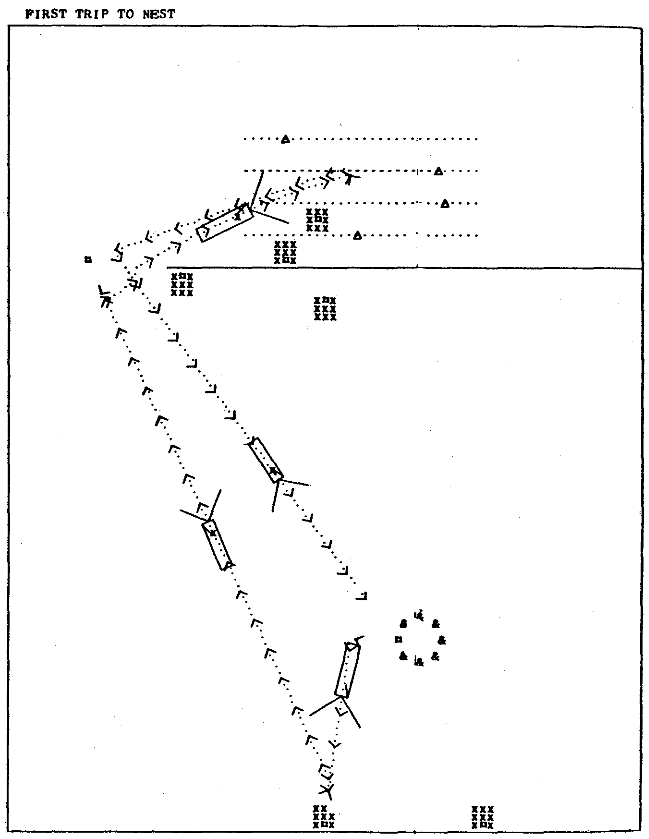


Figure 2

arrow, every fifth cycle of behavior. When arrows are shown without four dots intervening, PERCY is turning, or, in one case, standing and eating food.

Only three decisions were involved in this series, involving more than 300 individual actions by the time the trip was finished. About half of this total derived from the decision to seek food after material was picked up at the location shown in the figure.

This single food-hunting sequence thus took nearly 150 cycles. It was made up of four parts, in each of which PERCY was executing a limited objective, or *subtask*, contributing to the desired goal. The parts, and their corresponding objectives, were as follows:

(i) A series of turns, searching for a target of interest, and ending when the door landmark was seen.
(ii) A series of moves in the direction of that landmark, aiming to get near it, and ending when the objective was attained.
(iii) Another series of turns, searching for food, and ending when food was spotted.
(iv) A series of moves in pursuit of the food, that brought PERCY within arm's length of it; a sidestep to avoid the wall occurred at one point. When PERCY was near the food, it extended its arm in a series of movements, seized and ate the food, and returned its arm to the rest position.

This behavior was generated in face of an environment that is known only in the most limited way, and sensed very crudely. For example, PERCY has only the simplest ability to discriminate distances; it uses only the three categories "far," "near" and "at arm's length." The only information it has about the location of key features of its environment is contained in the way it associates landmarks with its goals.

It clearly takes a program of some complexity to translate a single goal into a cogent course of action in the face of an environment that is so poorly known. The next two sections describe how the purposive system makes it possible to reuse the major part of that program for other goals, and thus to handle a complex task successfully with a program of moderate size. For further information about PERCY, see Jacobs, 1971.

THE HIERARCHY OF ABSTRACT PLANNING

The ability to realize an economical system for abstract planning with a variety of goals depends critically on the way programs for individual goals are combined. The purposive system uses a hierarchical organization to achieve this economy. Further, the structure of the system is independent of the nature of the tasks handled. This is why the greater part of the program is common not only to different goals, but also to distinct tasks, as long as they use the same set of perceptions and actions.

Thus, it would take a relatively modest enlargement of the present PERCY program to add an exploration task. This would precede the nest-building, and would find a suitable place for a nest, and also pick the landmarks to be used in the later task. If other bugs were added to PERCY's environment, courting and fighting tasks could be readily introduced.

The hierarchy in the purposive system has four levels, corresponding to four components in a physical realization of the system. These are called the *body, task, method* and *strategy* components.

The body component forms the system's interface with the environment. It contains the sensory and motor apparatus; thus it receives the stimuli that originate in the environment, and carries out the actions that respond to these stimuli.

The task component controls the operation of the body component. It coordinates sequences of actions into subtasks, such as those described in Section 3. It frees the higher components from any concern with the details of environmental interaction.

The method component guides the task component through the succession of subtasks appropriate for a given goal. Since conditions in the environment determine how a subtask will wind up, management at this level requires more than a linear sequencing of subtasks. The conversion of a decision into its subtasks is expressed as a *submethod,* which resembles a subtask in structure.

The strategy component evaluates and selects goals. Since the other components take complete responsibility for execution of its decisions, this component is concerned only with the outcomes of its decisions.

There is an obvious analogy in this hierarchy with the organization of a large enterprise. The strategy component corresponds to the top executive level, while the body component acts like the mass of employees who deal with the suppliers and customers. The other two components correspond to the intervening levels of management.

Figure 3 names the communications that are passed in support of this hierarchy. (In general, the received form of the message is distinguished from the issued form.) The way these communications are handled is important in allowing the system to operate simply and economically. The only communications that occur internally are the following:

(i) Once in every cycle of behavior, the body component reports a *perception* to the task component. In return it receives an *intention*. This specifies the action to be performed, and also specifies the kinds of targets that are of interest in forming the perception for the next cycle.
(ii) When the subtask reaches a conclusion, the task component reports the *situation* that has been reached to the method component. That component issues a *plan* that specifies the next subtask to be performed, and identifies the kinds of targets involved.
(iii) When the submethod finally arrives at an outcome, the plan in effect is communicated to the strategy component. That component sends back its goal, which sets the next submethod in motion.

Under this scheme of operation, the body component is continually active in receiving stimuli from the environment and issuing responses. At successively higher levels, each component is involved less frequently than its predecessor, and is more detached from what is going on in the environment. The term "abstract" planning is therefore an apt description of the way the system controls its behavior.

The upward and downward flows in Figure 3 may be found suggestive of the afferent and efferent flows in the Central Nervous System of a living creature. There, too, a decision can be turned over to lower levels of the CNS for execution without demanding the conscious attention of the higher brain centers. This analogy will be pursued elsewhere, for it can be shown that the structure and operations of the purposive system have implications for a theory of the functioning of the Central Nervous System in cognitive activity.

THE OPERATION OF THE PURPOSIVE SYSTEM

The components of the purposive system function as finite automata. In this way the structure takes maximum advantage of the use of abstraction. The inputs and outputs of each component are simply the communications that have been described, together with external stimuli and responses of the body component. The state variable of the component provides the task context, appropriately abstracted, for processing the communications most simply and efficiently.

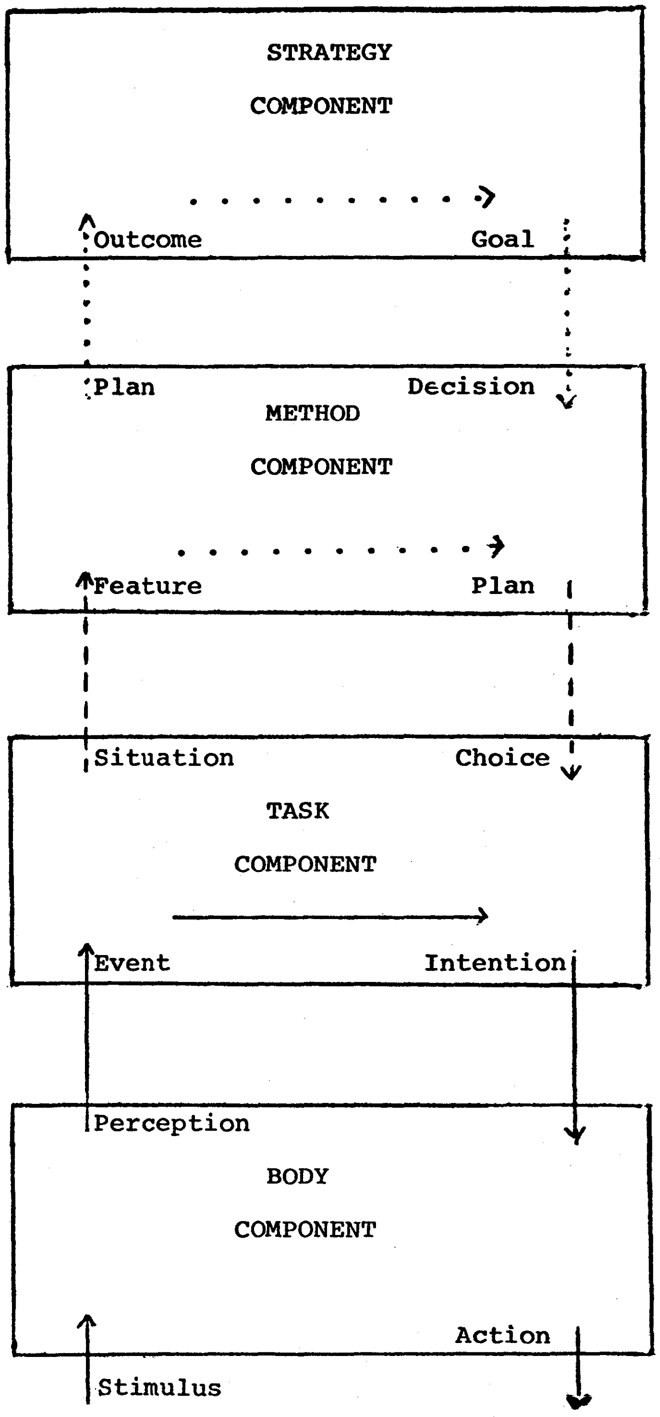


Figure 3—Components and flow of communication in a purposive system

Except in the body component, each state variable acts as a pointer to an entry in a list structure. This structure is called the *knowledge* of the system. It contains, in a general format, all information specific to the task. The knowledge, as acted on by the general operations of the system, simply expresses the two functions—the state transition function and the output function—that describe the component as a finite automation.

This is a very efficient arrangement. Its efficiency is enhanced by introducing, as parameters of the knowledge of subtasks and submethods, the specific objectives of a class of goals. Then a single entry in knowledge can apply to a whole group of state variables that differ only in their parameters. This makes it possible to execute a variety of subgoals with a relatively small number of subtasks and submethods. Here again, the structure is integrally bound up with the process of abstraction.

An example from PERCY's program may help to illustrate these points. The following stage of the task is assumed to have been reached: PERCY is hunting food, has not yet picked up material on the current trip, and has the door landmark in sight. The system's state (not including the knowledge of the task) is given by four variables:

(i) The goal (state of the strategy component) indicates that food is the principal target; the alternate targets are the door landmark and any item of material.
(ii) The plan (state of the method component) indicates which subtask is in progress: i.e., going to a sighted landmark. Its parameters are the identities of the landmark and of the alternate targets (food and material) whose sighting would also bring the subtask to an end.
(iii) The situation (state of the task component) specifies the stage reached in the subtask: The landmark is visible but not near, and neither alternate target nor obstacle is in sight.
(iv) The perception (state of the body component) records that the principal target is in sight but still "far away." No other item of interest is in sight. The arm is in the rest position, and nothing is held.

It has been mentioned that the first three of these states act as pointers to entries in knowledge. For example, the entry that is pointed to by the situation (iii) just described contains three items of information:

(i) The intention, or output of the task component. The intention in this case specifies the action of moving toward the principal target, and also identifies the types of the principal and alternate targets.
(ii) The criteria by means of which the set of inputs —i.e., the possible perceptions—is partitioned into *events*. The events are the distinct meanings that the next reported perception can have in this situation. They include the following: "no change," "principal target near," "alternate target sighted," "obstacle in way," and "target not seen."
(iii) For each event, the new situation that points to the knowledge entry needed should that event be *recognized*. For example, if the event is "no change," the new situation is the same as its predecessor; if the event is "alternate target sighted," the new situation is the one that marks the end of the subtask, with the particular target sighted as parameter.

With this format, only the simplest and most general operations are needed in order to make the task component function as a finite automaton and, in so doing, carry out its role in the production of behavior. These operations include receiving the perception from the task component, applying the criteria in knowledge to recognize the event, replacing the situation by the new one for that event, and issuing the output located in the knowledge entry for the new situation. Except when the subtask is done, this output is the intention and it goes to the body component.

If the subtask is done, the output of the new situation is the situation descriptor itself, and it goes to the method component. In that case, the method component will return a plan. The plan is recognized as a *choice*, and this input to the task component will cause a second change of situation in the same cycle. This second situation will issue an intention to begin the new subtask. The process has already been shown schematically in Figure 3.

The same operations, using the same format of knowledge, take place in the method component when a subtask ends. In this case the input from the task component is the situation, and its recognized form is the *feature*. After the new plan replaces its predecessor, the output to the task component is that plan. And when the submethod ends, the plan is reported to the strategy component. The goal that is received in return is recognized as the decision, which then causes another change of state to the initial plan of the next submethod.

When a submethod comes to an end and an input to the strategy component takes place, it is recognized as an outcome of the goal that has been in effect. However, a new operation—evaluation—takes place at this level in arriving at the new state of the component, and an appropriate modification of the structure of knowledge is necessary.

A block diagram summarizing the operation of the purposive system is shown in Figure 4. The RECOGNIZE operation carries out the task-independent steps just described: It applies the criteria listed in the knowledge entry to the other input, selects the listed alternative that best matches that input, and issues the corresponding output. This points to the knowledge entry that replaces the one just used, and activates the communication needed for the next step of the process.

RESULTS OF THE APPROACH

A summary of the results accomplished on PERCY's task is given here as a basis for assessing the potential of abstract planning. That task was chosen as one requiring an appreciable number of nonroutine decisions, producing activity of long duration without human intervention. (Subject to this, the example was constructed with an eye to minimizing the labor of simulating an environment and the necessary interactions with it.)

In the process of completing its nest, PERCY used its five types of subtasks about 80 times. They required more than 1900 individual actions. The course taken by each subtask or action depended generally on the particular targets of the current goal as well as on the external conditions that were encountered.

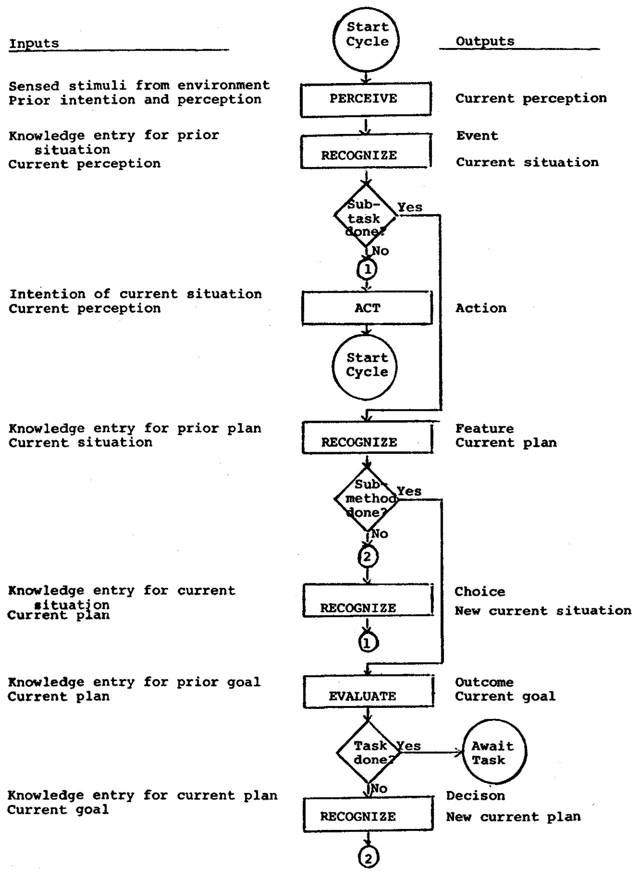


Figure 4—Block diagram of purposive system

Number of Trip	Outcome	Description of Outcome		Number of Cycles
1	1	Material taken at	(205, 15)	33
	2	Food eaten at	(228,410)	147
	3	Material placed at	(246,120)	124
2	1	Material taken at	(115,335)	68
	2	Material placed at	(250,130)	68
3	1	Material taken at	(205,320)	56
	2	Food eaten at	(189,370)	75
	3	Material placed at	(260,134)	117
4	1	Material taken at	(115,340)	78
	2	Material placed at	(270,130)	81
5	1	Material taken at	(110,340)	87
	2	Food eaten at	(217,390)	60
	3	Material placed at	(274,120)	136
6	1	Material taken at	(205,320)	69
	2	Material placed at	(270,110)	78
7	1	Material taken at	(110,335)	75
	2	Food eaten at	(234,370)	64
	3	Material placed at	(260,106)	152
8	1	Material taken at	(200,320)	65
	2	Food eaten at	(276,390)	98
	3	Material placed at	(250,110)	171
		Task completed		1902

Figure 5—Sequence of outcomes in a single run of PERCY's nest-building task.

PERCY had to make successful decisions in order to show good performance. No constraints were imposed on these decisions to keep PERCY from choosing unsatisfactory courses of action. Further, it operated with only the most summary memory of what happened in past decisions, and with almost no information about environmental conditions on which the quality of its decisions depended. It had no way of exploring the future beyond the point at which the next decision would be made.

In spite of these handicaps, PERCY was able to perform in impressively successful fashion. Figure 5 shows the actual outcomes of its series of decisions. The coordinates of the places where material and food were taken, and the nest locations where material was placed, are used to indicate the outcomes. The behavior in arriving at these outcomes was in every case as direct, purposeful and non-tentative as in the example of Figure 2. On the other hand, there was no mechanical or trivial pattern to this behavior.

PERCY did not spend an excessive amount of time going after food, which was hunted in only five of the eight trips. It also avoided the opposite error of subjecting itself to severe hunger. To an observer, every one of its many actions would appear meaningful in relation to its task.

The structure of the purposive system made it possible to realize these results with a comparatively simple program. Written in Fortran (not the most efficient language for the purpose), the program contained about 600 instructions, although no special effort was made to hold down this total. The complete run on a third-generation computer took about 3 seconds of CPU time.

The problem space in which PERCY operated was larger by many orders of magnitude than any that have been successfully tackled by a general problem-solving system. PERCY's accomplishments are thus a clear demonstration of the power of the abstract planning approach. Clearly, the way people handle most of their tasks is much closer to this pattern than it is to other existing general-purpose systems for problem solving, which use extensive exploration of a detailed internal representation of the problem.

In pointing this out, there is no intention to claim that the present approach deserves to be described as "intelligent." It seems safe to assert that in order to merit that characterization, a system must possess both problem-solving and abstract planning capabilities, and other capabilities as well. Nevertheless, it is believed that for many kinds of tasks, the abstract planning approach will turn out to be an efficient, economical way to deal with them.

BIBLIOGRAPHY

L FRIEDMAN
Instinctive behavior and its computer synthesis
Behavioral Science Vol 12 No 2

———
Robot control strategy
Proceedings of the International Joint Conference on Artificial Intelligence.
Bedford Mass The Mitre Corp 1969

W JACOBS
Help stamp out programming
Theoretical Approaches to Non-Numerical Problem Solving
Banerji and Mesarovic ed Berlin Springer-Verlag 1970

———
How a bug's mind works
Proceedings American Society for Cybernetics Fourth Symposium
To appear 1971

N NILSSON
A mobile automation
Proceedings of the International Joint Conference on Artificial Intelligence
Bedford Mass The Mitre Corp 1969

A SAMUEL
Some studies in machine learning using the game of checkers
IBM Journal of Research and Development
Vol 3 No 3 1959

Unconventional superspeed computer systems

by TIEN CHI CHEN

IBM San Jose Research Laboratory
San Jose, California

INTRODUCTION

As machine systems grow in complexity to provide ever higher quality and quantity of service, increasing attention has to be paid to aspects of computing which are not the requirements for problem solving, but rather demands imposed by the handling technique prescribed for earlier machines!

A study of the self-optimizing capabilities of large computing systems suggests that unconventional systems can be conceived which can gain efficiency, adapt to human usage patterns, and be consistent with future hardware promises.

SUPERMACHINE SYSTEMS

When hardware (and to an extent software) investment for a given system exceeds a certain critical size, facilities can be provided for self-optimization. The system no longer needs to process the workload passively, exactly as externally prescribed; it becomes a complex of autonomous units, each actively manipulating resources under its own jurisdiction. We shall designate such a large, self-optimized complex a "supermachine system." Self-optimization will be shown to be a practical necessity for throughput efficiency; conventional processing, by the same token, will be seen as a handicap.

PARALLELISM, PIPELINING AND EFFICIENCY

An important reason for large systems is throughput enhancement. Within a given technology, there are two basic techniques to achieve this end: (a) quantitative split of workload, that is, parallelism by identical processors, and (b) qualitative division of labor, an extreme case of which is pipelining.

Employing parallelism, a multiprocessor of multiplicity M is made to sweep over job requirements to achieve complete coverage in space-time. The job to be processed has a time-dependent natural width function $w(t)$. For convenience of analysis we assume $w(t)$ to have only two values 1 and W, with time expenditures t_1 and t_2 respectively. Jobs with other $w(t)$ can usually be recast in terms of this standardized model (Figure 1).

The parallelism ratio ρ can be defined by

$$\rho = (\text{space-time of the part of job with width } W)/(\text{space-time of job}) = Wt_2/(t_1+Wt_2)$$

which is a property of the job. The multiprocessor needs to make only one pass to cover the narrow portion of the job and n passes to cover the wide portion, n = the integer part of $[(w+M-1)/M]$. The efficiency of coverage is

$$\eta = (\text{space-time of job})/(\text{space-time swept by multiprocessor}) = 1/M[1-(1-n/W)\rho].$$

as shown in Figure 2. If W is not exactly divisible by M, there will be a residual inefficiency even when $\rho=1$.

η becomes $1/k$ when $1-\rho=(kW/M-n)/(W-n)$. Figure 3 shows that, for $W=32$ and $n=1$, even if M exactly matches W, η falls to 50 percent, 20 percent and 10 percent while ρ is 96.8 percent, 87.1 percent, and 71.0 percent respectively; and for $W=100=M$, $\eta=10$ percent when ρ remains as high as 90.9 percent. Further, no matter how large M is, the effective multiplicity ηM can never exceed $1/(1-\rho)$.

Instead of many identical general processors, the division of labor approach employs a collection of nonidentical, specialized mechanical operators functioning simultaneously to form a general facility. An extreme form of this is pipelining.

A simple pipeline is constructed by first analyzing a given computing process into a linear sequence of

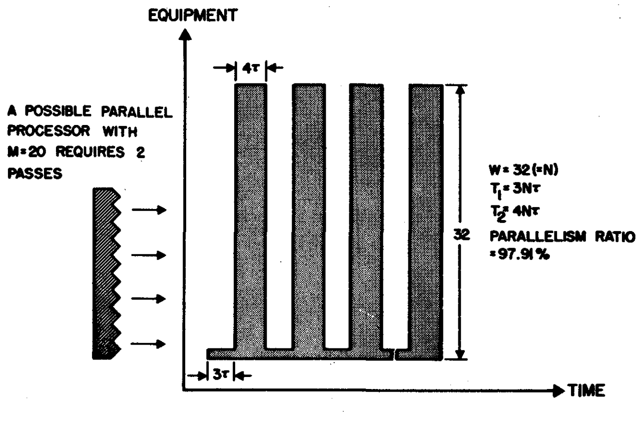

EQUIPMENT
A POSSIBLE PARALLEL PROCESSOR WITH M=20 REQUIRES 2 PASSES
4τ
3τ
32
W=32(=N)
T₁=3Nτ
T₂=4Nτ
PARALLELISM RATIO =97.91%
TIME

Figure 1—Job profile for the inner loop of a 32×32 matrix multiplication

microprocedure segments, each of the latter is then assigned an independent operator which completes its work in exactly one time-quantum τ, or a cycle. Each job will have turnaround time of $S\tau$, but the steady-state pipeline throughput is $S/S\tau = 1/\tau$, independent of S (Figure 4). Given sufficient work-load, the overall throughput can approach this value also.

The space-time diagram of such a pipeline with a jobstream of length L is shown in Figure 5, and is equivalent to a standard parallelism configuration with $M=W=S$, $n=1$ and $\rho=1-1/L$, hence $\eta=1/[1+(S-1)/L]$.

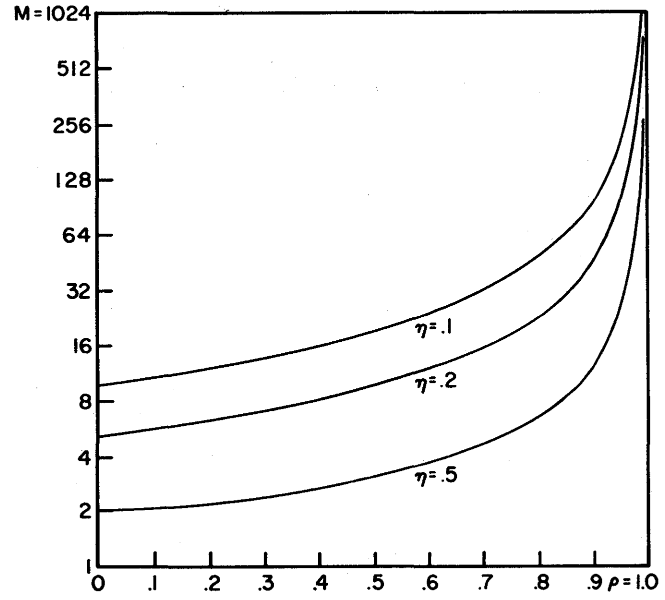


Figure 3—Efficiency curves for the case W=M

Therefore, though details differ considerably, essentially the same efficiency worries plague the parallel and simply pipelined systems. Realistic superspeed systems often involve networks of unevenly quantized pipelines and nonuniform parallelism. The job stream may not possess ideal multiprocessing characteristics, and the efficiency becomes even more elusive. Self-optimization then becomes a critically needed feature, to allocate available resources to maximize throughput, yet preserving the correctness of results.

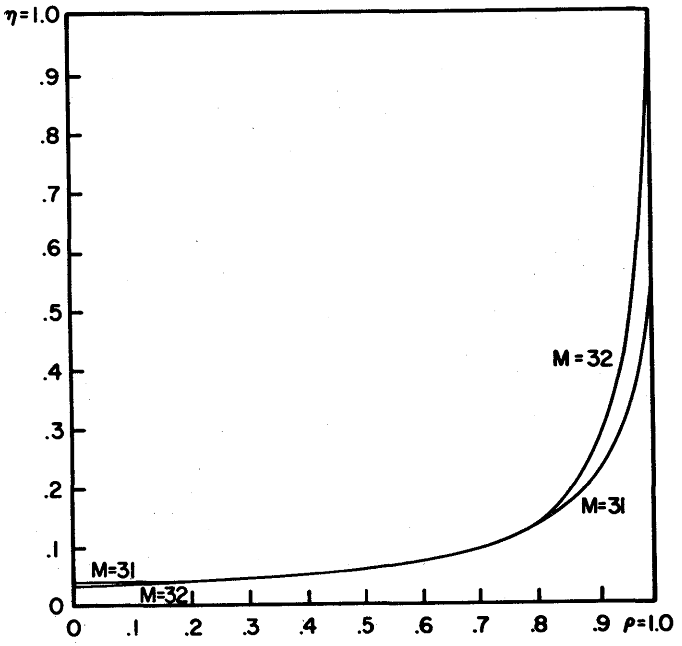


Figure 2—Efficiency (η) versus parallelism ration (ρ) for W=32
If M=31=W-1, η=0.52 when ρ=1.0

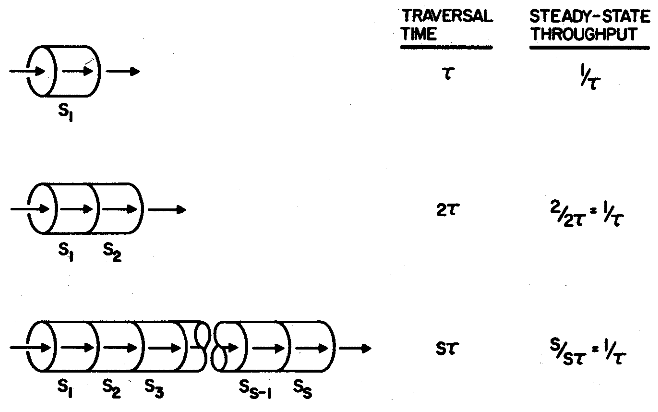


Figure 4—The invariance of steady-state throughput of a linear pipeline

SELF-OPTIMIZATION

There are two main categories of optimization:

(a) Procedure redefinition
(b) Resequencing of the procedure collection.

Redefinition calls for much perception and a global comprehension, and tends to fall in the domain of *external optimization*, by human or compiler, although the IBM System/360, Model 90 series show a modest ability.[2]

Resequencing of procedures is a chief ingredient of multiprogramming.[3] By handling many procedures together, the total equipment needed for adequate throughput will approximate the sum of the average (rather than the peak) requirements. Coincidental peak requirements often can be resolved, and under-usage of equipment avoided, by permuting the workload sequence.

To be truly effective, this freedom to deploy available resources should extend down to each autonomous unit, within the scope of which the input work stream(s) can be analyzed into a number of causally independent streamlets, behaving like requirements from different users. Available resources are then assigned to enhance average progress without completely ignoring any one streamlet. Thus, the supermachine system is expected to practice not only multiprogramming in the usual sense of the word, but also multiprogramming in the small, or "micro-multiprogramming" (Figure 6). The workload characteristic to be exploited is job independence, rather than the much more restricted job parallelism.

The control technique is generalized table management. Each unit will alter linkages, recode, reclassify, change priorities, resolve conflicts, handle errors and anticipate future events. The hardware requirement consists mainly of memories for waiting streamlets and

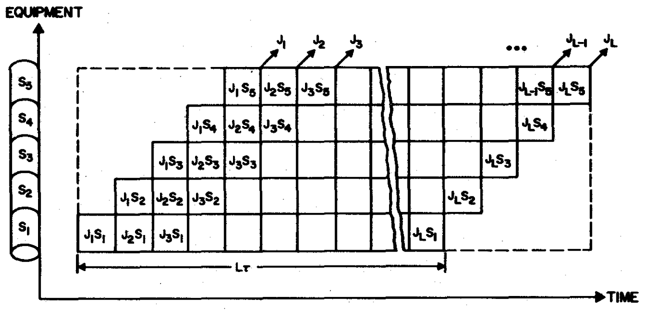


Figure 5—Profile of a stream of L jobs through a 5-segment pipeline. If L = 32, parallelism ratio = 96.25 percent and efficiency = 88.89 percent

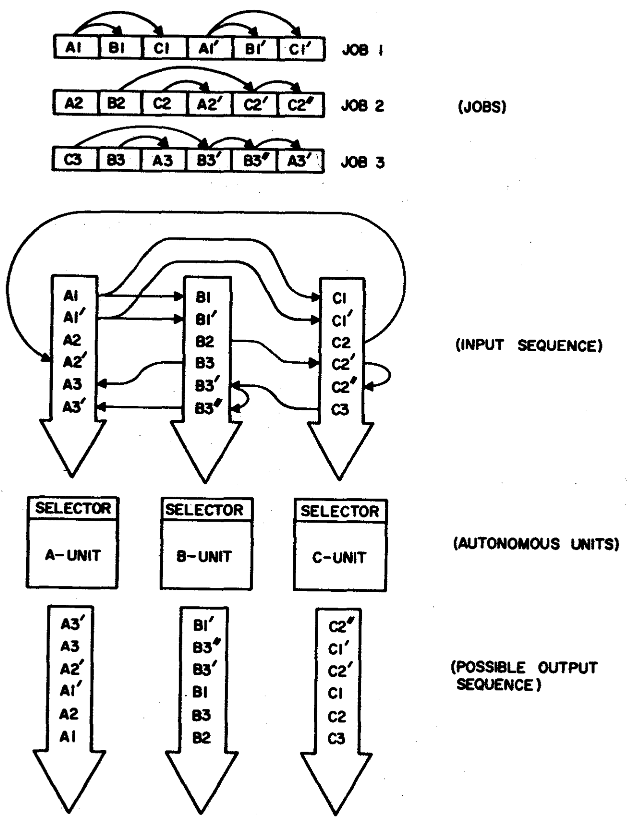


Figure 6—Micro-multiprogramming

encoded signals, and capabilities for selection and switching; in other words associative memory copackaged with standard logical hardware. Such a distributed intelligence is costly with current technology, but can exploit LSI for its speed, economy, compactness, also its penchant for regularity.

CONVENTIONAL PROCESSING

To achieve effective self-optimization, the system should be granted maximum freedom to redefine, or resequence the work within its power. This implies *minimum overspecification* by external agents. The real-time happenings, because of multiprogramming, overlapped I/O, interruption features and conditional branches, are virtually impossible to anticipate. Detailed specifications, made without the intimate knowledge of real-time events, tend to handicap the self-optimization.

Most problem programs today are written in procedural languages. These programs are pre-processed by a compiler into machine instructions, to be subsequently handled explicitly by the machine. The compiled machine codes historically were the only messages

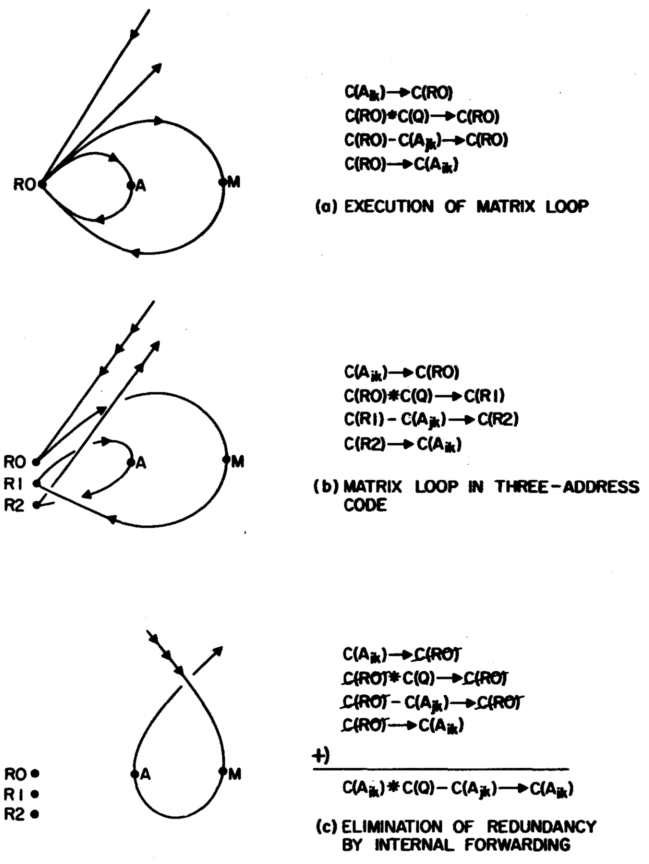


Figure 7—Processing of multioperator statements

understood by the earlier machines, and are still well-suited for most machines today; yet they contain the excessive tactical detail which condemns the supermachine system to inefficiency.

PROCESSING MULTI-OPERATOR STATEMENTS

Consider the following FORTRAN innerloop, important in solving simultaneous equations:

```
    DO 300  K=J, 100
300  A(I, K)=A(I, K)*Q-A(J, K)
```

During compiling, statement 300 is recast into instructions. For one- and two-address schemes, the arithmetic part reads as follows.

$$\begin{aligned} C(A_{ik}) &\rightarrow C(R0)\\ C(R0)*C(Q) &\rightarrow C(R0)\\ C(R0)-C(A_{jk}) &\rightarrow C(R0)\\ C(R0) &\rightarrow C(A_{ik}) \end{aligned}$$

Pipelined arithmetic operators M, A can be installed for multiplication and addition, each with a throughput of one result per cycle. Therefore, one could rightfully expect an overall throughput of one matrix element (A_{ik}) per cycle. However, the instruction processing stream crosses upon itself three times at $R0$, thus slowing the flow to $\frac{1}{6}$ the expected maximum rate (Figure 7a).

The use of a three-address code

$$\begin{aligned} C(A_{ik}) &\rightarrow C(R0)\\ C(R0)*C(Q) &\rightarrow C(R1)\\ C(R1)-C(A_{jk}) &\rightarrow C(R2)\\ C(R2) &\rightarrow C(A_{ik}) \end{aligned}$$

results in the diagram in Figure 7b. There is no pipeline crossing, and unit throughput can be maintained. However, the redundant use of the three registers is apparent, and the pipeline is lengthened unnecessarily; also three-address codes usually consume more program space.

A still better way is to apply surgery on the instruction sequence to eliminate the redundant paths, resulting in the clean pipeline in Figure 7c.

$$\begin{aligned} C(A_{ik}) &\rightarrow C(R0)\\ C(R0)*C(Q) &\rightarrow C(R0)\\ C(R0)-C(A_{jk}) &\rightarrow C(R0)\\ C(R0) &\rightarrow C(A_{ik})\\ +) \quad & \overline{}\\ C(A_{ik})*C(Q)-C(A_{jk}) &\rightarrow C(A_{ik}) \end{aligned}$$

This technique, employed in real time, is found in the S/360 Model 90 series of computers.[2,3] It re-creates the original string of operations. Clearly, hardware would be saved and compiling simplified if the fragmentation into instructions *never had taken place.*

ARRAY PROCESSING

The program in the last section involves the systematic handling of vectors. In routine computing involving arrays, the system is only given piecemeal information about the individual array elements, making it hard to divine the intent of array processing. Consequently, undue burden is placed on the decoding mechanism and the memory access hardware, and arithmetic pipelines tend to run at fraction capacity. There is the extra cost to perform index arithmetic to locate the array elements one at a time, and execution of branch instructions to 'close the loop.'

Concise descriptions of array procedures already exist in APL/360[5] and to a lesser extent in PL/1, and are easy to use. Should these specifications remain unaltered by the act of compiling, the supermachine can be designed to mobilize its resources in real time.

Memory data can be accessed *en masse.* Arithmetic operators can be reserved, and pre-configured into appropriate pipelines. Work areas can be created. I/O devices can be synchronized. Competing programs can be downgraded in priority. In general, near-peak efficiency can be achieved for the self-optimized supermachine system.

NAME HANDLING

During compiling, the names in the procedural language program are mapped into addresses.

Various relocation schemes further map the addresses into some other addresses. "Paging"[6] and "cache"[7] mechanisms perform dynamic mappings in real time, based on information not available to the compiler, and have tremendous performance implications to large machines. The initial mapping by the compiler, not capable of replacing paging and caching, is at least partly redundant.

Most procedural languages today use names; the latter's value to the human user lies not just in the mnemonic value, but in the freedom to associate. A name is, so to speak, a *universal reference-quantum.* Names are used to designate words, arrays, bits, character strings, and structures. They also designate branch targets, subprograms, and aspects of the operating system. They may designate other names, emptiness (null) or a state of incomplete definition. One entity may have several names, or several items may have the same name, the ambiguity being resolvable by context.

Direct handling of names by the machine not only removes a redundancy; its implication to the human user will be profound. During debugging, the correlation between the machine code and human procedural program will be more easily understood, and there is the possibility of following and manipulating associations efficiently during computation.

The handling of array names will permit the real-time assignment of space to fit the needs exactly. The programmer need not specify array dimensions, and indeed can alter them during computation. Already this freedom is being exploited to great advantage in APL/360 and PL/1. The rigid dimensioning of arrays is proving to be a fetter for both man and machine.

MACHINE "*M*" FOR MULTI-OPERATOR STATEMENTS

The unconventional supermachine systems are indicated in Figure 8. One can consider first Machine *M*, which uses (variable length) multi-operator statements,

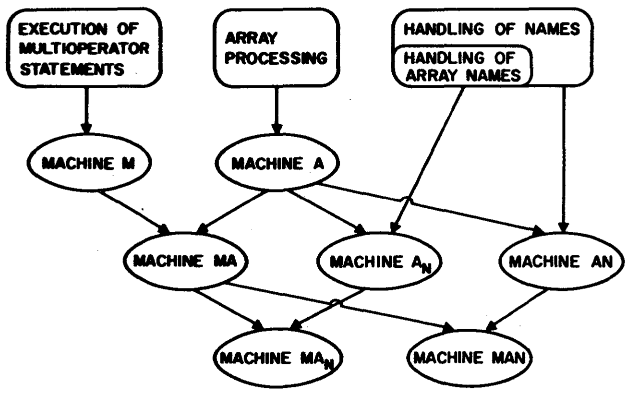


Figure 8—Unconventional machines

possibly with rigid field formats and conventional addressing. Decoding will become more orderly, and the intermediate register assignment will be entirely up to the supermachine system in real time.

As each statement describes a causal chain of events, at any time normally only one member of the chain can be handled. The unfinished statement segment can be put in a waiting state; other statements bearing no causal interlock can be processed simultaneously. The hardware to detect and resolve logical conflicts, and to redefine procedures (such as that used for internal-forwarding[2]) will be simplified.

MACHINE "*A*" FOR ARRAY PROCESSING

Array processing is one of the most-discussed unconventional machine features, but is usually presented outside the context of general purpose computing. Machine *A* is meant for array processing consistent with multiprogramming through self-optimization. It would use special array instructions, possibly created from a special compiler. There is, however, no need to prevent the system from handling its regular computing load.

When arrays *A*, *B*, interact to produce array *C*, elements within each array often are not causally linked, and thus can operate independently to fill an arithmetic pipeline. Array processing is then just a special form of multiprogramming, in which the 'jobs' are extremely well-defined, and are known in advance to be independent.

In Machine *A*, arithmetic for large arrays can be assigned a high priority and behaves like, say, up to 16 programs in parallel. Other jobs are given lower priority to ensure non-conflict. The handling, however, must be such that all jobs are processed sooner or later. Since array processing does not tax the decoding mechanism too heavily, it is actually desirable to mix-in conventional jobs which are decode-limited.

Both statement processing and array processing enhance performance and are mutually consistent. It is possible to consider them together, to yield Machine "*MA*." The format incompatibility can be resolved by adopting statements to array processing also.

We observe that procedural languages like APL/360 and PL/1 already have array facilities, and can be used as a basis. These languages also have a distinct name-orientation which seems particularly advantageous for arrays.

NAME HANDLING MACHINES

To handle names directly in all ramifications, would seem to require significant overhead beyond those needed for multi-operator statements, array processing and standard self-optimization. This is because associated with each name is information concerning attributes and whereabouts of the named object. Although the handling technique is again table management, consistent with self-optimization needs, nevertheless, the resources needed will not be trivial, and except for special purpose tasks (such as text handling, list processing), name handling should combine with statement processing and array handling.

MACHINES "A_N" AND "MA_N" WITH ARRAY NAMING

Machine A_N handles only those names affecting arrays, and is organized like Machine *A*. MA_N is similarly Machine *MA* with array naming.

The simplest way to achieve array naming is to compile array names into indirect addresses, and a dynamic array management system is triggered on each referral. This is reminescent of interpretive matrix arithmetic software which dynamically manages a shared memory pool.[8] During computation both the array characterization and array contents can change.

As arrays increase in size, the name-handling overhead rapidly becomes insignificant relative to compute cost. It is interesting that, at the same time, dynamic memory management is truly meaningful with array naming, and is most beneficial when storage space is at a premium, namely when arrays are large. The hardware realization of A_N and MA_N will further allow much smaller arrays to be efficiently treated under extensive multiprogramming.

The concatenation of vectors to form a longer vector will be automatically accomplished; when these vectors are character strings, this facility will have a clear impact on the automatic handling of procedure language statements and textual material.

MACHINE "*MAN*"

The above technique can be extended to all types of namable information, but the overhead cost would be relatively large if the associated data processing time is small. Machine *MAN* attempts to pay an overhead, in order to reduce the overhead due to conventional compiling. It is a fully interpretive machine.

Here a procedure language is executed more or less directly. The system will have powerful facilities both for scanning for delimiters and for I/O conversion.

The choice of procedural language will have a strong bearing on machine efficiency. The APL/360 language seems closest to the ideal, because of its conciseness, array orientation, excellent non-numeric handling capabilities with which most other procedural languages can be simulated. Even the strict (right to left) execution sequence is an asset for hardware implementation.

Once the procedural language has been decided upon, an important machine-suitable canonical subset can be selected for hardware implementation. Canonical statements will be handled efficiently. More complex statements can be treated by slower means, which definitely includes software assistance.

One can, therefore, write a code only the innerloop of which is written in canonical form, gaining both efficiency and convenience. Or, the user can submit a non-canonical code, and during execution a canonical version can be created and used. This is analogous to the 'loop mode' feature in the System/360 Model 90 series, where the first traversal of the loop lays the groundwork for the subsequent efficient computation. The distinction between compiling and interpretation tends to disappear.

THE HUMANIZATION OF THE COMPUTER

The concept of computer instructions is a quarter-century old, and compilers have been in use for more than a decade. Both have a genuine reason for their invention, and are still playing a vital role in computing today.

Their introduction represented giant strides toward humanizing the computer, but not necessarily the ultimate step.

Within the computer complex, it has become known that the most valuable resource is the human user. Into the global cost-effectiveness equation must be factored his throughput, turnaround time, endurance limit and ability to learn. These are tied in with his physiological makeup and behavioral patterns which have withstood the test of millenia.

Circuitry and memory costs represent but a small function of the total cost of a computer installation.[9] The projected sharp drop due to large scale integration, when applied to conventional designs, may mean a small lowering of the user's total cost for the same performance level. By the same token, the doubling or even quadrupling of circuitry and memory would represent a minor increase of total cost. Yet the added hardware, if placed conventionally, will tend not to have too significant an effect on systems whose throughput bottlenecks lie in the costly I/O and auxiliary memory equipment.

Unconventional designs, on the other hand, can enhance both internal supermachine efficiency and human effectiveness. Each design represents an aspect of "language-directed computer design"[10] as seen from a self-optimizing supermachine viewpoint, consistent with the advance of future hardware, and made possible by the recent advances in the codification of software techniques. The future compiler or prepossessor, freed of the drudgery of detailed tactical machine code specifications, can now emphasize the strategic deployment of resources towards global optimization of throughput performance.

REFERENCES

1 T C CHEN
Parallelism, pipelining, and computer efficiency
Computer Design Vol 10 pp 69-74 1971

2 T C CHEN
The overlap design of the IBM System/360 Model 92 central processing unit
Proc AFIPS Fall Joint Computer Conference 1964 Part II pp 73-83

3 D W ANDERSON et al
Machine philosophy and instruction handling
IBM J Res Dev Volume 11 pp 8-24 1967
R M TOMASULO
An efficient algorithm for exploiting multiple arithmetic units
Ibid pp 25-33

4 E F CODD
Multiprogramming
Advances in Computers Volume 3 pp 78-155 1962

5 A D FALKOFF K E IVERSON
The APL/360 terminal system
Interactive systems for experimental applied mathematics
Proc of ACM Symposium Washington DC 1967 M Klerer and J Reinfelds editors Academic Press New York 1968 pp 22-37

6 B RANDALL C J KUEHNER
Dynamic storage allocation systems
Comm ACM Volume 11 pp 297-305 1968

7 J S LIPTAY
Structural aspects of the System/360 Model 85. II: the Cache
IBM Systems J Volume 7 pp 15-21 1968
C J CONTI
Concepts for buffer storage
IEEE Computer Group News Volume 2 No 8 pp 9-13 1969

8 F H BRANIN et al
An interpretive program for matrix arithmetic
IBM Systems J Volume 4 pp 2-24 1965

9 M G SMITH W A NOTZ
Large-scale integration from the user's point of view
Proc AFIPS Fall Joint Computer Conference 1967 pp 87-94

10 W M McKEEMAN
Language-directed computer design
Proc AFIPS Fall Joint Computer Conference 1967 pp 413-417
C McFARLAND
A language-oriented computer design
Proc AFIPS Fall Joint Computer Conference 1970 pp 629-640

High speed division for binary computers

by H. LING

International Business Machines Corporation
San Jose, California

INTRODUCTION

Beyond the steps of SHIFT and SUBTRACT, division used in early machines generally relied upon Newton's method.[7,10] In order to increase the speed of division the following two approaches, namely the *S-R-T* recoding method[5,9,10,12] and the iterative multiplication scheme,[1,3,13] have resulted. The S-R-T-method offering an average shifting distance of 2.6 bits[4] is very close to the scheme used in Stretch;[2] the iterative multiplication scheme providing a quadratic convergence rate with preselected starting block has been used in IBM 360/91. However, the shifting distance of the S-R-T method varies with the format of the denominator; the required number of multiplication and the starting table of the quadratic convergence scheme need further improvement. This paper is concerned about the second approach, i.e., the iterative multiplication scheme. Recently the author[8] has developed a method in which one of the iterative multiplication is replaced by a fixed point subtraction. In this paper a better method is developed, this new method not only eliminates one of the iterative multiplication but also reduces the size of the starting table. The detailed derivation of the algorithm will be presented first, followed by a detailed description of the implementation procedure along with two examples. The comparison of the proposed division scheme with the others are also enclosed.

THE ALGORITHM

In binary division the quotient Q can be described as

$$Q=2^{E_x-E_y}q \qquad (1)$$

where E_x, E_y are the exponent of the numerator and the denominator

$$q=n/\delta$$

n and δ are the mantissa of the numerator and denominator

The normalized denominator δ can be written as

$$\delta=0\cdot 1\delta_2\delta_3\delta_4\cdots\delta_n.$$

Let us rewrite q into the following

$$q=\frac{n}{0\cdot 1\delta_2\delta_3\cdots\delta_L+(2^{-L})0\cdot\delta_{L+1}\delta_{L+2}\cdots\delta_n} \qquad (2)$$

and let

$$k=\frac{1}{2(0\cdot 1\delta_2\delta_3\cdots\delta_L)} \qquad (3)$$

Where L represents the first leading L-bit of the denominator. By substituting equation (3) into equation (2) we obtain

$$q=\frac{2nk}{1+k(2^{-L+1})(0\cdot\delta_{L+1}\cdots\delta_n)} \qquad (4)$$

Before further expansion of equation (4), the value of L is to be decided first. The choice of L is limited by either the size of the table where the value of k is kept, or the maximum number of fan-in and fan-out when the value of k is to be logically implemented. However, the size of the table and the maximum number of fan-in and fan-out can be slightly reduced due to the fact the leading bit of the normalized denominator is always equal to one, and the last bit of the L-leading bit can always be set to zero (if it is not so) by modifying the equation (4). Therefore, the size of the k-table is equal to $2^{(L-2)}$, or the maximum number of fan-in is equal to $2^{(L-2)}$, or the maximum number of fan-in is equal to $L-2$. Under these constraints L is chosen to be 8.

Let us rewrite equation (2) into the following

$$q=\frac{n}{0\cdot 1\delta_2\delta_3\delta_4\delta_5\delta_6\delta_7+2^{-7}\delta_8+2^{-8}0\cdot\delta_9\delta_{10}\cdots\delta_n} \qquad (5)$$

with slight algebraic manipulation, equation (5) can be written as follows

$$q=\frac{n}{0\cdot d_1d_2d_3d_4d_5d_6d_7+2^{-8}0\cdot d_9d_{10}\cdots d_n} \tag{6}$$

where

$$d_1=\delta_2\delta_3\delta_4\delta_5\delta_6\delta_7\delta_8$$

$$d_2=\delta_8[\delta_2(\delta_3'+\delta_4'+\delta_5'+\delta_6'+\delta_7') +\delta_2'(\delta_3+\delta_4+\delta_5+\delta_6+\delta_7)]+\delta_8'\delta_2$$

$$d_3=\delta_8[\delta_3(\delta_4'+\delta_5'+\delta_6+\delta_7')+\delta_3'(\delta_4+\delta_5+\delta_6+\delta_7)] +\delta_8'\delta_3$$

$$d_4=\delta_8[\delta_4(\delta_5'+\delta_6')+\delta_5\delta_6(\delta_4+\delta_7)]+\delta_8'\delta_4$$

$$d_5=\delta_8\{\delta_5[\delta_6'(\delta_2+\delta_3)+\delta_7']+\delta_5[\delta_6\delta_7(\delta_2+\delta_3)] +\delta_2'\delta_3'\delta_6\delta_7\}+\delta_8'\delta_5$$

$$d_6=\delta_8[\delta_6\oplus\delta_7]+\delta_8'\delta_6$$

$$d_7=\delta_7'\delta_8+\delta_7\delta_8'=[\delta_7\oplus\delta_8] \tag{7}$$

$$d_9d_{10}\cdots d_n=\delta_8'(\delta_9\delta_{10}\cdots\delta_n)+\delta_8(\delta_9'\delta_{10}'\cdots\delta_n')$$

Substituting equation (7) into equation (3) and (4), we obtain

$$k=\frac{1}{(0\cdot d_1d_2d_3d_4d_5d_6d_7)2} \tag{8}$$

and

$$q=\frac{2nk}{1\pm k(2^{-7}0\cdot d_9d_{10}\cdots d_n)} \tag{9}$$

When δ_8 equals to one, the denominator of equation (9) equal to $1+k(2^{-7}0\cdot d_9d_{10}\cdots d_n)$; when δ_8 equals to zero, the denominator of equation (9) is chosen to be $1-k(2^{-7}0\cdot d_9d_{10}\cdots d_n)$.

In order to converge $[1/1\pm k(2^{-7}0\cdot d_9d_{10}\cdots d_n)]$ rapidly, the term S_p is so defined to control the significant digit by equation (10)

$$S_p=\tfrac{3}{2}(2S_{p-1})-f(2S_{p-1}) \tag{10}$$

where

$$S_1=\tfrac{3}{2}D-f(D) \tag{11}$$

D is the denominator of q of equation (9) and $f(D)=(D/2)(D+1)$.

It can be seen from equation (11) that if D has L-leading ones, or L-leading zeros after the one following the binary point, $2S_1$ will have at least $2L$ leading ones. Since $1\pm k(2^{-7}0\cdot d_9d_{10}\cdots d_n)$ will either have 8 leading ones or 8 leading zeros after the one following the binary point, $2S_1$ will have at least 16 leading ones, and $2S_2$ will have at least 32 leading ones. Substituting $P=2$ into equation (10), we have

$$S_2=\tfrac{3}{2}(2S_1)-f(2S_1) \tag{12}$$

Dividing both sides of equation (12) by D, equation (12) becomes

$$\frac{S_2}{D}=\frac{3}{2}\left(\frac{2S_2}{D}\right)-\left(\frac{2S_1}{D}\right)\left(\frac{1+2S_1}{2}\right)$$

$$=\left(1-\frac{D}{2}\right)(1+D'^2) \tag{13}$$

or

$$\frac{2S_2}{D}=(2-D)(1+D'^2) \tag{14}$$

Generally speaking 32-bit precision is sufficient for single precision division, by substituting equation (14) into equation (9) we finally obtain

$$q=2nk(2-D)(1+D'^2) \tag{15}$$

or

$$Q=2^{E_x-E_y}2nk(2-D)(1+D'^2) \tag{16}$$

Equation (15) shows that if there are two multipliers available, a 32-bit precision quotient can be obtained with three multiplications.

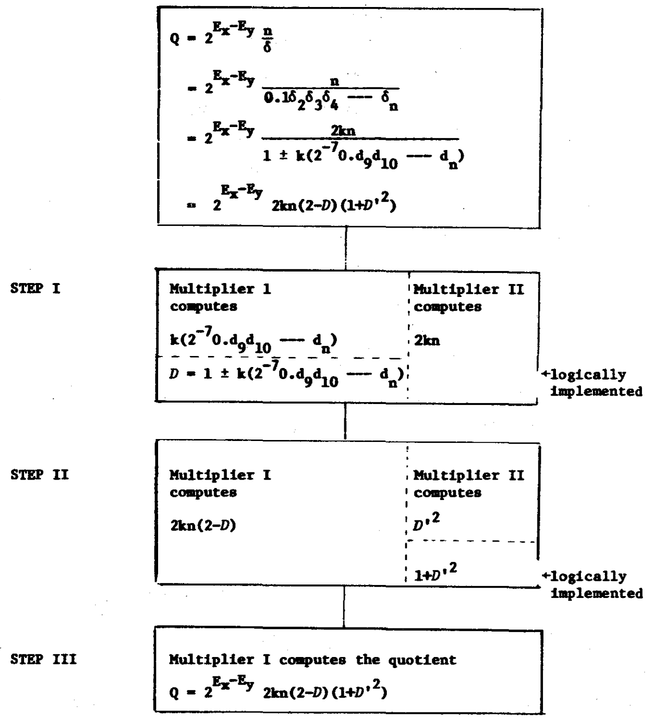

$Q = 2^{E_x-E_y}\frac{n}{\delta}$
$= 2^{E_x-E_y}\frac{n}{0.1\delta_2\delta_3\delta_4 --- \delta_n}$
$= 2^{E_x-E_y}\frac{2kn}{1 \pm k(2^{-7}0.d_9d_{10} --- d_n)}$
$= 2^{E_x-E_y}2kn(2-D)(1+D'^2)$
STEP I
Multiplier 1 computes
$k(2^{-7}0.d_9d_{10} --- d_n)$
$D = 1 \pm k(2^{-7}0.d_9d_{10} --- d_n)$
Multiplier II computes
2kn
←logically implemented
STEP II
Multiplier I computes
$2kn(2-D)$
Multiplier II computes
D'^2
$1+D'^2$
←logically implemented
STEP III
Multiplier I computes the quotient
$Q = 2^{E_x-E_y}2kn(2-D)(1+D'^2)$

Figure 1—The data flow of the proposed division schemes

DESCRIPTION

The implementation of this division scheme requires the logical implementation of d's, finding k from table 1, and three consecutive multiplications to find the Q.

The general data flow of this method can be divided into following three steps as shown in Figure 1.

In order to explain the operating procedure step by step, two examples are given.

Example 1. Find the quotient of 11957/44459 with 32-bit precision.

In binary these numbers are shown as

$$Q=\frac{10111010110101}{1010110110101011}$$

$$Q=2^{-2}\,\frac{0.10111010110101}{0.1010110110101011}$$

since $\delta_8=1$, all the d's are logically implemented according to equation (7).

$$\begin{aligned} d_1&=1\\ d_2&=0\\ d_3&=1\\ d_4&=0\\ d_5&=1\\ d_6&=1\\ d_7&=1 \end{aligned}$$

$$d_9d_{10}d_{11}d_{12}d_{13}d_{14}d_{15}d_{16}=\delta_8(\delta_9'\delta_{10}'\delta_{11}'\delta_{12}'\delta_{13}'\delta_{14}'\delta_{15}'\delta_{16})$$

$$=01010101$$

Using $d_1d_2d_3d_4d_5d_6d_7$ ($d_1=1$ always) as an entry into table 1, k is obtained.

$k=0.10111100010100100110010000000110$

STEP I.

Multiplier I computes $k(2^{-7}0.d_9d_{10}d_{11}d_{12}d_{13}d_{14}d_{15}d_{16})=$ 0.0000000001111101000011101011011 (round off beyond 32-bit)

$$D=0.1111111110000010111100010100101$$

$$2-D=1.0000000001111101000011101011011$$

For $\delta_8=1$, $2-D$ can be obtained directly by setting the leading bit of $k(2^{-7}0.d_9d_{10}\cdots d_{16})$ equal to one. No mathematical operations are needed. Multiplier II computes $2kn$

$2kn=1.00010010110111111111000011101100$

(round off beyond 32-bit)

STEP II.

Multiplier I computes $2kn(2-D)$

$$2kn(2-D)=1.0001001101100101111010001101010$$

Multiplier II computes D'^2

$$D'^2=0.000000000000000000111101000110$$

$$1+D'^2=1.000000000000000000111101000110$$

$1+D'^2$ can be obtained by directly setting the leading bit of D'^2 equal to one.

STEP III.

Multiplier I computes Q

$Q=2^{E_x-E_y}2kn(2-D)(1+D'^2)$

$Q=2^{-2}(1.0001001101100010011010101000110100101...)$

$\uparrow$ accuracy up to 34-bit

which is 0.2689444207 in decimal.

Example 2. Find the quotient of 11957/44203 with 33-bit precision.

In binary these numbers are shown as

$$Q=\frac{10111010110101}{1010110010101011}$$

$$Q=2^{-2}\,\frac{0.10111010110101}{0.1010110010101011}$$

since $\delta_8=0$, all the d's are obtained from equation (7). In this case they are equal to the δ's. Using $d_1d_2d_3d_4d_5d_6d_7$ ($d_1=1$ always) as an entry into Table I, k is obtained.

$$k=0.101111101000001011111010000011$$

STEP I.

Multiplier I computes

$k(2^{-7}0.d_9d_{10}\cdots d_{16})$

$$=0.0000000011111110100000101111101$$

$$D=1+k(2^{-7}0.d_9d_{10}\cdots d_{16})\quad\text{for}\quad\delta_8=0$$

$$D=1.0000000011111110100000101111101$$

Multiplier II computes $2kn$

$$2kn=1.0001011000010001110111000100100$$

TABLE I—The k-Table

d	k
1000000	1.0000000000000000000000000000000
1000001	0.1111110000001111110000001111110
1000010	0.1111100000111110000011111000010
1000011	0.1111010010001001100011010110000
1000100	0.1111000011110000111100001111000
1000101	0.1110110101110011000000111011011
1000110	0.1110101000001110101000001110101
1000111	0.1110011011000010101101000100100
1001000	0.1110001110001110001110001110010
1001001	0.1110000001110000001110000001110
1001010	0.1101110101100111110010001010011
1001011	0.1101101001110100000011011010100
1001100	0.1101011110010100001101011110011
1001101	0.1101010011000111011110110000010
1001110	0.1101001000001101001000001101001
1001111	0.1100111101100100011101001010100
1010000	0.1100110011001100110011001100110
1010001	0.1100101001000101100001111110011
1010010	0.1100011111001110000011000111110
1010011	0.1100010101100101110010000111110
1010100	0.1100001100001100001100001100010
1010101	0.1100000011000000110000001100000
1010110	0.1011111010000010111110100000110
1010111	0.1011110001010010011001000000110
1011000	0.1011101000101110100010111010001
1011001	0.1011100000010111000000101110000
1011010	0.1011011000001011011000001011011
1011011	0.1011010000001011010000001011010
1011100	0.1011001000010110010000101100100
1011101	0.1011000000101100000010110000001
1011110	0.1010111001001100010000010101110
1011111	0.1010110001110110100100011000010
1100000	0.1010101010101010101010101010101
1100001	0.1010100011101000001111110101100
1100010	0.1010011100101111000001010011101
1100011	0.1010010101111110101101010000001
1100100	0.1010001111010111000010100011111
1100101	0.1010001000110111110000110010110
1100110	0.1010000010100000101000001010000
1100111	0.1001111100010001011001011110100
1101000	0.1001110110001001110110001001111
1101001	0.1001110000001001110000001001110
1101010	0.1001101010010000011100111101101
1101011	0.1001100100011111000110100101001
1101100	0.1001011110110100001001011110111
1101101	0.1001011001001111110110100110110
1101110	0.1001010011110010000010010101000
1101111	0.1001001110011010100001011100010
1110000	0.1001001001001001001001001001001
1110001	0.1001000011111101101111000000101
1110010	0.1000111110111000001000111110111
1110011	0.1000111001111000001101010110111
1110100	0.1000110100111101110010110000100
1110101	0.1000110000001000110000001000110
1110110	0.1000101011011000111100101111110
1110111	0.1000100110101110010000001000101
1111000	0.1000100010001000100010001000100
1111001	0.1000011101100111101010110110000
1111010	0.1000011001001011100010100111111
1111011	0.1000010100110100000010000101010

TABLE I—The k-Table (Continued)

d	k
1111100	0.1000010000100001000010000100001
1111101	0.1000001100010010011011101001100
1111110	0.1000001000001000001000001000001
1111111	0.1000000100000010000001000000100

$d=0.d_1d_2d_3d_4d_5d_6d_7$

$k=\frac{1}{2d}$

TABLE II—The Quotient Digits q_{j+1} of Example 1 Generated by S-R-T Method

j	$2x_j$	q_{j+1}	operation
0	0.101110101101010	1	shift, subtract d
1	0.000110100101001	0	shift
2	0.001101001010010	0	shift
3	0.011010010100100	0	shift
4	0.110100101001000	1	shift, subtract d
5	0.010010011100101	0	shift
6	0.100100111001010	1	shift, subtract d
7	1.110010111101001	0	shift
8	1.100101111010010	0	shift
9	1.001011110100100	−1	shift, add d
10	1.101110011110011	0	shift
11	1.011100111100110	−1	shift, add d
12	0.010000101110111	0	shift
13	0.100001011101110	1	shift, subtract d
14	1.101100000110001	0	shift
15	1.011000001100010	−1	shift, add d
16	0.000111001101111	0	shift
17	0.001110011011110	0	shift
18	0.011100110111100	0	shift
19	0.111001101111000	1	shift, subtract d
20	0.011100101000101	0	shift
21	0.111001010001010	1	shift, subtract d
22	0.011011101101001	0	shift
23	0.110111011010010	1	shift, subtract d
24	0.010111111111001	0	shift
25	0.101111111110010	1	shift, subtract d
26	0.001001000111001	0	shift
27	0.010010001110010	0	shift
28	0.100100011100100	1	shift, subtract d
29	1.110010000011101	0	shift
30	1.100100000111010	0	shift
31	1.001000001110100	−1	shift, add d
32	1.100111010010011	0	shift
33	1.001110100100110	−1	shift, add d
34	1.100011111110111	0	shift
35	1.000111111101110	−1	shift, add d
36	1.100110110000111	0	shift

STEP II.

Multiplier I computes $2kn(2-D)$

$$2kn(2-D)=1.00010100111111010110100001001011$$

Multiplier II computes D'^2

$$D'^2=0.00000000000000000111111010000100$$

$$1+D'^2=1.00000000000000000111111010000100$$

STEP III.

Multiplier I computes Q

$$Q=2^{E_x-E_y}2kn(2-D)(1+D'^2)$$

TABLE III—The Quotient Digits q_{j+1} of Example 2 Generated by S-R-T Method

j	$2x_j$	q_{j+1}	operation
0	0.101110101101010	1	shift, subtract d
1	0.000111000101001	0	shift
2	0.001110001010010	0	shift
3	0.011100010100100	0	shift
4	0.111000101001000	1	shift, subtract d
5	0.011010111100101	0	shift
6	0.110101111001010	1	shift, subtract d
7	0.010101011101001	0	shift
8	0.101010111010010	1	shift, subtract d
9	1.111110111111001	0	shift
10	1.111101111110010	0	shift
11	1.111011111100100	0	shift
12	1.110111111001000	0	shift
13	1.101111110010000	0	shift
14	1.011111100100000	0	shift
15	1.011111001000000	−1	shift, add d
16	0.010100100101011	0	shift
17	0.101001001010110	1	shift, subtract d
18	1.111100000000001	0	shift
19	1.111000000000010	0	shift
20	1.110000000000100	0	shift
21	1.100000000001000	0	shift
22	1.000000000010000	−1	shift, add d
23	1.010110011001011	−1	shift, add d
24	0.000011001000001	0	shift
25	0.000110010000010	0	shift
26	0.001100100000100	0	shift
27	0.011001000001000	0	shift
28	0.110010000010000	1	shift, subtract d
29	0.001101101110101	0	shift
30	0.011011011101010	0	shift
31	0.110110111010100	1	shift, subtract d
32	0.010111011111101	0	shift
33	0.101110111111010	1	shift, subtract d
34	0.000111101001001	0	shift
35	0.001111010010010	0	shift
36	0.011110100100100	0	shift

TABLE IV—The Comparison of Various Methods Used to Obtain the Quotient with 32-bit Precision

M (multiplication)
S (subtraction)

Method / Precision digit	Anderson	Ferrari	S-R-T	Wallace	The proposed method
time required to obtain 32-bit precision quotient	4M	6M, 2S	Example 1 13 cycles	4M	3M
	12 cycles	20 cycles	Example 2 10 cycles	12 cycles	9 cycles

$$Q=2^{-2}(1.00010100111111100111101000010010000\cdots)$$

accuracy up to 33 bit

Which is 0.02705020021 in decimal.

COMPARISON

In order to compare the speed of the various division methods, the machine cycle will be used as a basic unit. The precision of the quadratic convergence method with preselected starting block is easy to predict. However, the exactly shifting pattern of the S-R-T method has to be evaluated step by step. According to Anderson[1] that 32-bit by 32-bit multiplication takes 3 main machine cycles, this is the converting factor used in this paper to convert the number of multiplications into machine cycles.

The iterative multiplication approach includes the Wallace proposed scheme,[13] the Anderson's denominator convergence method[1,6] and Ferrari's OGS method.[3] For solving the example 1 and 2 in section III with 32-bit precision, Wallace method requires 3 iterative multiplications with starting block of 6-bit to find the reciprocal of the denominator, totally 4 multiplications are needed to find the quotient. Anderson's denominator convergence scheme requires 4 iterative multiplications with starting block of 7-bit. Ferrari's OGS method requires 5 multiplications and two subtractions with starting block of 6-bit to find the reciprocal of the denominator, totally 6 multiplications and 2 subtractions are needed to find the quotient. Therefore, the use of Wallace method requires 12 machine cycles, Anderson's method 12 cycles, and Ferrari's method 20 cycles.

For 32-bit precision, the use of S-R-T method to find the quotient in example 1 and example 2 require 13 and 10 cycles respectively. The set of rules for quotient determination is the following

$$q_{j+1}=1 \qquad 2x_j \geq (\tfrac{1}{2})$$

$$q_{j+1}=0 \qquad (-\tfrac{1}{2}) \leq 2x_j \leq (\tfrac{1}{2})$$

$$q_{j+1}=-1 \quad 2x_j < (-\tfrac{1}{2})$$

where $2x_j$ is the partial dividend at the start of jth step, q_j is the quotient, the d and $2x_j$ for example 1 can be written as

$$d=0.1010110110101011$$

$$2x_j=0.10111010110101$$

for example 2 can be written as

$$d=0.1010110110101011$$

$$2x_j=0.10111010110101$$

The quotient digits q_j for example 1 and 2 are shown in Tables II and III respectively.

The detailed comparison of the division speed of the various methods is listed in Table IV.

REFERENCES

1 S F ANDERSON J G EARL
R E GOLDSCHMIDT D M POWERS
Floating-point execution unit
IBM Journal of Research and Development Vol 11 No 1 pp 34-53 January 1967

2 W BUCHHOLZ
Planning a computer system
McGraw-Hill pp 216 1962

3 D FERRARI
A Division method using a parallel multiplier
IEEE Transactions on Electronic Computers Vol Ec-16 pp 224-226 April 1967

4 C V FREIMAN
Statistical analysis of certain binary division algorithm
Proceeding IRE Vol 49 pp 91-103 January 1961

5 H L GARNER
Number systems and arithmetic
Advanced in Computers Edited by F L Alt and M Rubinoff Academic Press pp 173-175 1965

6 R E GOLDSCHMIDT
Application of division by convergence
M Sc Thesis Dept of Elec Engineering MIT June 1964

7 C C GOTLIEB J N P HUME
High-speed data processing
McGraw-Hill pp 51-52 1958

8 H LING
High speed computer division using multiple-bit decoding technique
To be published

9 O L MacSORLEY
High-speed arithmetic in binary computers
Proceeding of IRE Vol 49 pp 67-91 January 1961

10 J E ROBERTSON
A *new class of digital division method*
IEEE Transaction on Computers Vol Ec-7 pp 218-222 Sept 1958

11 P RABINOWITZ
Multiple precision division
CACM 4 pp 98 1961

12 K D TOCHER
Techniques of multiplication and division for automatic binary computers
Quart Journal Mach and Applied Math Vol XI Pt 3 pp 364-384 1958

13 C S WALLACE
A suggestion for a fast multiplier
IEEE Transaction on Computers Vol Ec-13 pp 14-17 Feb 1964

A unified algorithm for elementary functions

by J. S. WALTHER

Hewlett-Packard Company
Palo Alto, California

SUMMARY

This paper describes a single unified algorithm for the calculation of elementary functions including multiplication, division, sin, cos, tan, arctan, sinh, cosh, tanh, arctanh, ln, exp and square-root. The basis for the algorithm is coordinate rotation in a linear, circular, or hyperbolic coordinate system depending on which function is to be calculated. The only operations required are shifting, adding, subtracting and the recall of prestored constants. The limited domain of convergence of the algorithm is calculated, leading to a discussion of the modifications required to extend the domain for floating point calculations.

A hardware floating point processor using the algorithm was built at Hewlett-Packard Laboratories. The block diagram of the processor, the microprogram control used for the algorithm, and measures of actual performance are shown.

INTRODUCTION

The use of coordinate rotation to calculate elementary functions is not new. In 1956 Volder developed a class of algorithms for the calculation of trigonometric and hyperbolic functions, including exponential and logarithm. In 1959 he described a COordinate Rotation DIgital Computer (CORDIC) for the calculation of trigonometric functions, multiplication, division, and conversion between binary and mixed radix number systems. Daggett in 1959 discussed the use of the CORDIC for decimal-binary conversions. In 1968 Liccardo did a master's thesis on the class of CORDIC algorithms.

It is not generally realized that many of these algorithms can be merged into one unified algorithm.

COORDINATE SYSTEMS

Let us consider coordinate systems parameterized by m in which the radius R and angle A of the vector $P=(x, y)$ shown in Figure 1 are defined as

$$R=[x^2+my^2]^{1/2}$$

$$A=m^{-1/2}\tan^{-1}[m^{1/2}y/x]$$

It can be shown that R is the distance from the origin to the intersection of the curve of constant radius with the x axis, while A is twice the area enclosed by the vector, the x axis, and the curve of constant radius, divided by the radius squared. The curves of constant radius for the circular $(m=1)$, linear $(m=0)$, and hyperbolic $(m=-1)$ coordinate systems are shown in Figure 1.

ITERATION EQUATIONS

Let a new vector $P_{i+1}=(x_{i+1}, y_{i+1})$ be obtained from $P_i=(x_i, y_i)$ according to

$$x_{i+1}=x_i+my_i\delta_i \tag{3}$$

$$y_{i+1}=y_i-x_i\delta_i \tag{4}$$

where m is the parameter for the coordinate system, and δ_i is an arbitrary value. The angle and radius of the new vector in terms of the old are given by

$$A_{i+1}=A_i-\alpha_i \tag{5}$$

$$R_{i+1}=R_i{*}K_i \tag{6}$$

where

$$\alpha_i=m^{-1/2}\tan^{-1}[m^{1/2}\delta_i] \tag{7}$$

$$K_i=[1+m\delta_i^2]^{1/2} \tag{8}$$

The angle and radius are modified by quantities which are independent of the coordinate values. Table I gives the equations for α_i and K_i after applying identities $A2$ and $A5$ from the appendix.

For n iterations we find

$$A_n=A_0-\alpha \tag{9}$$

$$R_n=R_0{*}K \tag{10}$$

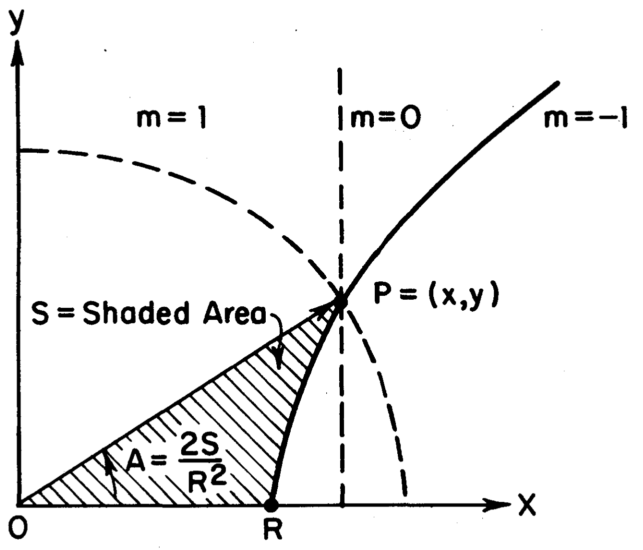


Figure 1—Angle A and Radius R of the vector $P=(x, y)$

where

$$\alpha=\sum_{i=0}^{n-1}\alpha_i \tag{11}$$

$$K=\prod_{i=0}^{n-1}K_i \tag{12}$$

The total change in angle is just the sum of the incremental changes while the total change in radius is the product of the incremental changes.

If a third variable z is provided for the accumulation of the angle variations

$$z_{i+1}=z_i+\alpha_i \tag{13}$$

and the set of difference equations (3), (4), and (13) is solved for n iterations, we find,

$$x_n=K\{x_0 \cos(\alpha m^{1/2})+y_0 m^{1/2}\sin(\alpha m^{1/2})\} \tag{14}$$

$$y_n=K\{y_0 \cos(\alpha m^{1/2})-x_0 m^{-1/2}\sin(\alpha m^{1/2})\} \tag{15}$$

$$z_n=z_0+\alpha \tag{16}$$

where α and K are as in equations (11) and (12).

TABLE I—Angles and Radius Factors

Coordinate System m	Angle α_i	Radius Factor K_i
1	$\tan^{-1}\delta_i$	$(1+\delta_i^2)^{1/2}$
0	δ_i	1
−1	$\tanh^{-1}\delta_i$	$(1-\delta_i^2)^{1/2}$

These relations are summarized in Figure 2 for $m=1$, $m=0$ and $m=-1$ for the following special cases.

1. A is forced to zero: $y_n=0$.
2. z is forced to zero: $z_n=0$.

The initial values x_0, y_0, z_0 are shown on the left of each block in the figure while the final values x_n, y_n, z_n are shown on the right. The identities given in the appendix were used to simplify these results. By the proper choice of the initial values the functions $x\,z$, y/x, $\sin z$, $\cos z$, $\tan^{-1} y$, $\sinh z$, $\cosh z$, and $\tanh^{-1} y$ may be obtained. In addition the following functions may be generated.

$$\tan z=\sin z/\cos z \tag{17}$$

$$\tanh z=\sinh z/\cosh z \tag{18}$$

$$\exp z=\sinh z+\cosh z \tag{19}$$

$$\ln w=2\tanh^{-1}[y/x] \text{ where } x=w+1 \text{ and } y=w-1 \tag{20}$$

$$(w)^{1/2}=(x^2-y^2)^{1/2} \text{ where } x=w+\tfrac{1}{4} \text{ and } y=w-\tfrac{1}{4} \tag{21}$$

CONVERGENCE SCHEME

The angle A of the vector P may be forced to zero by a converging sequence of rotations α_i which at each step brings the vector closer to the positive x axis.

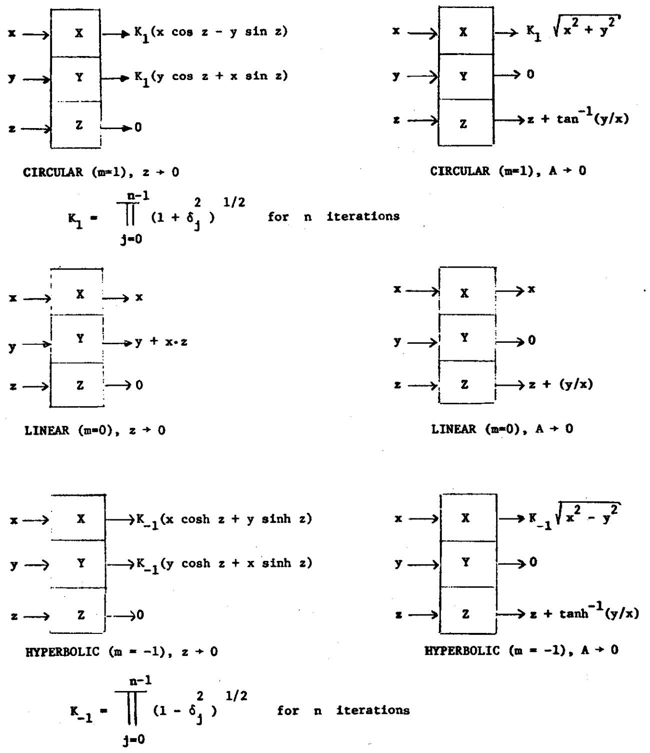

x → X → K₁(x cos z − y sin z)
y → Y → K₁(y cos z + x sin z)
z → Z → 0
CIRCULAR (m=1), z → 0
x → X → K₁ √(x² + y²)
y → Y → 0
z → Z → z + tan⁻¹(y/x)
CIRCULAR (m=1), A → 0
K₁ = ∏_{j=0}^{n-1} (1 + δ_j²)^{1/2} for n iterations
x → X → x
y → Y → y + x·z
z → Z → 0
LINEAR (m=0), z → 0
x → X → x
y → Y → 0
z → Z → z + (y/x)
LINEAR (m=0), A → 0
x → X → K₋₁(x cosh z + y sinh z)
y → Y → K₋₁(y cosh z + x sinh z)
z → Z → 0
HYPERBOLIC (m = −1), z → 0
x → X → K₋₁ √(x² − y²)
y → Y → 0
z → Z → z + tanh⁻¹(y/x)
HYPERBOLIC (m = −1), A → 0
K₋₁ = ∏_{j=0}^{n-1} (1 − δ_j²)^{1/2} for n iterations

Figure 2—Input-output functions for CORDIC modes

The magnitude of each element of the sequence may be predetermined, but the direction of rotation must be determined at each step such that

$$|A_{i+1}| = ||A_i| - \alpha_i| \tag{22}$$

The sum of the remaining rotations must at each step be sufficient to bring the angle to at least within α_{n-1} of zero, even in the extreme case where $A_i=0$, $|A_{i+1}|=\alpha_i$. Thus,

$$\alpha_i - \sum_{j=i+1}^{n-1} \alpha_j < \alpha_{n-1} \tag{23}$$

The domain of convergence is limited by the sum of the rotations.

$$|A_0| - \sum_{j=0}^{n-1} \alpha_j < \alpha_{n-1} \tag{24}$$

$$\max |A_0| = \alpha_{n-1} + \sum_{j=0}^{n-1} \alpha_j \tag{25}$$

To show that A converges to within α_{n-1} of zero within n steps we first prove the following theorem.

Theorem

$$|A_i| < \alpha_{n-1} + \sum_{j=i}^{n-1} \alpha_j \tag{26}$$

holds for $i \geq 0$.

Proof

We proceed by induction on i. The hypothesis (26) holds for $i=0$ by (24). We now show that if the hypothesis is true for i then it is also true for $i+1$. Subtracting α_i from (26) and applying (23) at the left side yields

$$-\left[\alpha_{n-1} + \sum_{j=i+1}^{n-1} \alpha_j\right] < -\alpha_i < |A_i|$$

$$-\alpha_i < \left[\alpha_{n-1} + \sum_{j=i+1}^{n-1} \alpha_j\right] \tag{27}$$

Application of (22) then yields

$$|A_{i+1}| < \alpha_{n-1} + \sum_{j=i+1}^{n-1} \alpha_j \tag{28}$$

as was to be shown. Therefore, by induction, the hypothesis holds for all $i \geq 0$.
In particular, the theorem is true for $i=n$ so that

$$|A_n| < \alpha_{n-1}. \tag{29}$$

The same scheme may be used to force the angle in z to zero. The proof of convergence proceeds exactly as before except that A is replaced by z in equations (22) through (29). By equation (25) z has the same domain of convergence as A.

$$\max |z_0| = \max |A_0|. \tag{30}$$

Note that since K is a function of δ_i^2, where $\delta_i = m^{-1/2} \tan[m^{1/2}\alpha_i]$, K is independent of the sequence of signs chosen for the α_i. Thus, for a fixed sequence of α_i magnitudes the constant $1/K$ may be used as an initial value to counteract the factor K present in the final values.

TABLE II—Shift Sequences for a binary code

radix ρ	coordinate system m	shift sequence $F_{mi}; i \geq 0$	domain of convergence $\max |A_0|$	radius factor K
2	1	0, 1, 2, 3, 4, i, . . .	~1.74	~1.65
2	0	1, 2, 3, 4, 5, $i+1$, . . .	1.0	1.0
2	−1	1, 2, 3, 4, 4, 5, ... *	~1.13	~0.80

* for $m=-1$ the following integers are repeated: $\{4, 13, 40, 121, \ldots, k, 3k+1, \ldots\}$

USE OF SHIFTERS

The practical use of the algorithm is based on the use of shifters to effect the multiplication by δ_i. If ρ is the radix of the number system and F_i is an array of integers, where $i \geq 0$, then a multiplication of x by

$$\delta_i = \rho^{-F_i} \tag{31}$$

is simply a shift of x by F_i places to the right. The integers F_i must be chosen such that the angles

$$\alpha_{m,F_i}| = m^{-1/2} \tan^{-1}(m^{1/2}\rho^{-F_i}) \tag{32}$$

satisfy the convergence criterion (23). The domain of convergence is then given by (25).

Table II shows some F sequences, convergence domains, and radius factors for a binary code.

The hyperbolic mode ($m=-1$) is somewhat complicated by the fact that for $\alpha_i = \tanh^{-1}(2^{-i})$ the convergence criterion (23) is not satisfied. However, it can be shown that

$$\alpha_i - \left(\sum_{j=i+1}^{n-1} \alpha_j\right) - \alpha_{3i+1} < \alpha_{n-1} \tag{33}$$

and that therefore if the integers $\{4, 13, 40, 121, \ldots, k, 3k+1, \ldots\}$ in the F_i sequence are repeated then (23) becomes true.

TABLE III—Prescaling Identities

Identity	Domain	Domain of Convergence
$\sin\left(Q\frac{\pi}{2}+D\right)=\begin{cases}\sin D \text{ if } Q \bmod 4=0\\ \cos D \text{ if } Q \bmod 4=1\\ -\sin D \text{ if } Q \bmod 4=2\\ -\cos D \text{ if } Q \bmod 4=3\end{cases}$	$\lvert D\rvert<\frac{\pi}{2}=1.57$	1.74
$\cos\left(Q\frac{\pi}{2}+D\right)=\begin{cases}\cos D \text{ if } Q \bmod 4=0\\ -\sin D \text{ if } Q \bmod 4=1\\ -\cos D \text{ if } Q \bmod 4=2\\ \sin D \text{ if } Q \bmod 4=3\end{cases}$	$\lvert D\rvert<\frac{\pi}{2}=1.57$	1.74
$\tan\left(Q\frac{\pi}{2}+D\right)=\sin\left(Q\frac{\pi}{2}+D\right)\Big/\cos\left(Q\frac{\pi}{2}+D\right)$	$\lvert D\rvert<\frac{\pi}{2}=1.57$	1.74
$\tan^{-1}\left(\frac{1}{y}\right)=\frac{\pi}{2}-\tan^{-1}(y)$	$\lvert y\rvert<1.0$	∞
$\sinh(Q\log_e 2+D)=\frac{2^Q}{2}[\cosh D+\sinh D-2^{-2Q}(\cosh D-\sinh D)]$	$\lvert D\rvert<\log_e 2=0.69$	1.13
$\cosh(Q\log_e 2+D)=\frac{2^Q}{2}[\cosh D+\sinh D+2^{-2Q}(\cosh D-\sinh D)]$	$\lvert D\rvert<\log_e 2=0.69$	1.13
$\tanh(Q\log_e 2+D)=\sinh(Q\log_e 2+D)/\cosh(Q\log_e 2+D)$	$\lvert D\rvert<\log_e 2=0.69$	1.13
$\tanh^{-1}(1-M2^{-E})=\tanh^{-1}(T)+(E/2)\log_e 2$	$0.17<T<0.75$	$(-0.81, 0.81)$
where $T=(2-M-M2^{-E})/(2+M-M2^{-E})$	for $0.5\le M<1, E\ge 1$	
$\exp(Q\log_e 2+D)=2^Q(\cosh D+\sinh D)$	$\lvert D\rvert<\log_e 2=0.69$	1.13
$\log_e(M2^E)=\log_e M+E\log_e 2$	$0.5\le M<1.0$	$(0.10, 9.58)$
$\operatorname{sqrt}(M2^E)=\begin{cases}2^{E/2}\operatorname{sqrt}(M) & \text{if } E \bmod 2=0\\ 2^{(E+1)/2}\operatorname{sqrt}(M/2) & \text{if } E \bmod 2=1\end{cases}$	$\begin{cases}0.5\le M<1.0\\ 0.25\le M/2<0.5\end{cases}$	$(0.03, 2.42)$
$(M_x2^{E_x})(M_z2^{E_z})=(M_xM_z)2^{E_x+E_z}$	$0.5\le\lvert M_z\rvert<1.0$	$(-1.0, 1.0)$
$(M_y2^{E_y})/(M_x2^{E_x})=(M_y/2M_x)2^{E_y-E_x+1}$	$0.25\le\lvert M_y/2M_x\rvert<1.0$	$(-1.0, 1.0)$

EXTENDING THE DOMAIN

The limited domain imposed by the convergence criterion (25) may be extended by means of the prescaling identities shown in Table III. For example, to calculate the sinc of a large argument, we first divide the argument by $\pi/2$ obtaining a quotient Q and a remainder D where $\lvert D\rvert<\pi/2$. The table shows that only $\sin D$ or $\cos D$ need be calculated and that $\pi/2$ is within the domain of convergence. Note that the sine and cosine can be generated simultaneously by the CORDIC algorithm and that the answer may then be chosen as plus or minus one of these according to Q mod 4. As a second example, to calculate the logarithm of a large argument we first shift the argument's binary point E places until it is just to the left of the most significant non-zero bit. The fraction M then satisfies $0.5\le M<1.0$ and as shown in the table therefore falls within the domain of convergence. The answer is calculated as $\log_e M+E\log_e 2$.

ACCURACY

The accuracy at the nth step is determined in theory by the size of the last of the converging sequence of rotations α_i, and for large n is approximately equal in digits to F_{n-1}. The accuracy in digits may conveniently be made equal to L, the length of storage used for each variable, by choosing n such that $F_{n-1}=L$.

In practice the accuracy is limited by the finite length of storage. The truncation of input arguments performed to make them fit within the storage length gives rise to unavoidable error, the size of which depends on the sensitivity of the calculated function to small changes in the input argument. In a binary code, the truncation of intermediate results after each of L iterations gives rise to a total of at most $\log_2 L$ bits of error. This latter error can be rendered harmless by using $L+\log_2 L$ bits for the storage of intermediate results.

In a normalized floating point number system it is desirable that all L bits of the result be accurate, independent of the absolute size of the argument. To accomplish this for very small arguments it is necessary to keep each storage register in a normalized form; i.e., in a form where there are no leading zeros. It is possible to do this by transforming the iteration equations (3), (4), (13) to a normalized form according to the following substitutions.

$$x \text{ becomes } x' \tag{34}$$

$$y \text{ becomes } y'\,2^{-E} \tag{35}$$

$$z \text{ becomes } z'\,2^{-E} \tag{36}$$

$$\alpha_F \text{ becomes } \alpha_F'\,2^{-F} \tag{37}$$

where E, a positive integer, is chosen such that the initial argument, placed into either the y or z register, is normalized.

The result of the substitutions is

$$x' \leftarrow x' + my'2^{-(F+E)} \tag{38}$$

$$y' \leftarrow y' - x'2^{-(F-E)} \tag{39}$$

$$z' \leftarrow z' + \alpha_F'2^{-(F-E)} \tag{40}$$

For simplicity the subscripts i and $i+1$ have been dropped. Instead, α has been expressed as a function of F as in equation (32), and the replacement operator ($\leftarrow$) has been used. i may be initialized to a value such that $F_i=E$:

$$i_{\text{initial}} \leftarrow \{i \mid F_i = E\}, \tag{41}$$

and n may be chosen such that L significant bits are obtained:

$$n \leftarrow \{n \mid F_{n-1} - E = L\}. \tag{42}$$

Note that $n - i_{\text{initial}} \approx L$ and that therefore providing $L+\log_2 L$ bits for the storage of intermediate results is still adequate.

The radius factor K is now a function of $i=i_{\text{initial}}$ as well as m.

$$K_{m,i} = \prod_{j=i}^{n-1} (1+m2^{-2F_j})^{1/2} \tag{43}$$

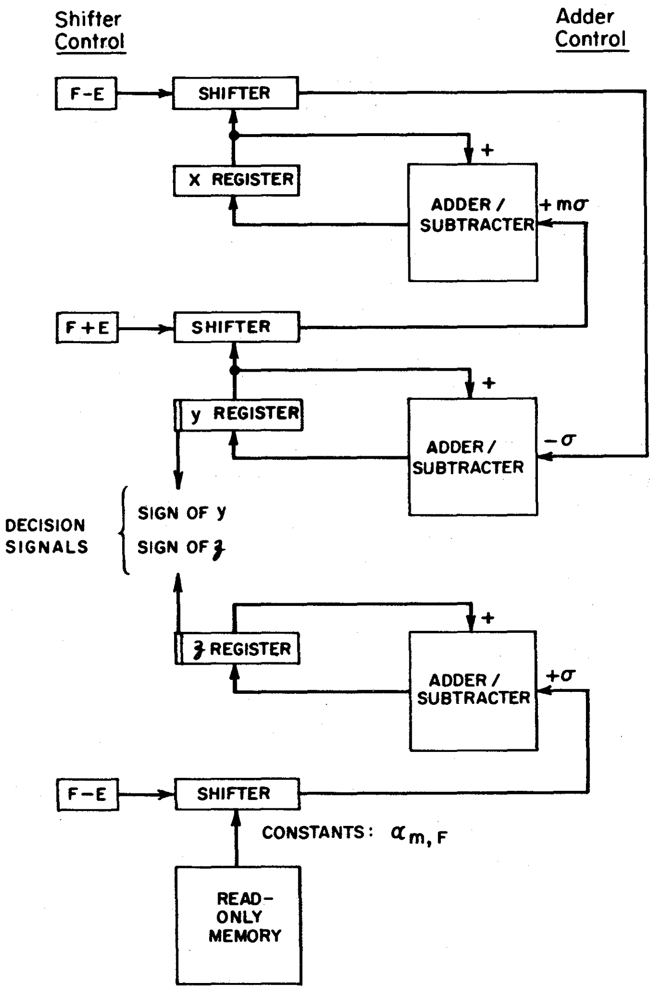


Figure 3—Hardware block diagram

Fortunately, not all the reciprocal constants $1/K_{m,i}$ need to be stored since for large values of i

$$\frac{1}{K_{m,i}} \approx 1 - m(2/3)2^{-2i}, \tag{44}$$

and therefore all the constants having $i > L/2$ are identical to within L significant bits. Therefore, only $L/2$ constants need to be stored for $m=+1$ and also for $m=-1$. For $m=0$ no constants need to be stored since $K_{0,i}=1$ for $i \geq 1$.

A similar savings in storage can be made for the angle constants $\alpha_{m,F}$ since for large values of F

$$\alpha'_{m,F} \equiv \alpha_{m,F}\,2^F \approx 1 - m(1/3)2^{-2F}, \tag{45}$$

and thus, as for the K constants, only $L/2$ constants need to be stored for $m=+1$ and also for $m=-1$. For $m=0$ no constants need to be stored since $\alpha'_{0,}{}^{F}=1$ for $F \geq 1$.

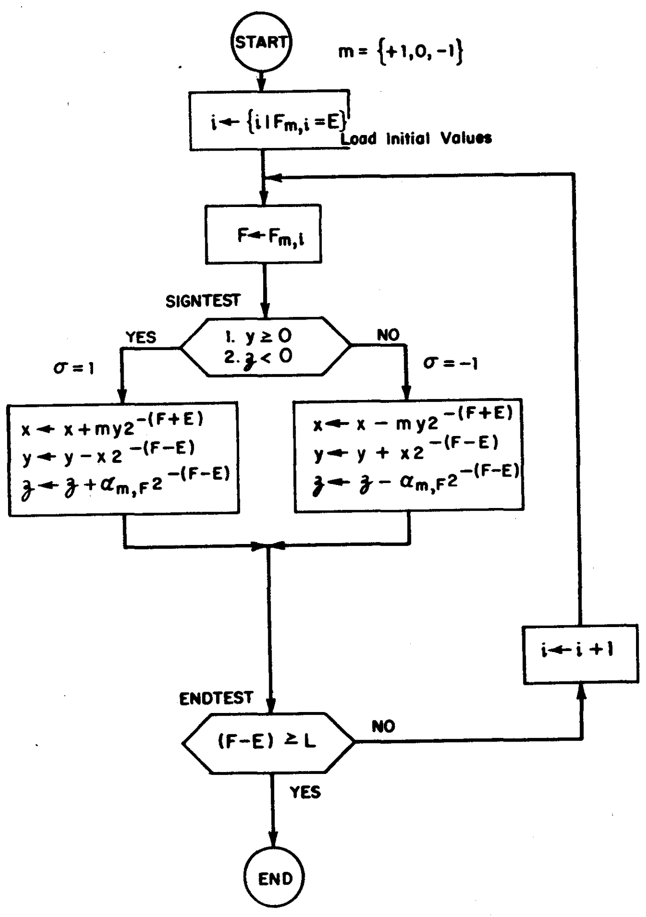


Figure 4—Flowchart of the microprogram control

HARDWARE IMPLEMENTATION

A hardware floating point processor based on the CORDIC algorithm has been built at Hewlett-Packard Laboratories. Figure 3 shows a block diagram of the processor which consists of three identical arithmetic units operated in parallel. Each arithmetic unit contains a 64-bit register, an 8-bit parallel adder/subtracter, and an 8-out-of-48 multiplex shifter. The assembly of arithmetic units is controlled by a microprogram stored in a read-only memory (ROM), which also contains the angle and radius-correction constants. The ROM contains 512 words of 48 bits each and operates on a cycle time of 200 nanoseconds.

The processor accepts three data types: 48-bit floating point, 32-bit floating point, and 32-bit integer. All the functions are calculated to 40 bits of precision (approximately 12 decimal digits), and the accuracy is limited only by the truncation of input arguments.

The essential aspects of the microprogram used to execute the CORDIC algorithm are shown in Figure 4. The initial argument and correction constants are loaded into the three registers and m is set to one of the three values 1, 0, −1. If the initial argument is small, it is normalized and E is set to minus the binary exponent of the result, otherwise, E is set to zero. Next, i is initialized to a value such that $F_{m,i}=E$. A loop is then entered and is repeated until $F_{m,i}-E=L$. In this loop the direction of rotation necessary to force either of the angles A or z to zero is chosen; the binary variable σ, used to control the three adder/subtracters, is set to either +1 or −1; and the iteration equations are executed.

Table IV gives a breakdown of the maximum execution times for the most important functions. The figures in the column marked "data transfers from computer" are the times for operand and operation code transfers between the processor and an HP-2116 computer.

The processor retains the result of each executed function. Thus, add, subtract, multiply and divide require only one additional operand to be supplied, and the one operand functions do not require any operand transfers. The first operand is loaded via the LOAD instruction, and the final result is retrieved via the STORE instruction.

TABLE IV—Maximum Execution Times

ROUTINE	CORDIC EXECUTION μsec	PRESCALE, NORMALIZE, MISC. μsec	DATA TRANSFERS FROM COMPUTER μsec	TOTAL μsec
LOAD	0	5	25	30
STORE	0	0	15	15
ADD	0	15	25	40
SUBTRACT	0	25	25	50
MULTIPLY	60	15	25	100
DIVIDE	60	15	25	100
SIN	70	85	5	160
COS	70	85	5	160
TAN	130	85	5	220
ATAN	70	15	5	90
SINH	70	55	5	130
COSH	70	55	5	130
TANH	130	55	5	190
ATANH	70	45	5	120
EXPONENTIAL	70	55	5	130
LOGARITHM	70	45	5	120
SQUARE-ROOT	70	25	5	100

CONCLUSION

The unified CORDIC algorithm is attractive for the calculation of elementary functions because of its simplicity, its accuracy, and its capability for high speed execution via parallel processing. Its applications include desktop calculators, as in the HP-9100 series; air navigation computers, as described in Volder's original work; and floating point processors, as illustrated in this paper.

ACKNOWLEDGMENTS

The author wishes to thank the many people at Hewlett-Packard Laboratories and Cupertino Division for their contributions and support.

REFERENCES

1 D H DAGGETT
Decimal-binary conversion in Cordic
IRE Transactions on Electronic Computers Vol EC-8 No 3 pp 335-339 September 1959

2 M A LICCARDO
An interconnect processor with emphasis on Cordic mode operation
Masters Thesis EE Dept University of California at Berkeley September 1968

3 J E VOLDER
Binary computation algorithms for coordinate rotation and function generation
Convair Report IAR-1 148 Aeroelectronics Group June 1956

4 J E VOLDER
The Cordic trigonometric computing technique
IRE Transactions on Electronic Computers Vol EC-8 No 3 pp 330-334 September 1959

APPENDIX

Mathematical identities

Let $i = (-1)^{1/2}$

$$z \equiv \lim_{m \to 0} m^{-1/2} \sin(zm^{1/2}) \tag{A1}$$

$$z \equiv \lim_{m \to 0} m^{-1/2} \tan^{-1}(zm^{1/2}) \tag{A2}$$

$$\sinh z \equiv -i \sin(iz) \tag{A3}$$

$$\cosh z \equiv \cos(iz) \tag{A4}$$

$$\tanh^{-1} z \equiv -i \tan^{-1}(iz) \tag{A5}$$

A software system for tracing numerical significance during computer program execution

by H. S. BRIGHT, B. A. COLHOUN, and F. B. MALLORY

Computation Planning, Inc.
Washington, D. C.

INTRODUCTION

This report will describe and discuss a presently-operational software system for tracing and displaying numerical accuracy in digital computer calculation. A FORTRAN program processed by the system is executed in an artificial arithmetic, in which every arithmetic step produces, in addition to the numerical result, an estimate of the number of significant digits in that result. Programs have been processed successfully with significance mode segments totalling as many as 1400 FORTRAN statements.

Actual input data and results of previous computations are used in unaltered form. The user may specify initial accuracy of data; may select any part or parts of a program for execution in the error-indicating mode; and may request accuracy of any variable quantity at any point during program execution.

Numerical errors in digital computations which are logically correct arise from three principal sources: "inherent," or input data inaccuracy; "analytic," or mathematical compromise; and "generated" error, from finite-precision machine arithmetic.[1]

In the work described here, most concern has been given to inherent and generated error categories. We have considered mainly the limitations of input data and the behavior of subtract error and rounding error generation. Specifically, we have developed a computer program for tracing and displaying, and localizing the sources of, the changes in number significance that occur during machine computation.

MACHINE ARITHMETIC

In scientific and engineering calculations, widespread use is made of the *normalized floating-point* arithmetic feature in which a number is represented by a *fraction part* (a number between positive and negative unity) and an *exponent part* (a scale factor, an integer power to which the machine's number base or radix R is to be raised to constitute the desired multiplier).

The floating-point feature is an automatic means for scaling the number value whenever an arithmetic operation would result either in an unnormalized number (i.e., one or more high-order zeros in the base-R representation of the fraction part) or in a carry out of the high-order digit position of the fraction part. This automatic feature has the advantage of great user convenience, but suffers from the danger that gross errors may be generated without being evident to the user.

Thus, conventional floating-point arithmetic is dangerously *optimistic*, in that it may camouflage error under an unknown number of not-known-to-be-valid digits. At the other extreme, error bounds analysis (which can be mechanized through the use of Interval Arithmetic)[6] provides worst-case estimates and in general is *pessimistic*. The initial version of the present system seeks to provide *realistic* predictions of remaining accuracy, utilizing significance arithmetic.[5]

Subtract error, following subtraction of two nearly-equal quantities, is the result of the cancellation of a large number of high-order digits. The fact that this can happen internally in a large calculation may cause sudden loss of all or almost all significance. This is the most frequent cause for gross failure of a calculation in a program that is logically correct. It is often difficult to anticipate, and a user would need to hand-simulate the entire calculation, using actual input data, in order to observe the numerical behavior of calculations. In a typical program that is large enough to justify computer usage, the number of program steps executed is so large as to make hand simulation infeasible.

Round-off error may be propagated up to or through the lowest-order retained digit when lower-order digits are removed in order to represent results in an arithmetic register. Wilkinson[3] showed that in some repeated operations (e.g., summations) the accumulated round-off error may grow rapidly; with a simple recursion formula, the rounding error can accumulate exponentially as the number of terms.

The example below involves a common procedure for checking the validity of results obtained during computation. It demonstrates the danger of judgments without consideration for, and knowledge of, numerical significances encountered during calculation. Forsythe[4] attributes to Moler the following set of two linear equations, two solutions S_1 and S_2, and the corresponding residuals R_1 and R_2, and asks which solution is more nearly correct:

$$0.780x+0.563y-0.217=0$$
$$0.913x+0.659y-0.254=0$$

$$S_1=(x_1, y_1)=(0.999, -1.001)$$
$$S_2=(x_2, y_2)=(0.341, -00.87)$$

$$R_1(x_1, y_1)=(-.001343, -.001572)$$
$$R_2(x_2, y_2)=(-.000001, 0)$$

Residual observation (how nearly does an approximate solution satisfy a problem) is a widely-used method for checking the accuracy of computer programs and procedures for solving polynomial equations, systems of non-linear and linear equations, matrix inversions, and the like. It might be concluded, because residuals R_2 are smaller than R_1, that S_2 is the better of the two solutions; however, note that S_1 compares favorably with the true solution (1.0, -1.0), while S_2 does not.

In this example, the examination of residuals, without consideration of the significance of the coefficients and of the results, leads to the wrong conclusion. The coefficients each have three significant digits as do x_1, y_1 and x_2, while y_2 has only two significant digits; thus, the R_i cannot be calculated with the accuracy displayed and should be written as

$$R_1(x_1, y_1)=(-.001N_i, -.001N_j)$$

$$R_2(x_2, y_2)=(-.00N_k-.00N_l)$$

where each N_m is a digit of unknown value. Obviously now R_1 is the better solution; but this is obvious only if the significances of the numbers are known.

In this example, the calculations are so brief and simple that the results can be calculated quickly by hand. In a typical computer calculation, this is not the case. The system described here can, however, supply the needed significance estimates for even a large and complicated calculation, so that judgments can be made by the user with reasonable objectivity.

The basic concept of Significance Arithmetic[5] is that each number should carry with it an index specifying the most probable number of digits currently valid. At each arithmetic step, significance algorithms are executed to perform an estimate of the result significance, taking into account the nature of the calculation step, the previous significance and value of each incoming operand, and the precision of the machine for this operational step. The nature of rounding, if any, applied by the hardware, as well as the actual overall accuracy of supporting routines, must be taken into account.

APPLICABILITY

Besides providing a testing and debugging tool for numerical calculation, the system as described here has other potential applications, all related to the user's need to make explicitly evident the relationship between precision ("How many digits are recorded?") and significance or accuracy ("How many of the recorded digits are valid?").

One of these applications is assisting a user to make a choice between single-precision, double-precision, or perhaps higher-order calculations. This choice is usually made on intuitive grounds, perhaps coerced by memory size and/or processing time limitations, and certainly affected by custom or perhaps dogma within the computing organization. By executing his program in significance arithmetic, a user can obtain a factual measure of the result significance, and can choose to use the order of precision which yields the required accuracy.

An obvious application is the use of this system as an experimental tool for determining computer word-length (precision) requirements.

It is less obvious that the system can also be used to determine the accuracy requirements for input data: By executing an entire program or program system with different preset values for significance of one input datum (simulating various *assumed accuracy* levels for that particular item) the overall result of different accuracies for that datum can be observed. Accuracy specification for an input datum can thus be determined by test.

User source programs, expressed in FORTRAN language, need be modified by the user only to the extent that certain control statements or *pseudo-operations* must be inserted to guide the system (hereinafter called SigPac) by requesting the desired services and providing necessary data initialization information.

The SigPac System processes the user program including the user-inserted pseudo operations, preinitializes significance of entities not initialized by the user, and converts all arithmetic steps to significance arithmetic operations. The result of this process is a FORTRAN program which, when executed after compilation, produces output as requested by the user including significance information, with normal output suppressed if desired.

Execution of a program in significance mode means that significance arithmetic procedures are employed which replace the hardware commands for single and double precision addition, subtraction, multiplication, division, and data movement, taking into account the sign, magnitude, and initial significance of incoming operands and producing the same quantities for the result of each arithmetic operation. The significance-mode library routines, in addition to producing the identical numerical results as the actual library routines, also provide significance estimates of computed function values.

The current system, which is primarily used for the testing of numerical procedures in programs considered logically correct, provides several kinds of pseudo-operations.

Initialization of significance values of program constants and variables may be accomplished during compilation (*"static initialization"*) or during execution (*"dynamic initialization"*). The user may specify the initial significances of constants and variables, expressed as numbers of either decimal or binary digits. An entity whose initial significance is not specified by the user is assigned a "default significance" by the system.

SYSTEM CONCEPT (SEE FIGURE 1)

The user program, with pseudo-operations inserted, is preprocessed by a special-purpose translator-compiler called the SigPac Scanner. The result is compilable FORTRAN source coding, consisting of the significance-indicating-arithmetic representation of the user program merged with CALLs on the SigPac run-time support library.

This program is compiled by the production FORTRAN compiler. The resulting object code is loaded and executed in accordance with user service requests, under control of OS/360, in local batch or remote batch mode. The SigPac system produces, in addition to selective numerical output as in normal execution, user-requested information on the numerical significance of selected quantities.

Selective output of symbolic names, number values, and significance information is provided through an appropriate SigPac output command which operates in conjunction with the conventional FORMAT statement.

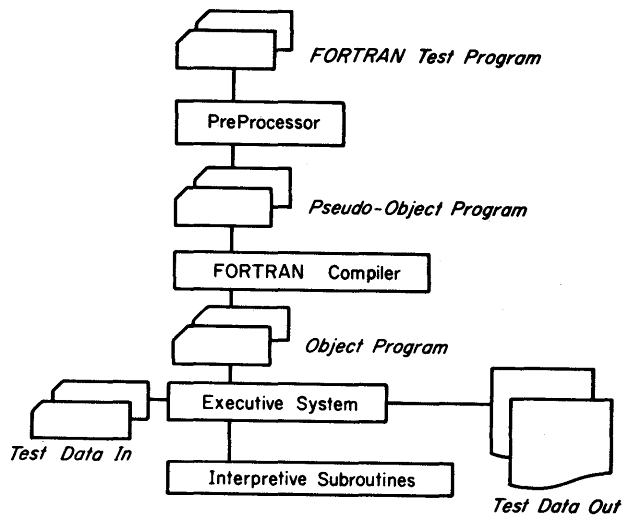


Figure 1—SigPac system concept
(Reproduced with permission, from Ref. 8)

The portion or portions of the source program which are to be executed in significance mode may be freely selected by the user at any point by inserting a SIGENTER pseudo-operation to cause the system to enter into, and a SIGEXIT, to exit from, significance mode. After exiting from significance mode the system reverts to normal program execution; it preserves all significance information, which becomes active again when significance mode is reentered.

SYSTEM LOGISTICS: TIME AND SPACE COSTS FOR USE

In principle, one would expect a system such as this (which executes machine coding in interpretive form to trace numerical significance) to be extremely costly in operation. For example, in the 360 version described here, a single arithmetic step requires up to 238 machine instructions to be executed using the present simple algorithms for significance prediction. (Remark: A part of this cost is due to the need for exact simulation of floating hexadecimal arithmetic.)

Space requirements for significance-mode calculations with this system are substantial if large arrays of numerical variables are operated upon, but are not prohibitive: for example, in the 360 system, SigPac assigns to each numeric value an additional four bytes (32 bits) of data space for storage of current significance index and related control information.

In practice, we have found that thoughtful examination of typical user programs frequently discloses that only minor segments are suspect with respect to generated error: examples are integrations, matrix arithmetic, and statistical analysis procedures. The use of the SIGENTER and SIGEXIT pseudo-ops to cause execution of the bulk of the user program in "normal" mode, and in Significance Mode only program segments* suspected to be data-sensitive, produces in such cases useful information at modest time and space cost. In meaningful examples, fully detailed manual tracing of error propagation would be infeasible because of time, cost, and the unpredictability that is inevitable in massive hand calculations.

For example, inversion (using conventional Gauss elimination with pivot search) of a real matrix of order 100 involves only a few million steps; if carried out in Significance Mode, such a procedure would take only several minutes on a high-speed machine, as compared to a few seconds in normal mode. Hand calculation of error would be infeasible.

For a very large calculation (e.g., solution of a difference equation representation of a hyperbolic differential equation system) in which essentially all of the problem's arithmetic must be performed in significance mode and which in normal mode requires hours of machine time per problem on a high-speed dedicated machine, we feel that SigPac usage would be inordinately costly. Such work would be economically attractive on future machines on which significance algorithms (which have been developed and rigorously evaluated through extensive experiments using the present SigPac system) could be micro-coded for high-speed operation at run time.

SIGNIFICANCE ALGORITHMS

The first-trial versions of the arithmetic significance algorithms, which are similar to those used by research workers in the field (e.g., (2)), are now described.

For multiplication and division, the significance of the result is equal to the significance of the less-significant argument. Thus,

$$S_R = \min(S_x, S_y) \qquad [1]$$

where S_R, S_x and S_y represent the significances of the result and arguments respectively.

The add-subtract algorithm requires information concerning not only the significances but also the size of the exponents e_x and e_y.

* Originally suggested to us by R. Danek of NASA-GSFC (cf. Ref. 8.)

Let $Z = |e_x - e_y|$ and $e_R =$ exponent of result. Then,

$$\begin{bmatrix} e_x > e_y \Rightarrow S_R = \min(S_y + Z,\ S_x) + e_R - \max(e_x, e_y) \\ e_x < e_y \Rightarrow S_R = \min(S_x + Z,\ S_y) + e_R - \max(e_x, e_y) \\ e_x = e_y \Rightarrow S_R = \min(S_x,\ S_y) + e_R - \max(e_x, e_y) \end{bmatrix} \qquad [2]$$

These basic rules are for machines employing normalized binary fractions and exponents in their floating point notation. It is of course necessary to keep track of some high-order zeros on floating hexadecimal computers such as the 360, which use a hexa-decimal exponent with a binary fraction normalized to the nearest hexadecimal digit, and on floating octal machines such as the Bn500.

Significance will be lost during multiplication if $N > P$, where

$$N = (LZ_R)_U + \min(S_x, S_y). \qquad [3]$$

$(LZ_R)_U$ is the number of leading zeros in the fraction of the result prior to normalization and P is the precision. $N - P$ bits are lost.

Additional details must be considered when handling the constant "zero." Zero may have full significance of input as data; may have any significance between full and none if calculated; and may have special meaning as a residual result. Exact numbers, if allowed, must also be treated independently (e.g., integer powers of floating-point numbers).

The rules [1], [2], and [3] produce statistical estimates intended to indicate "most probable number of valid digits" as a measure of current significance of each variable being traced.

Despite the simplicity of this set of algorithms, it will be seen below that initial test results were sufficiently realistic to be useful.

TEST CASE: COMPLEX ROOTS OF QUARTIC

The following example of a SigPac application was one of six that were used by NASA's Goddard Space Flight Center (GSFC) as tests of the validity of the approach and the accuracy of results.

This test program solves, by an iterative procedure, the polynomial equation $F(x) = x^4 + C_1x^3 + C_2x^2 + C_3x + C_4 = 0$, whose coefficients C_i are all real.

The following example is one of seventeen cases which were tested: $F(x) = x^4 - 5x^3 + 9.0001x^2 - 7.0003x + 2.0002 = 0$. For these coefficients, the exact roots are $1.0 \pm 0.01i$, 2.0, and 1.0.

The computer-produced roots with their system-generated significance indices are:

Real Component of Root	Significance (Dec. Dig.)
$\underline{9.99999999999}986386D-01$	12.0
$\underline{9.99999999999}986386D-01$	12.0
$\underline{1.99999999999999}98D+00$	14.8
$\underline{1.000000000001966}D+00$	16.0

Imaginary Component of Root	Significance (Dec. Dig.)
$-\underline{1.000000000000}243D-02$	12.6
$\underline{1.000000000000}243D-02$	12.6

The underlined digits are those which SigPac determined to be significant.

Significance predictions were produced by the system for the double precision numerical values of each of the seventeen sets of roots, corresponding to the seventeen sets of coefficients. In this example five of the six component *significance* estimates shown corresponded to the known *accuracy* of the respective quantities within one decimal digit; the sixth was optimistic by four digits. In all but 4 of the 102 components calculated, the predictions were within four digits; most were well within that range.

The relatively lax convergence criterion used with this relaxation solution was, it is felt, responsible for the fact that true accuracy of roots was lower than indicated arithmetic significance in every case. Consequently, we believe that the effective performance of the system in tracing arithmetic significance was probably even better than would be concluded by comparing, as we have done here, significance against accuracy, using initial trial algorithms.

Prof. B. F. Cheydleur originated the concept of significance arithmetic in 1949 and served as a consultant on our initial system development.

Prof. R. L. Ashenhurst consulted in development of the external-index significance algorithms for the basic operations and for library routines.

The initial implementation was performed with the sponsorship of NASA under contract NAS5-11705, using the IBM 360/95 computer at GSFC, implementing a suggestion by one of us.[8] Mr. J. D. Linnekin was Technical Project Officer and supplied the acceptance tests.

Future versions of the present system will provide for program execution using other kinds of error-indicating arithmetic, including the bounds-indicating Interval Arithmetic of Moore[6] and the signed-error-indicating "n, $n+1$" arithmetic of Moshos and Turner.[7] Also, the ability to perform non-trivial experiments will permit improvement of the significance-prediction algorithms.

REFERENCES AND NOTES

1 This categorization is due to Ashenhurst and Metropolis[2]

2 R L ASHENHURST N METROPOLIS
Error estimation in computer calculation
Amer Math Monthly 72/2 Part II Feb 1965 pp 47-58

3 J H WILKINSON
Rounding errors in algebraic processes
Prentice-Hall 1963

4 G E FORSYTHE
Pitfalls in computation, or why a math book isn't enough
Stanford University Technical Report No CS-147 January 1970 Available from Commerce Clearinghouse as AD-699 897

5 B F CHEYDLEUR
Binary notations in automatic computer algorithms and operation codes
3rd ACM National Conference 1949 Oak Ridge conference paper preprint only Also *Significance arithmetic notation and algorithms*
Unpublished internal memorandum Naval Ordnance Laboratory 1955 pp 1-19 (Available from ACM New York)

6 R E MOORE
Interval analysis
Prentice-Hall 1966

7 G J MOSHOS L R TURNER
Automatic estimates of computational errors
IEEE Conference Paper CP-63-1474 October 1963

8 H S BRIGHT
A proposed numerical accuracy control system
ACM Symposium on Experimental Applied Mathematics Washington D C August 1967 Proceedings published by Academic Press New York 1968 pp 314-334

Addendum (To Reference and Notes)

Authors' Comment: In late 1964, in consultation with B. F. Cheydleur (who (Ref. 5) had originated the Significance Arithmetic concept) we planned a Significance Arithmetic Processor for the Philco 213 computer, which was never built. In December 1966 we proposed an arithmetic-statement-only processor to Ford Motor Company. Earlier versions of the present paper were proposed to and rejected by: FJCC'70 (April 1970—"Trivial"); *SCIENCE* (June 1970—"Merely statistical prediction"); and *DATAMATION* (August 1970—"Too heavy"). The present paper was proposed (thanks to a suggestion by B. A. Galler) to SJCC'71 in August 1970 and was accepted after correction of errors.

On January 21, 1971, through Dr. Hans Oser of NBS, we learned that an Interval Arithmetic package using a program precompiler concept had been developed at the Mathematics Research Center, University of Wisconsin; the precompiler had been unpublished but was described in their May 1970 Internal Report 1065 by F. D. Crary and T. D. Ladner. We are grateful to Dr. J. M. Yohe of MRC for these details.

During the oral presentation of Reference 8, on August 7, 1967, Dr. Elizabeth Cuthill of the U. S. Naval Ship R&D Center offered the comment that the proposed precompiler concept would permit a user program to be executed in any kind of arithmetic. She suggested that the Interval Arithmetic of Moore and the multiple-precision concept as exemplified in the well-known JPL package would be particularly useful. Unfortunately, her remarks were not included in the publication of conference proceedings by Academic Press. If so, they might have been useful to the workers at MRC.

We feel that this situation illustrates the need for an early-publication process such as that which existed in 1961-63 through ACM (in "Research Summaries," *CACM*, edited by Mandy Grems) but which is not presently open to workers in this field. The oral presentation of the present paper, at the suggestion of Prof. Macon (Technical Program Chairman, SJCC'71) will be accompanied by an informal discussion by Dr. Yohe of the relationship between the work reported here and that at MRC/UW.

Automated interpretation and editing of fuzzy line drawings

by SHI-KUO CHANG†

IBM Thomas J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

One of the problems in automated line drawing analysis is to construct a cleaner drawing from the original drawing. For example, a chemist using an interactive chemical structure analysis system will usually draw a rough sketch of a chemical structure on a display device or a tablet. It is desirable to obtain a cleaner version of the rough sketch so that the cleaner drawing can be redisplayed. Moreover, the cleaner drawing can then be used to generate hard-copy output through a plotter, a magnetic film recorder or a photocomposer. Such an automated line drawing editing program can enhance the usefulness of an interactive system and is also of value in itself.

For a particular application, one usually can design a specific program to clean up a particular class of drawings. We can easily envisage a program for editing chemical structures, a program for editing a special kind of engineering drawings and so on. However, such special-purpose programs will not be able to edit a wide variety of different kinds of line drawings. In order to do so, we must design a general-purpose line drawing editing program.

In this paper we address ourselves to the problem of designing a general purpose line drawing editing program. First, we discuss the general philosophy of a table-driven line drawing editing program. The basic notion is to regard a line drawing as a *fuzzy program* and then to attempt to interpret the fuzzy program. The interpretation process is controlled by several tables. Therefore, the line drawing editing program is *table-driven* in the sense that by changing the tables the program can interpret different kinds of line drawings. It is analogous to a table-driven compiler. The main difference is that the program processed by a table-driven compiler has a *unique* interpretation, whereas a fuzzy program may have *several* interpretations and the line drawing editing program must be able to search for the correct interpretation. Next, a program for the automated editing of interconnected polygons is described. It is also shown how the *weight table* can be updated so that the average time of searching for the correct interpretations for a given collection of fuzzy programs is minimized. Several illustrative examples are given. Finally, some possible extensions of the present program are discussed. It is suggested that a hierarchically organized line drawing editing program may be able to interpret more complicated line drawings.

THE INTERPRETATION OF FUZZY LINE DRAWINGS

A *line drawing* is a collection of elementary line segments. The elementary line segments may consist of straight lines or simple analytic curves or more complicated curves. The set of all the different elementary line segments is called the *vocabulary* of the line drawings. For example, we may use straight lines in the $\pm 0°$, $\pm 45°$, $\pm 90°$, and $\pm 135°$ directions to construct line drawings called *chain-coded* line drawings.[1] Such chain-coded line drawings have a finite vocabulary $V = \{C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$, where C_i denotes a *directed line segment* on a finite grid whose length is $(\sqrt{2})^q$ and whose angle referenced to the x-axis is $ix(45°)^q$, $q = i \bmod 2$, as shown in Figure 1. Any chain-coded line drawing can be constructed using chains of elements in V. For example, the triangle shown in Figure 2 can be chain-coded as

$$C_0C_0C_3C_5$$

Since for automated processing of line drawings all line segments have parameters with limited precision, the vocabulary of the line drawings can usually be assumed to be finite. The vocabulary can be thought of as the 'building blocks' for the line drawings. Using the line segments in the vocabulary one can construct

† Present address: School of Electrical Engineering, Cornell University, Ithaca, New York.

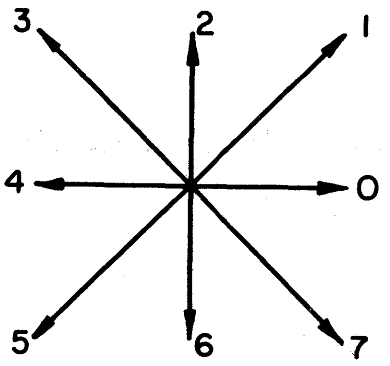


Figure 1—The eight basic line segments for the chain code

infinitely many different line drawings. However, for a given application only some of those possible drawings are of interest. For example, one may wish to generate only those drawings that resemble chemical structure diagrams. In such cases we impose additional constraints on the generation process. Generally speaking, there are two kinds of constraints: (1) *Prototype Generation Constraint* which specifies what prototype drawings are to be created, and (2) *Interconnection* or *Spatial Relationship Constraint* which specifies what kind of interconnections can be made between prototypes or what kind of spatial relationships must be satisfied between prototypes.

For example, for the generation of chemical structures, the prototypes are polygons. They are then interconnected to form more complicated structures. Thus the Prototype Generation Constraints (abbreviated to PGC) allow only the construction of polygons. The Interconnection Constraints (IC) allow all kinds of interconnection between polygons and the construction of tree-like structures.

As another example, for the generation of mathematical expressions, the prototypes are alphanumeric characters and special function symbols. They must satisfy certain spatial relationships to form valid expressions. Thus the PGC's allow the construction of alphanumeric characters and special symbols, and the Spatial Relationship Constraints (SRC) allow the creation of valid mathematical expressions.

For the automated analysis and generation of line drawings, such constraints (PGC, IC and SRC) can best be expressed by formal descriptive grammar rules. In Reference 2 we have discussed the use of hierarchical two-dimensional grammars for the analysis and generation of drawings. In Reference 3 the problem of analyzing two-dimensional mathematical expressions is treated. It suffices to say here that formal descriptive techniques have been used with some success to analyze and generate (artificial) line drawings. However, in all cases considered, the drawings are assumed to be ideal or close to ideal. It is to the analysis of less ideal, or fuzzy, drawings that we should now focus our attention.

First of all, it is obvious that line drawings such as hand-printed or hand-written characters, chemical structures, engineering sketches are all non-exact or fuzzy. Fuzziness comes in at two levels: (1) instead of a finite vocabulary, an infinite vocabulary is used, (2) instead of well formed exact structures, non-exact structures are created. We will discuss these two points in some detail.

In the automated analysis of line drawings we assume that the vocabulary is finite. For example, in the chain-coded drawing we assume that all the line segments are in the eight prescribed directions. However, in the original drawing there might be curved line segments or straight lines in other directions. In order to obtain the chain-coded version, the original drawing has to be interpreted or, in usual terminology, quantized. *Quantization* is thus the process in which the original infinite vocabulary is reduced to a finite vocabulary so that subsequent analysis of the drawing can be facilitated. Quantization is usually regarded as a preprocessing operation. Nevertheless, an incorrect quantization will probably result in an incorrect analysis. Thus we prefer to call the vocabulary reduction process the *interpretation process,* in which the 'meaning' of individual line segments is decided by *conditional* (i.e., context dependent) *quantization* of the original line segments.

This point of view can be reformulated as follows. We can regard the original line drawing as a *fuzzy program,* the *fuzzy instructions* being the individual fuzzy line segments. Our objective is to interpret the fuzzy program so that we can construct the corresponding non-fuzzy program (non-fuzzy line drawing). Thus the problem becomes one of interpreting fuzzy programs. If the interpretation is correct, then the non-fuzzy program is *executable* in the sense that it does not violate any of the prototype generation constraints. If the interpretation is incorrect, then some of the con-

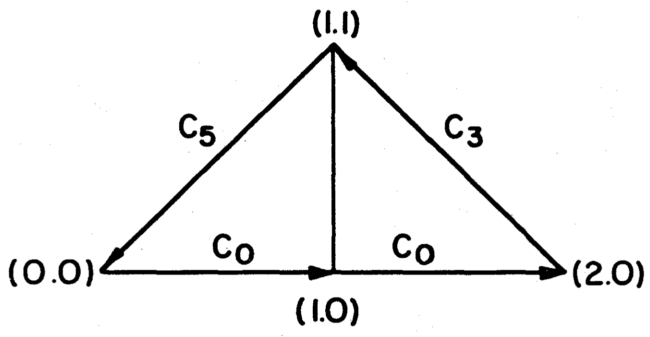


Figure 2—A chain-coded triangle

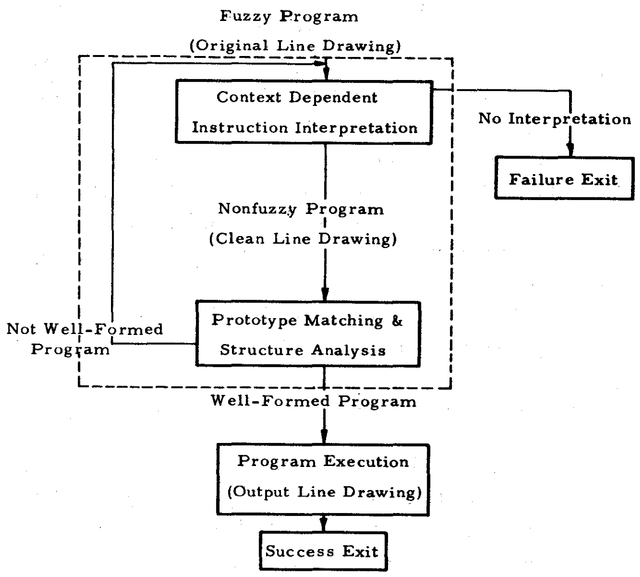


Figure 3—Automated program interpretation system

straints are not satisfied and the non-fuzzy program is *not executable*. The second kind of fuzziness can also be taken care of using the formulation just described. The interconnection constraints and spatial relationship constraints determine the *well-formedness* of the non-fuzzy line drawings. Therefore, if the interpretation is correct, then the resulting non-fuzzy program (line drawing) should have a well-formed structure. An incorrect interpretation will generate a program (line drawing) having no well-formed structures.

Therefore, our objective can be stated as follows: find the non-fuzzy program corresponding to a given fuzzy program such that the resulting non-fuzzy program is executable (i.e., it satisfies all the constraints).

In summary, the system we propose is diagrammatically represented in Figure 3. The input fuzzy pro-

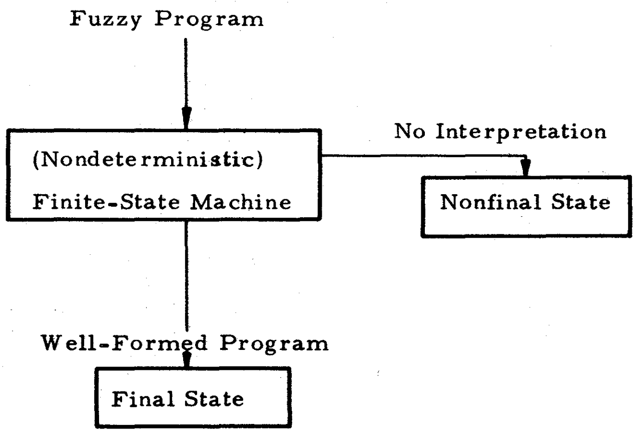


Figure 4—Equivalent theoretical model

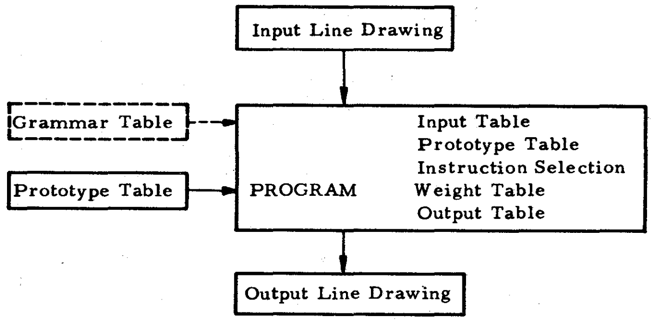


Figure 5—Automated line drawing editing program

gram is first interpreted. If the resulting non-fuzzy program is well-formed, then it is executed (i.e., the clean line drawing is sent to the output device). Otherwise another interpretation pass is entered. When all possible interpretations have been tried without success, the failure exit can be taken. This is the "No Interpretation" case.

It should be mentioned that the theory of fuzzy programs had been studied in detail.[4] The equivalent theoretical model for the system shown in Figure 3 is illustrated in Figure 4. The (non-deterministic) finite-state machine represents the interpretive process enclosed in the dotted-line-block in Figure 3. In this particular case the programs are line drawings, and the constraints are incorporated into the finite state machine so that the machine stops in a final state if and only if the fuzzy program satisfies all the imposed constraints. In Reference 4 we have treated the problem of executing fuzzy programs using finite-state machines.

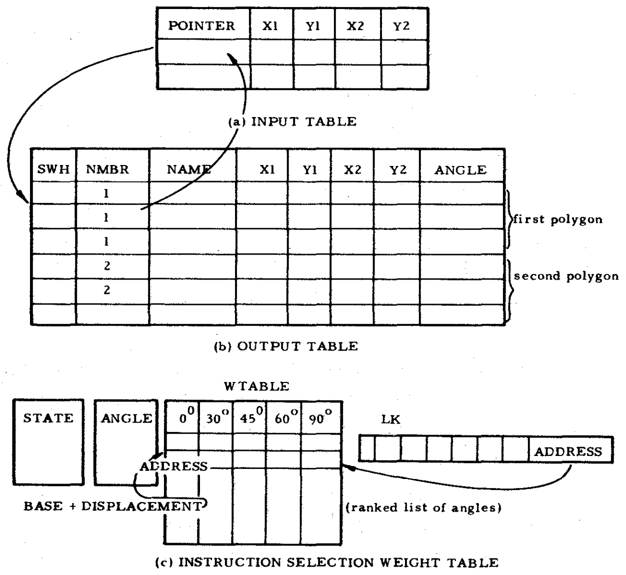
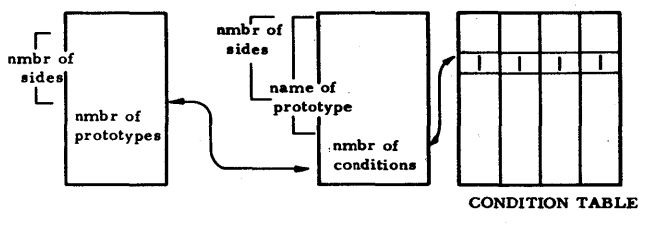
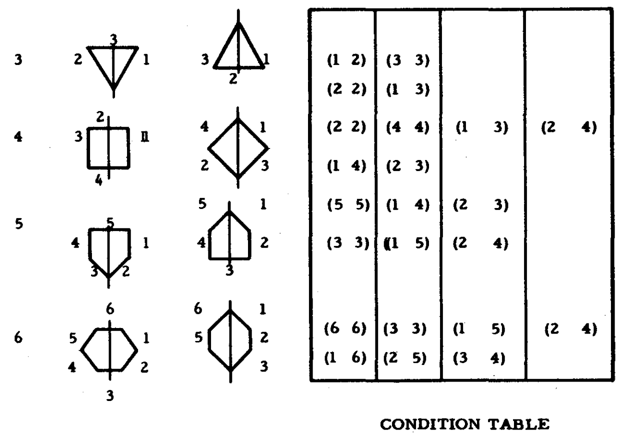


Figure 6



(a) PROTOTYPE TABLE



(1 2)	(3 3)		
(2 2)	(1 3)		
(2 2)	(4 4)	(1 3)	(2 4)
(1 4)	(2 3)		
(5 5)	(1 4)	(2 3)	
(3 3)	(1 5)	(2 4)	
(6 6)	(3 3)	(1 5)	(2 4)
(1 6)	(2 5)	(3 4)	

(b) Polygon Prototypes and their Descriptions

Figure 7

We have shown that if the fuzzy program is regular, then one can decide whether it has a non-fuzzy execution and, in case it does, one can effectively construct the corresponding non-fuzzy program. Therefore, from a theoretical point of view, we can always interpret (or reject) a fuzzy program provided that it is regular. Since line drawings can always be regarded as regular expressions, theoretically their interpretations can always be found. However, from a practical point of view, we not only want to find the correct interpretation but also would like to find it in the shortest time possible. In other words, efficiency in finding the correct interpretation is also of importance. This problem will be discussed in the next section.

AN EXPERIMENTAL PROGRAM FOR LINE DRAWING INTERPRETATION

In the above we have discussed the general concepts of fuzzy program (fuzzy line drawing) interpretation. In this section an experimental program for the interpretation of interconnected polygons will be described.

The experimental line drawing interpretation program is organized as shown in Figure 5. The program is initialized by reading in a table of (preconceived) prototypes and another table of initial weights for fuzzy instruction interpretation. Once properly set up, the program can accept fuzzy line drawings and attempt to interpret them. If the interpretation is successful, then the weight table will be updated so that successful interpretations are reinforced. The overall effect of the updating procedure is such that the average search time for a given collection of fuzzy programs will be minimized. Thus we may say that the program is able to *learn* the idiosyncracies of the individual user and tries to adjust its interpretation of his drawings accordingly.

In the following paragraphs we describe the experimental program in some detail.

First we describe the four basic tables of the program.

(1) INPUT TABLE: An array of size N by 5, where N is the number of line segments of the input drawing. In any row, the first entry contains a pointer to the corresponding line segment in the output table (initially 0), the remaining four entries are the x and y coordinates of the starting and terminating points of the line segment. See Figure 6(a).

(2) OUTPUT TABLE: An array of size M by 8, where M is the number of line segments of the output drawing ($\neq$N in general). In any row, the entries are SWH (1 if fixed and 0 if free), NMBR (k if this line segment is in the kth polygon), NAME (a pointer to the corresponding line segment in the input table), X1, Y1, X2, Y2 (x and y coordinates of the starting and terminating points of the line segment) and ANGLE

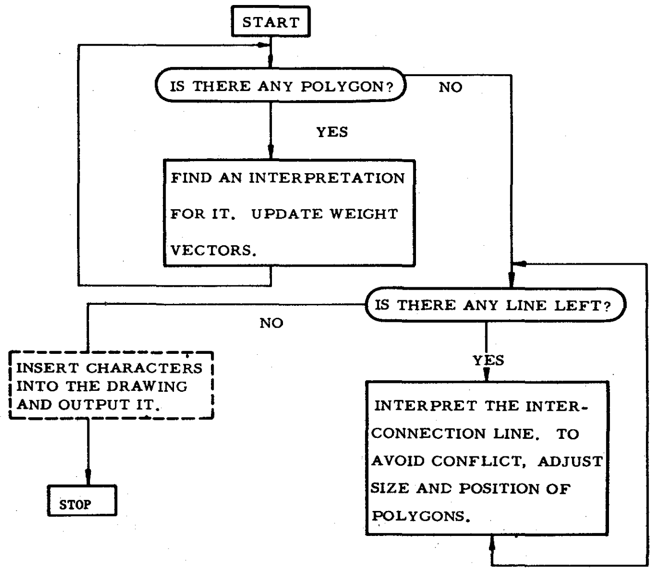

START
IS THERE ANY POLYGON?
NO
YES
FIND AN INTERPRETATION FOR IT. UPDATE WEIGHT VECTORS.
IS THERE ANY LINE LEFT?
NO
YES
INSERT CHARACTERS INTO THE DRAWING AND OUTPUT IT.
STOP
INTERPRET THE INTERCONNECTION LINE. TO AVOID CONFLICT, ADJUST SIZE AND POSITION OF POLYGONS.

Figure 8—Flowcharts for the experimental program. For clarity error exits are not shown

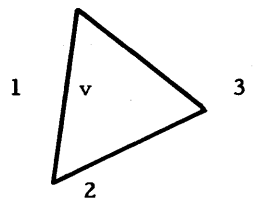
(1 if 0°, 2 if 30°, 3 if 45°, 4 if 60°, 5 if 90°). See Figure 6(b).

(3) PROTOTYPE TABLE: This table is organized as shown in Figure 7(a). Given the number of sides of a polygon, one first finds out the number of prototypes for this polygon, then one can find out the number of conditions for a particular prototype, finally the conditions for this prototype are found in the CONDITION TABLE. To see how the prototypes are described, in Figure 7(b) a number of prototypes and their corresponding descriptions are given. Each pair (I, J) says in effect that side I should be symmetric to side J with respect to a hypothetical axis (the program will use this description to generate both x-axis and y-axis symmteric polygons), and pair (I, I) simply specifies that side I is perpendicular to the axis in question. We can use this description to describe axial symmetric or complete symmetric polygons.

(4) INSTRUCTION SELECTION WEIGHT TABLE: This table is organized as shown in Figure 6(c). Given the *state* (i.e., the context in which it is located, e.g., a triangle, a rectangle, etc.) and the angle of a (fuzzy) line segment, one can search the STATE array and the ANGLE array to find out the BASE address and the DISPLACEMENT. In our case the *state* of a line segment is the number of branches of the polygon of which it is a part (the state is 1 if this line segment is not in any polygon). BASE+ DISPLACEMENT gives us the ADDRESS of a row in the WTABLE array. The weights in this row vector determine which of the five angles (0°, 30°, 45°, 60°, 90°) is most preferable when the fuzzy line segment has the given angle and is in the given state. The angles can be ordered according to their weights, larger weights being more preferable. One can thus construct a ranked list of angles for each fuzzy line segments, which is stored in the LK array. The last entry of the LK array is the ADDRESS of this weight vector. This link is provided because once we have selected an interpretation and chosen an angle in the LK array, we can then go back to WTABLE to update the weight vector.

Now we can explain the interpretation algorithm. The flowchart of the interpretation algorithm is shown in Figure 8. The main strategy is to "treat the polygons first, then handle the interconnections." The program first locates a polygon in the INPUT TABLE. It then uses the INSTRUCTION SELECTION WEIGHT TABLE to construct LK tables (ranked lists of angles) for all line segments in this polygon. Thus for each LK table there is a line segment. The program next looks at all possible interpretations of the polygon by testing whether the conditions of the prototypes in the PROTOTYPE TABLE are satisfied or not. From those possible interpretations, the correct one (the one satisfying all the constraints) is selected. The angles of the line segments are now fixed. The clean polygon is next constructed with respect to the selected interpretation. It is then stored in the OUTPUT TABLE.

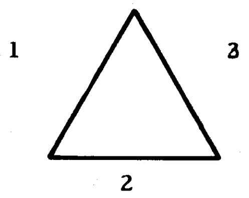
For example, the triangle shown in Figure 9(a) is fuzzy. Its line segments have LK tables as shown in Figure 9(b). The best interpretation is found to be ((2 2), (1 3)), and the clean triangle is shown in Figure 9(c). Notice that the angles are first determined so that the orientation of the polygon becomes fixed. The location and length of the line segments can then be deter-



(a) Original Fuzzy Triangle

Line 1	60°	$\underline{45^\circ}$	90°	30°	0°
Line 2	30°	$\underline{0^\circ}$	45°	60°	90°
Line 3	$\underline{45^\circ}$	30°	60°	0°	90°

(b) Instruction Selection from LK Tables



(c) Clean Triangle with Interpretation (2

Figure 9

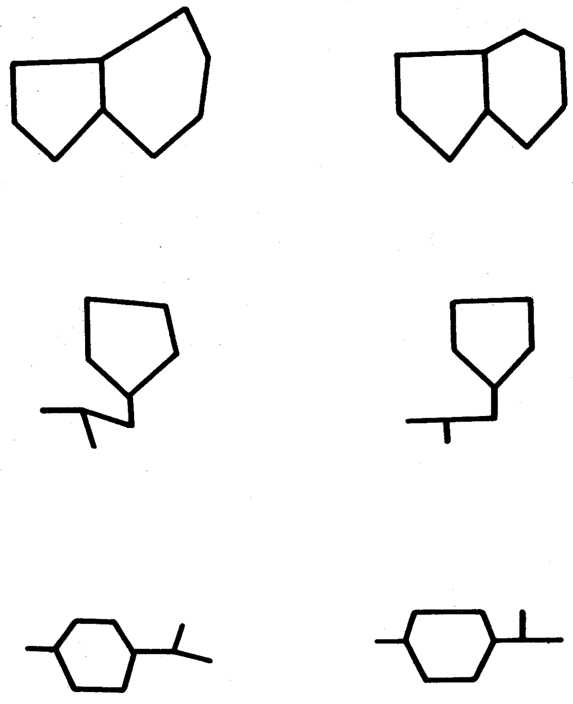


Figure 10—Examples on line drawing interpretation

mined by rotating, extending or shrinking the original line segments.

After a polygon has been successfully interpreted, WTABLE is updated so that such an interpretation is reinforced. The *updating procedure* is the following. Suppose $l_1, l_2, \ldots, l_n$ are the line segments that have just been interpreted. Let ADDRESS_i be the entry in WTABLE corresponding to l_i, ANGLE_i be the correct interpretation of the angle of l_i, and COST_i be the search cost incurred if the angle of l_i is not interpreted as ANGLE_i. Then for all i, $1 \leq i \leq n$,

$$\text{WTABLE}(\text{ADDRESS}_i, \text{ANGLE}_i) \leftarrow \text{WTABLE}(\text{ADDRESS}_i, \text{ANGLE}_i) + \text{COST}_i.$$

In other words, if the interpretation of the angle of l_i is ANGLE_i, then $\text{WTABLE}(\text{ADDRESS}_i, \text{ANGLE}_i)$ is increased by COST_i. Thus correct interpretations are reinforced. The updating procedure is simple and intuitively appealing. Its theoretical justification can be found in Reference 4, where it is shown that this procedure actually minimizes the average search time for a given collection of fuzzy programs.

The program repeats the above procedure for every polygon in the line drawing. Since the polygons may touch one another, some sides of the second (or third, . . .) polygon may have already been determined. In this case they are treated as fixed (i.e., SWH = 1 in its row in OUTPUT TABLE) and a similar procedure is carried out for the free line segments.

When all the polygons have been treated, the program then handles the interconnection line segments. Notice that since the 'state' is now different, by changing the BASE address one can use a different WTABLE for those line segments. For example, only 0°, 45°, 90° lines are allowed. Thus although the angles are the same, with different contexts (different states) the interpretation can be different. When the interconnection line segments are fixed, the position and size of the polygons can be adjusted, so that no conflict occurs (no overlapping polygons, no lines piercing into polygons, etc.) in the clean line drawing.

The present version of the experimental program, written in FORTRAN, is able to interpret interconnected polygons. Some of the results are illustrated in Figure 10 and Figure 11. The drawings are produced by a plotter. The drawings shown on the left are the originals and those on the right are the clean versions. It is clear that the polygons are symmetric with respect either to the x-axis or to the y-axis, and the interconnected lines have also been adjusted. One obvious application of this program is the editing of hand-drawn chemical structures. However, since the prototype table can be modified at will, the program is not restricted to

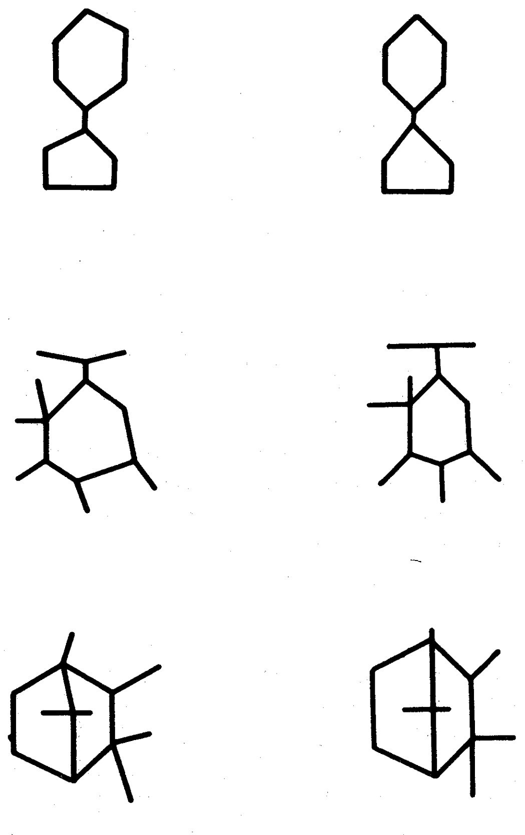


Figure 11—More examples on line drawing interpretation

this application and may be used to interpret other kinds of line drawings. For chemical structure editing, in the last phase one can insert alphabetical characters into the line drawing to make it complete. This phase is also indicated in the flowchart (see Figure 8), but has not been incorporated into the experimental program.

DISCUSSIONS AND CONCLUSIONS

In the previous sections we have described a general approach to fuzzy line drawing interpretation and a specific experimental program for the interpretation of interconnected polygons. The basic notion of our approach is to regard a fuzzy line drawing as a fuzzy program and to attempt to interpret the fuzzy program. The interpretation is found by testing whether the fuzzy program (or part of the fuzzy program when a local interpretation is possible) satisfies certain constraints. The constraints are either Prototype Generation Constraints or Spatial Relationship Constraints. In the experimental program the first kind of constraints are incorporated into the PROTOTYPE TABLE. The second kind of constraints are built into the program: spatial conflict is avoided by adjusting the size and location of the polygons. For a more general system, it would be more desirable to state all constraints explicitly. The spatial constraints can then be coded as grammar rules and read into the program, as shown by the dotted box in Figure 5. Moreover, the flowchart shown in Figure 8 indicates that the program consists of two steps:

(1) The interpretation of prototypes.
(2) The organization of prototypes into higher level structures.

In our case the prototypes are polygons, and the higher level structures are interconnected polygons. For more complicated line drawings, one may iterate the two steps to obtain higher and higher level structures. In each level, there may be a different set of Prototype Generation and Spatial Constraints. Therefore, we have a collection of constraints organized into levels or hierarchies. This is the concept of a hierarchical grammar.[2] In the present experimental program we have only two levels. The techniques developed are also applicable to a multi-level analysis and interpretation program. Such extension seems to be both theoretically interesting and practically useful.

In conclusion, the table-driven line drawing editing program described in this paper is but one small step toward the complete automation of fuzzy line drawing analysis. Two main concepts have been introduced. First, tables are used to control the interpretation process, so that the program can be used to interpret different kinds of line drawings. Second, a weight updating procedure is included in the program so that the average search time for a given collection of fuzzy programs can be minimized. However, the program is still not general enough to interpret a wide variety of fuzzy line drawings. It is suggested that a hierarchically organized program will be useful to interpret more complicated drawings. Experimental data should also be gathered to see whether the weight updating procedure indeed improves significantly the performance of the program. We hope that this preliminary investigation will lead to more interesting development in the future.

REFERENCES

1 FREEMAN GLASS
On the quantization of line-drawing data
IEEE Trans on Systems Science and Cybernetics Vol SSC-5 No 1 January 1969

2 S K CHANG
The analysis of two dimensional patterns using picture processing grammars
Conference Record Second ACM Symposium on Theory of Computing May 4-6 1970 Mass pp 206-216

3 S K CHANG
A method for the structural analysis of two dimensional mathematical expressions
Information Sciences 2 July 1970 pp 253-272

4 S K CHANG
Fuzzy programs—Theory and applications
Proc of the Polytechnic Institute of Brooklyn XXI International Symposium on Computers and Automata Brooklyn New York April 1971

Computer graphics study of array response

by GEORGE W. BYRAM, GEORGE V. OLDS and LEON P. LA LUMIERE

Naval Research Laboratory
Washington, D. C.

INTRODUCTION

Most sonar systems as well as some radar systems use a beam-formed array as their primary sensor. An array of small receptors is the most convenient means of sampling an extended field. It is often the only feasible means in large scale, long wavelength applications such as sonar.

Since the individual receptors of a large array cannot be moved, control of the array directional sensitivity must be introduced through processing of their output signals. The beam former performs this task by altering the relative delays and amplitudes of individual receptor outputs before summing them to form an array output.

The basic geometric variables involved in array response are shown in Figure 1. The special case of a linear array has been chosen for simplicity. Even in the more general case of a planar or volume array, only two main directions will be of interest. These are the actual direction of signal arrival and the desired, or beam-formed, direction.

The response of an array-beam former combination can be computed quite easily. The large number of parameters involved, however, complicates interpretation of numerical results. The use of three dimensional computer plotting techniques permits display of the response in a form which shows the influence of a parameter directly. This method greatly aids insight into array properties. It also affords a considerable degree of insight into some commonly used array pattern computation techniques. This has proved to be a useful teaching tool.

The following examples have been chosen to illustrate some of the more important factors affecting array response. They also illustrate the influence of format and parameter choices on the clarity of the resulting plots.

ARRAY RESPONSE EXAMPLES

Sampling and frequency response

A minimum length for an array is often dictated by a specified maximum width for its main lobe. If the

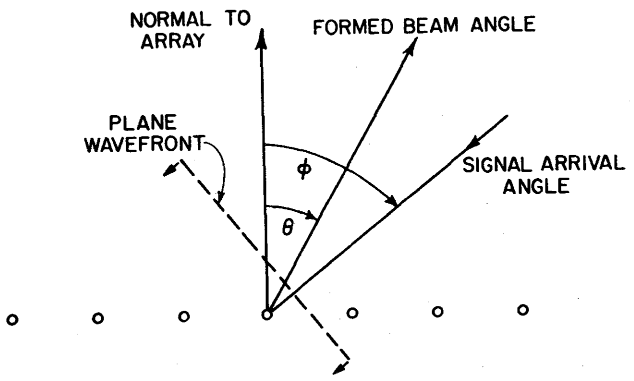


Figure 1—Geometric variables involved in array response

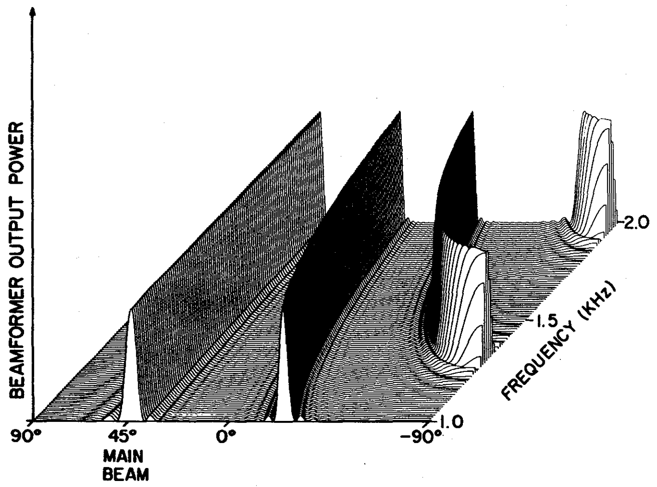


Figure 2—Response of 14 element equispaced array

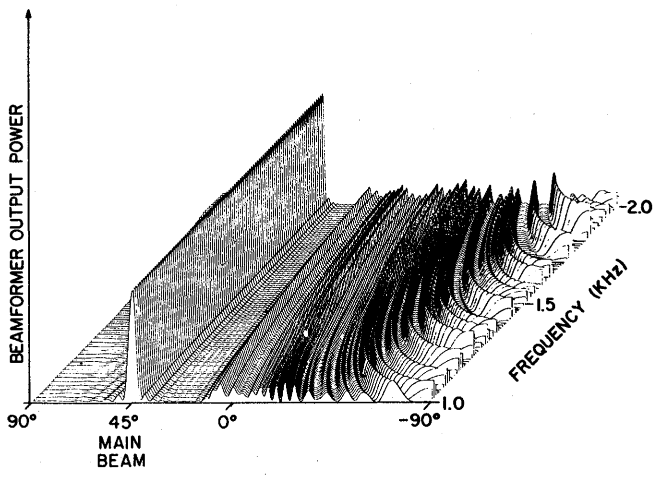


Figure 3—Response of perturbed array

number of receptors is limited their spacing may be greater than a half wavelength and the array will be undersampled. A periodic, undersampled array does not gather sufficient independent information to permit unique determination of the direction of arrival of a low bandwidth signal. Hence, strong grating lobes, arising from cycle to cycle ambiguity, will be present if the elements are equally spaced. The problem is less severe if the signal has high bandwidth or if the array has non-uniform spacing.

The frequency dependence of the response of a 14 element equispaced array is shown in Figure 2. The element spacing in this array is a half wavelength at approximately 500 Hz. There are two grating lobes at 1 KHz and four at 2 KHz. If the spacing is perturbed to disrupt the periodicity the grating lobes will be spread into many smaller lobes. Figure 3 shows the result of adding linearly increasing multiples of ten percent to the spacings, starting at one end. Although the array is now more severely undersampled, the largest side lobe is five or six dB down from the main lobe. The reduced width of the main lobe results from the increased length of the array. The side lobe reduction would have been slightly improved if the perturbed spacing had been rescaled to the same total length as the equispaced array. For a fixed length and a fixed number of elements a point of diminishing returns is reached quite rapidly and extensive search for optimum spacing schemes yields only slight further improvement.

Wavefront curvature and focussing

The computations for Figures 2 and 3 assumed a plane wavefront. If a source is quite close to the array, however, the curvature of the wavefront can be significant. In some cases it is possible to focus the array to favor a specific range.

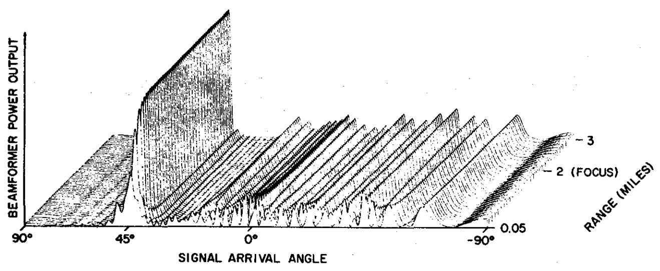


Figure 4—Effect of wavefront curvature on array response

Figure 4 shows the degradation in response of the perturbed array at 1.5 KHz as the range becomes much less than the focal distance. For extremely small ranges the main lobe is greatly reduced but there is only a slight increase in side lobe level. This is to be expected since the phase errors resulting from wavefront curvature are no worse than those already existing for signals arriving from non-beam-formed directions.

The extremely small delay differences involved in focussing an array at any reasonable distance would be lost in the delay fluctuations caused by local propagation effects. Hence the pattern degradation at small ranges is the most significant result shown in Figure 4. The very weak dependence of the pattern on range and focal distance for larger values indicates that the plane wave assumption is justified in almost all practical cases. To show the effects of wavefront curvature at and beyond the focal distance it is necessary to take an extreme case.

Two arrays are shown in Figure 5. Both are $3\frac{1}{2}$ wavelengths long and each is focussed on a point 5 wavelengths from its center. The linear array is focussed by adding extra delay in the beam former for the inner elements. The curved array has been focussed by bending it to match the curvature of the desired wavefront. Figure 6 compares the response of these two arrays. The response patterns are almost identical. The fast rise to maximum response and the slower fall off as range exceeds the focal distance are typical for such short focal distances.

Planar arrays

The arrays examined thus far have been mainly linear arrays. Although linear arrays are the easiest type to construct, they provide no directivity in those

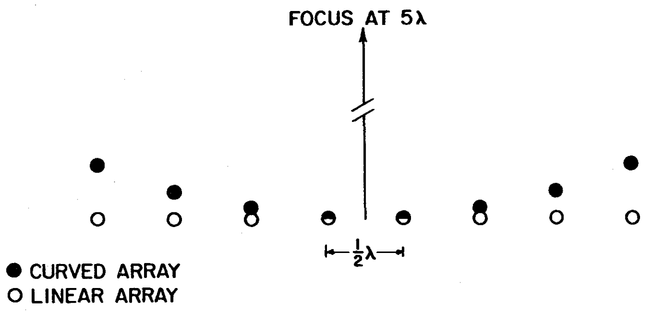


Figure 5—Curved and linear eight element arrays

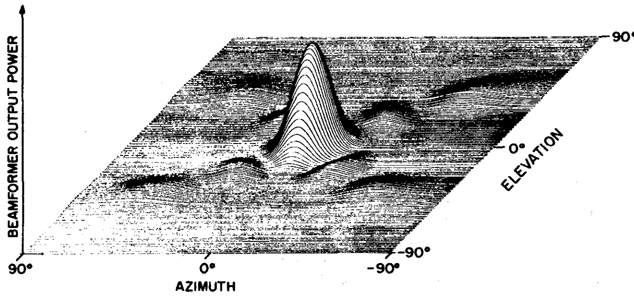


Figure 7—Response of "four by four" sixteen element planar array

planes to which the array is normal. At higher frequencies it becomes practical to use planar and volume arrays.

Figure 7 shows the response in azimuth and elevation of a four-by-four, sixteen element planar array. The main beam is formed broadside to the array. The third variable in such a plot is used up by the additional direction variable. Hence, to show the dependence on an additional parameter it would be necessary to choose some slice through the response surface as a starting point. It is more convenient in most cases to make a sequence of plots instead.

Figure 8 shows the result of increasing the frequency by 50 percent. The central portion of this plot corresponds to the previous plot. As the frequency becomes higher these side lobes will move inward toward the center of the plot and be joined by others coming in from the edges. Figure 9 shows the response at three times the original frequency. The array is now severely undersampled and the side lobes show a periodic arrangement. The increased width of the more extreme side lobes arises from the smaller effective aperture which the array presents to an oblique arrival.

Pattern multiplication

The planar array response shown in Figure 9 was computed directly. Some computation time could have been saved, however, by the use of pattern multiplication (1). The technique is sufficiently useful that it is worthwhile to repeat the pattern computation of Figure 9 by pattern multiplication, with illustrative plots of the intermediate steps.

Figure 10 shows the 16 element planar array and two possible pairs of subarrays which could be used for pattern multiplication. The dimensions shown in wavelengths correspond to the frequency used for the plot of Figure 9. The linear subarrays are the preferred choice because their patterns will be mirror images of each other in elevation-azimuth coordinates. Hence, only one pattern need be computed and stored. The

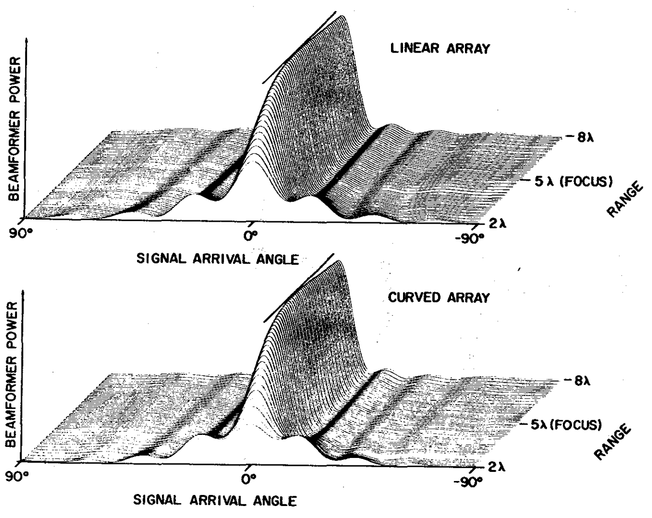


Figure 6—Example of array focussing

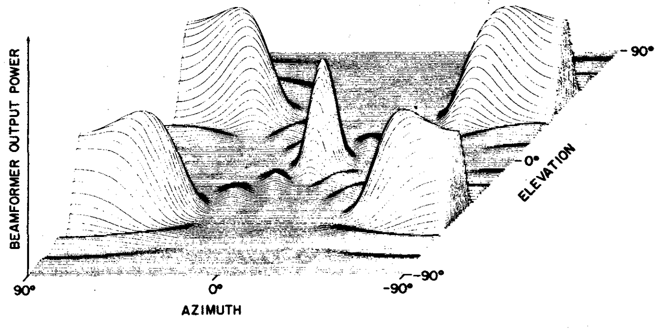


Figure 8—Response of "four by four" planar array

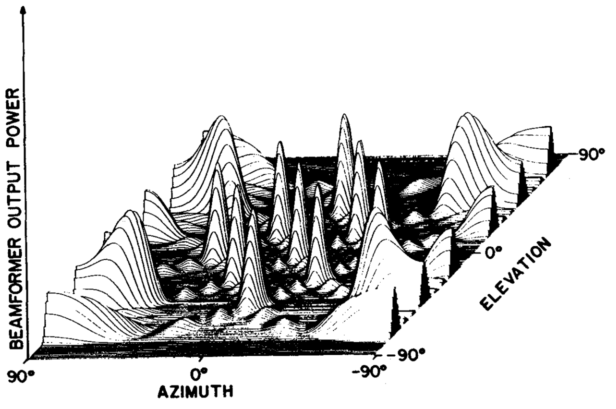


Figure 9—Response of "four by four" sixteen element planar array

required product can be obtained by taking successive multipliers and multiplicands from locations starting at opposite corners of the stored pattern.

The patterns of the two linear subarrays are shown in Figures 11 and 12. The mirror image relationship is clearly visible. These patterns are figures of revolution with their respective arrays as axes. This is not obvious in the figures because the use of elevation azimuth coordinates is equivalent to projecting the patterns on the surface of a sphere. The product of these two patterns, point by point, is the pattern of Figure 9. The contributions of the two individual patterns can be recognized more easily looking at the underside of the pattern as shown in Figure 13.

The lack of perspective correction in the plotting is quite a bit more conspicuous in the inverted plot because the inverted base plane provides a frame of reference for the eye.

The patterns of the other two possible subarrays are shown in Figures 14 and 15. The small two by two array possesses the same symmetry and same element

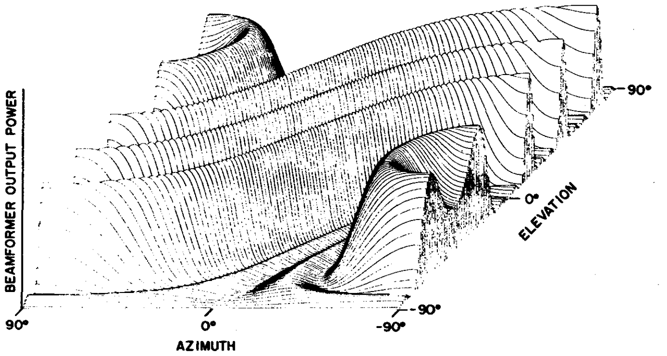


Figure 11—Response of left four element linear array

spacing as the complete array. Hence its pattern is quite similar to that of the complete array although broadened by the diffraction resulting from its smaller aperture.

The larger two by two subarray is quite severely undersampled and hence has a large number of grating lobes. The lobes of the small array tend to select the lobes of the larger array visible in the resulting product pattern. The straight lines across the front and rear of the pattern of the larger two by two array merely indicate the absence of nulls in the response at $\pm 90°$ elevation.

The use of pattern multiplication is very advantageous for large arrays having a high degree of symmetry or periodicity. The process can be extended by factoring a large array into many small arrays. For a completely periodic array, the gain in computational efficiency is similar to that which the fast Fourier transform affords over conventional techniques. This is to be expected since the far field pattern is the Fourier transform of the element distribution.

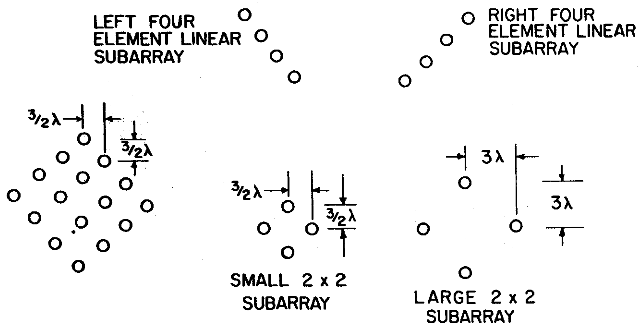

LEFT FOUR ELEMENT LINEAR SUBARRAY
RIGHT FOUR ELEMENT LINEAR SUBARRAY
3/2λ
3/2λ
3/2λ
3/2λ
3λ
3λ
SMALL 2 x 2 SUBARRAY
LARGE 2 x 2 SUBARRAY

Figure 10—Sixteen element planar array and two pairs of subarrays

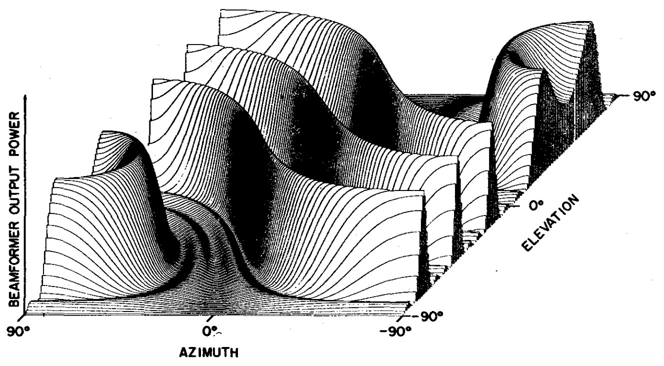


Figure 12—Response of right four element linear array

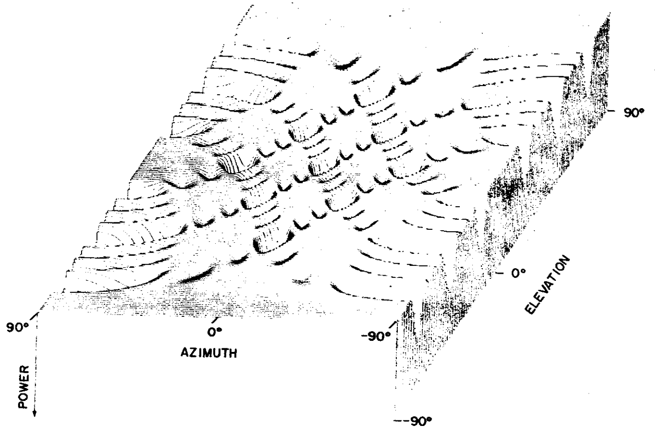


Figure 13—Response of planar array (inverted)

Volume arrays

The large number of elements in most volume arrays makes the use of pattern multiplication almost essential. The pattern of a four-by-four-by-four, 64 element volume array consisting of a stack of four of the arrays shown in Figure 10 can be obtained by one additional pattern multiplication. Figure 16 shows the end fire pattern of a four element linear array seen end-on in elevation azimuth coordinates. The figure of revolution nature of a linear array pattern is much more easily seen in this orientation. Multiplication of this pattern by the planar array pattern of Figure 9 results in the volume array pattern of Figure 17. The side lobe selection effect of the multiplication is again quite evident.

CHOICE OF PARAMETERS FOR ARRAY PLOTS

These examples indicate the insight into array properties obtainable through the use of computer plotting

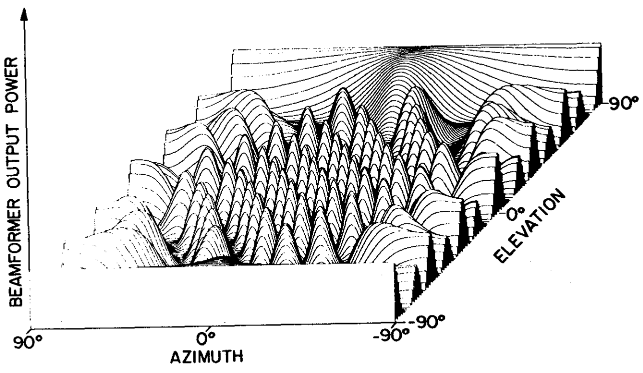


Figure 15—Response of large "two by two" array

techniques. Considerable care in the choice of parameters is required, however, to produce a meaningful and easy to interpret plot. One of the three available variables in a 3-D plot is used for the array response. Since directional sensitivity is the *raison d'etre* of an array it is generally essential to include all geometric variables describing signal arrival direction. In the case of a planar or volume array this uses up the remaining two variables. In the case of a linear array, however, only one direction variable is required. The third variable can be selected from quantities such as: frequency, amplitude shading parameter, beam steering angle, or wavefront curvature.

The orientation of a 3-D plot is quite important. The ridged surface obtained in frequency response plots such as Figures 2 and 3 would be extremely difficult to interpret if the axes used for signal arrival angle and frequency were interchanged. It would be almost impossible for the eye to separate individual ridges in the confusion of many lines almost parallel to the front of the plot.

It is also important that detail of interest in the

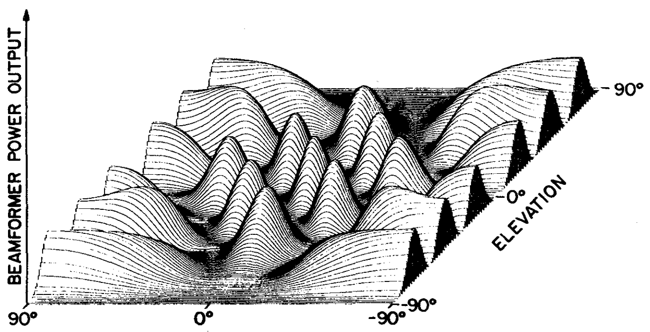


Figure 14—Response of small "two by two" array

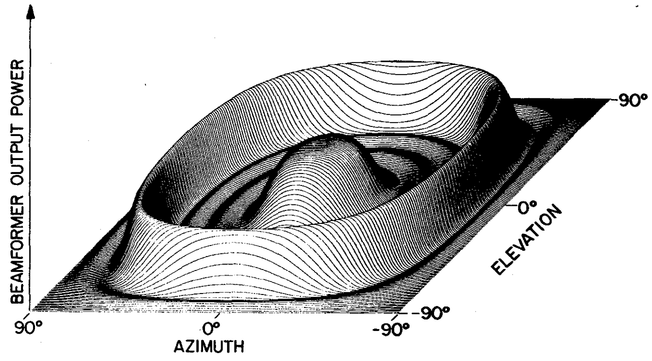


Figure 16—Endfire response of four element linear array

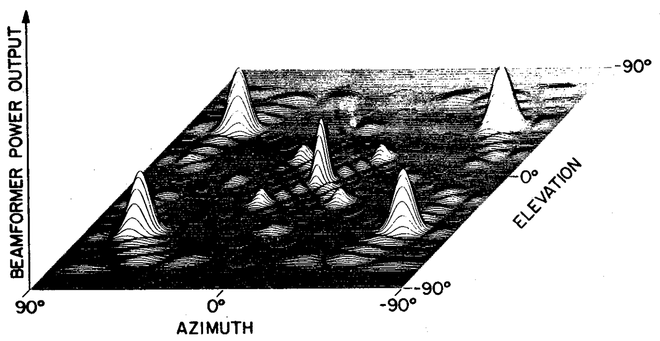


Figure 17—Response of "four-by-four-by-four" volume array

center and rear of a plot not be hidden by higher features in the foreground. It is often advantageous to reverse the frequency axis and put the lower frequencies toward the rear of the plot if the response extends to extremely low frequencies. This prevents the very broad main lobe at low frequencies from hiding points farther back into the plot. The entire first line of an elevation-azimuth plot corresponds to a single direction since the poles of such a coordinate system are singular. If the array has a significant response in that direction a straight line across the front of the plot will hide detail farther back. This effect is visible in Figure 15. Choice of a different coordinate system or a slight change in the orientation or steering of the array to provide a null in that direction can be helpful.

The planning of a 3-D plot is greatly aided by plotting a few selected slices of the pattern. This is one area in which a fast display and a high degree of man-machine interaction would be extremely useful.

General extrapolation plots (2) in frequency and beam angle for linear arrays are of aid in selecting parameters for 3-D plots in these variables.

REFERENCES

1 J D KRAUS
Antennas
pp 66-74 McGraw-Hill 1950

2 G W BYRAM G V OLDS
Computer display of array response
78th Meeting of the Acoustical Society of America
San Diego Cal

Computer manipulation of digitized pictures

by NATHANIEL MACON and MAXINE KIEFER

The American University
Washington, D.C.

INTRODUCTION

To an increasing degree, equipment is available which is capable of converting photographic information into machine readable form or converting computer files into a visual image more or less resembling a half-tone picture, thus rendering photographic and other pictorial information available as data for processing by digital computers. Three major directions of effort have ensued. Necessary utility routines have been developed for managing I/O, file manipulation, and the implementation of languages to facilitate programming effort; analytical work toward character and pattern recognition, and toward parametric characterization of pictures has led to algorithms for accomplishing various sorts of mensuration and analysis; and, finally, algorithms have begun to emerge which are designed to change the appearance of a picture by modifying the file which represents it. A brief list of references to this work is given at the end of the paper, including an excellent review of the field by Lipkin and Rosenfeld.

It is the third type of processing which has motivated the work described here, with the belief that picture modification in an on-line, interactive environment will soon be an important tool to experimental psychology and a useful training device for photo-interpreters, medical technicians, and others. In this context there are two types of manipulation: global manipulation of an entire file, for example, for the purpose of enhancing or degrading picture quality, and local manipulation in which one or more elements of a picture are modified by enlargement, rotation, translation, erasure, or the like.

REPRESENTATION OF IMAGE AND NOTATION

The image to be processed is represented in a file by a matrix, z, of positive integers. The matrix elements, z_{ij}, are the (digitized) average gray levels or transmittance values over a square area of the image, and each is associated with the midpoint of its respective square. In the discussion we define the image as a function of three variables, y_i, x_j, z_{ij}. The y_i, x_j are coordinates of midpoints of grid squares of the ith row and jth column of the M by N matrix; they normally take on the values $y_i=.5, 1.5, 2.5, \ldots; i=1, \ldots, M$ and $x_j=.5, 1.5, 2.5, \ldots, j=1, \ldots, N$; i.e., $y_i=i-.5$, $x_j=j-.5$. The upper left hand corner square of the image is represented by y_1, x_1, z_{11}.

An object or area within the image is defined by one or more connected boundaries. A list of y_i, x_j pairs outlining the object represents a boundary. The convention is followed of recording the coordinate pairs sequentially as though one were "walking around" the object, keeping it to the left. A restriction is that each boundary is continuous and closed in the sense that the absolute difference between successive y (and x) values in the list, including the first and the last, never exceeds 1; that is, no "jumping over elements" in either the horizontal or vertical direction is implied by the list. If, in scanning the boundary, it is found that the restriction does not hold, the "missing" coordinate pairs are linearly interpolated and inserted in the list.

Boundary points are considered to be inside, or a part of, the represented object. Where the area to be manipulated lies on or touches the frame of the image, the coordinate pairs of the frame itself are taken as the boundary.

DELINEATION OF OBJECTS

Manipulation of local areas or objects in a picture requires some technique of defining the domains of operation. In our programs, a local area was represented as a set of contiguous horizontal strips, the end points of which comprise its boundary. In the illustrations which follow, a boundary is denoted as a list of points $B_k=(y_{i_k}, x_{j_k})$, where $k=1, 2, \ldots, L$. This list is scanned and then sorted in ascending order of y. For each y, sets of left and right x-values are recorded in ascending order. The strips, not necessarily contiguous, which these sets define thus comprise the area, row by row.

To illustrate, assume a square area having an even number of elements, for example, 16 as in Figure 1a, with the following boundary:

k	y_{ik}	x_{jk}	k	y_{ik}	x_{jk}
1	.5	3.5	7	3.5	.5
2	.5	2.5	8	3.5	1.5
3	.5	1.5	9	3.5	2.5
4	.5	.5	10	3.5	3.5
5	1.5	.5	11	2.5	3.5
6	2.5	.5	12	1.5	3.5

This list, when sorted as described, yields:

y-values	Left and right x-values	
.5	.5, 1.5	2.5, 3.5
1.5	.5, 3.5	
2.5	.5, 3.5	
3.5	.5, 1.5	2.5, 3.5

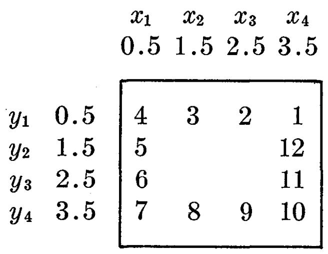


Figure 1a

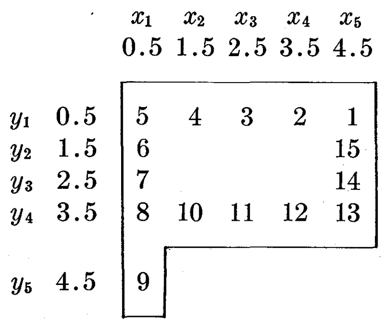


Figure 1b

Steps are taken in the initial scanning of the original boundary list to insure that each section of a horizontal strip contains an even number of x-values. When an odd number of x-values appear for a section of a given y, as, for example, when a horizontal boundary contains an odd number of elements or when a vertex is encountered, it is necessary to replicate one of the x-values. Suppose, for instance, that the object given in Figure 1b were to be represented. In the scanning process the pairs representing B_5, B_9, and B_{13} would be replicated and the strip representations after sorting would appear as:

y-values	left and right x-values		
.5	.5, .5	1.5, 2.5	3.5, 4.5
1.5	.5, 4.5		
2.5	.5, 4.5		
3.5	.5, 1.5	2.5, 3.5	4.5, 4.5
4.5	.5, .5		

Obviously, an object may have more than one boundary (e.g., an annulus); and a boundary can have singular points (e.g., a figure eight). When more than one boundary defines the area, the boundary lists are combined before sorting. In this manner, shapes other than simple closed curves, such as disjoint areas, can be manipulated in one operation. Once the areas are defined as strips, the elements contained therein can be translated with or without rotation, subjected to gray level transformations, replaced by other contents as in hole filling, and, by appropriate horizontal and vertical operations, contracted and expanded. This technique was used in the accompanying pictures to translate, rotate, and fill after erasure.

TRANSLATION AND ROTATION

Let i_r and j_r denote the indices of some pivot or anchor point (y_{i_r}, x_{j_r}), and let i_t and j_t represent the indices of the point into which the pivot point is to be mapped. Since the indices are here measures of distances, translation of an area of a picture into another position such that any (y_i, x_j) of the object maps into $(y_{i'}, x_{j'})$ is a trivial operation in that all the new indices for the z_{ij} values are given by

$$i'=i+(i_t-i_r), \quad j'=j+(j_t-j_r),$$

and the new coordinate pairs by $(y_{i'}, x_{j'})$. When translation is combined with rotation about the pivot through a counterclockwise angle, α, the new coordinates, which are given by

$$y_{i'} = y_i + (i - i_r) \cos\alpha - (j - j_r) \sin\alpha$$

$$x_{j'} = x_j + (j - j_r) \cos\alpha + (i - i_r) \sin\alpha$$

will not always be at the midpoints of the new cells (except in the case of angles which are multiples of 90 degrees). The fact that there is no longer a one to one correspondence between the original cells and the new cells requires a procedure for assigning z_{ij} values. Assignment of the gray level z_{ij} to the new cell into which the midpoint y_i, x_j falls would result in overwriting certain cells and leaving others empty. The "tilting" of each square element within the area rotated suggests a weighted assignment of the gray level on the basis of the area covered to the (possibly) six cells overlaid, as shown in Figure 2a. Such an allocation of gray levels might be called a "girdle" technique, since the orientation of the overlying square is constrained. In its place we implemented an approximation, the "chemise".

The chemise approximation allocates gray levels to underlying squares as if each rotated square had been allowed to "settle" about its center into the horizontal orientation shown in Figure 2b. Weights are assigned to the covered cells, never exceeding four in number, on the basis of the area covered. The indices and weights resulting are as follows:

$$i' = \text{integral part of } y' + .5$$

$$j' = \text{integral part of } x' + .5$$

$$p = \text{fractional part of } y' + .5$$

$$q = \text{fractional part of } x' + .5$$

$$w_{i'j'} = (1-p)(1-q)$$

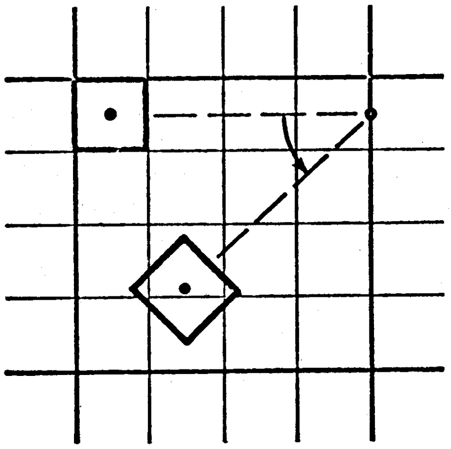


Figure 2a

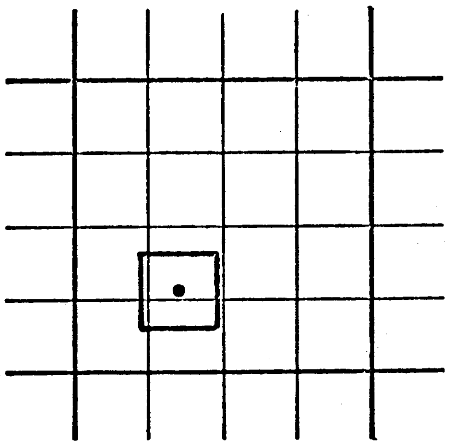


Figure 2b

$$w_{i',j'+1} = (1-p)q$$

$$w_{i'+1,j'} = p(1-q)$$

$$w_{i'+1,j'+1} = pq$$

These weights determine the proportionate parts of the z_{ij} value assigned to each of the four cells.

The illustrations which follow were obtained with the chemise approximation. It appears to be adequate for visual purposes.

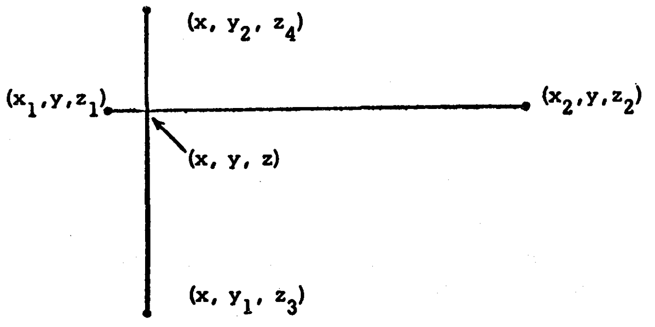
Picture 1, for example, is computer generated. The original, which is not reproduced, consisted of thirty horizontal bars of equal width, with gray levels ranging from 0 to 29. (The actual matrix is 400 by 400.) The portion of the original bars contained in a butterfly outline were rotated 20 degrees counterclockwise with the upper right wing tip as an anchor. Though the reproduction, which was obtained from a local vendor who scanned the computer file with off-the-shelf equipment, suffers from cyclical noise and inability to show all thirty gray levels, the appearance of straight lines has been retained and it is doubtful that quality could have been enhanced by increased precision in the algorithm.

HOLE FILLING AFTER ERASURE

Suppose that an object is to be erased, that is, that the gray levels within its boundary B_k, $k = 1, 2, \ldots, L$, are to be set to zero. Frequently the white area thus created will be visually distracting or unesthetic. The hole filling problem is that of associating a corresponding gray level z, which is in some sense consistent with the gray levels z_k of the B_k, with each point (x, y) of a given set of points interior to the boundary.

The boundary, B, may be taken to be a closed curve and the set of (x, y) to be the interior of B, though we will also provide for situations in which B "runs off the picture" by allowing for missing gray levels along the edge. The statement that the gray levels, z, in the interior are consistent with those of B is here interpreted to mean that a given implementation of a hole filling technique can yield a result which is subjectively acceptable in the sense that neither B nor the hole are visually apparent.

Consider the stencil consisting of a given point (x, y) interior to B and the corresponding boundary points (x_1, y), (x_2, y), (x, y_1) and (x, y_2), as shown on the diagram, with associated gray levels z, z_1, z_2, z_3 and z_4, respectively, where z is unknown (and, indeed, undefined).



Geometrically, the diagram may be interpreted either as the projection on the x–y plane of the five points whose coordinates are given, or as a two dimensional array having a z-value associated with each point. Algebraically, we are going to determine a number $z = z(x, y, x_1, x_2, y_1, y_2, z_1, z_2, z_3, z_4)$ such that

1. z is a linear combination of z_1, z_2, z_3, and z_4;
2. $\lim_{x \to x_i} z = z_i$ and $\lim_{y \to y_i} z = z_{i+2}$, $i = 1, 2$;
3. The horizontal weights (coefficients of z_1 and z_2) and vertical weights (coefficients of z_3 and z_4) will be directly proportional to two non-negative horizontal and vertical control parameters, L and M;
4. The horizontal and vertical weights will decrease monotonically with $|x - x_i|$ and $|y - y_i|$, where $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$.

In what follows we will first develop an interpolation formula for the stencil shown. Then we will develop a mixed interpolation—extrapolation formula for use when a point of B is missing, thus illustrating the handling of exceptional stencils.

To form the linear combination we first interpolate in the horizontal direction by forming

$$\frac{x - x_1}{x_2 - x_1} z_2 - \frac{x - x_2}{x_2 - x_1} z_1,$$

that is,

$$\frac{1}{x_2 - x_1}[(x - x_1)z_2 + (x_2 - x)z_1].$$

Similarly, the vertical interpolant is taken to be

$$\frac{1}{y_2 - y_1}[(y - y_1)z_4 + (y_2 - y)z_3].$$

The weights for these interpolants are defined as

$$\frac{L(y_2 - y_1)}{M(x_2 - x_1) + L(y_2 - y_1)} \quad \text{and} \quad \frac{M(x_2 - x_1)}{M(x_2 - x_1) + L(y_2 - y_1)},$$

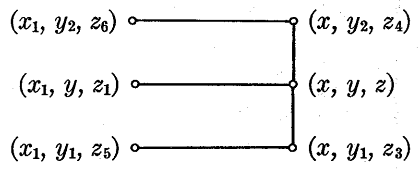
so that conditions (3) and (4) are met, and the linear combination is

$$z = \frac{1}{M(x_2 - x_1) + L(y_2 - y_1)} \left\{ L \frac{y_2 - y_1}{x_2 - x_1}[(x - x_1)z_2 + (x_2 - x)z_1] + M \frac{x_2 - x_1}{y_2 - y_1}[(y - y_1)z_4 + (y_2 - y)z_3] \right\}. \quad (1)$$

That condition (2) holds is a simple exercise.

Notice that changes in vertical or horizontal patterns can be achieved by adjusting L or M, and also there would be no loss of generality were we to set $M = 1 - L$. Furthermore, it is simple to introduce a "gain" parameter $K(x, y)$ as a multiplier of the right-hand side of the Equation (1). These parameters, L, M and K when used in conjunction with such unit operations as rotation, patching, superposition (or multiple-exposure), and smoothing would seem to be adequate for most practical purposes.

Suppose (x_2, y, z_2) is missing. Consider the stencil



where, z_5 and z_6 are assumed known. For the horizontal interpolation we wish to form

$$z_1 + m(x - x_1),$$

where m is an appropriately weighted average of the slopes

$$\frac{z_4-z_6}{x-x_1} \text{ and } \frac{z_3-z_5}{x-x_1}.$$

The weights are taken as

$$\frac{y-y_1}{y_2-y_1} \text{ and } \frac{y_2-y}{y_2-y_1},$$

respectively, yielding

$$z_1+m(x-x_1)=z_1+\left[\frac{(y-y_1)(z_4-z_6)}{(y_2-y_1)(x-x_1)}\right.$$

$$\left.+\frac{(y_2-y)(z_3-z_5)}{(y_2-y_1)(x-x_1)}\right](x-x_1)$$

$$=z_1+\frac{(y-y_1)(z_4-z_6)+(y_2-y)(z_3-z_5)}{y_2-y_1}$$

Once again we interpolate in the vertical direction by forming the linear combination

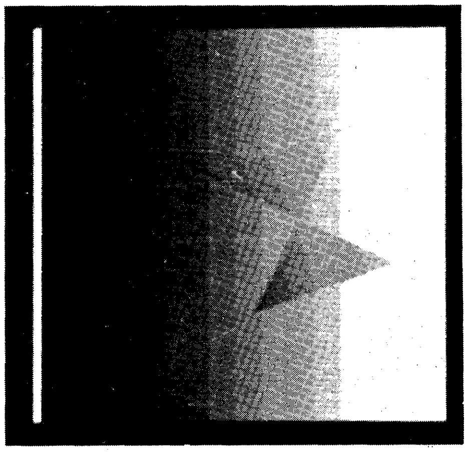
$$\frac{y-y_1}{y_2-y_1}z_4-\frac{y-y_2}{y_2-y_1}z_3$$

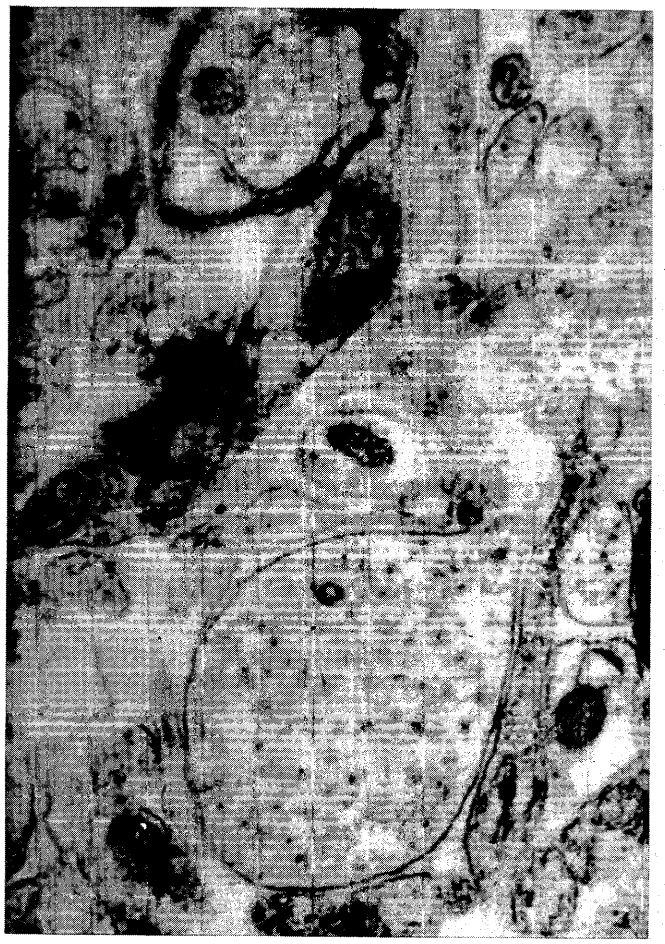
which reduces to

$$\frac{1}{y_2-y_1}[(y-y_1)z_4+(y_2-y)z_3].$$

The weights for the horizontal and vertical interpolants are taken to be

$$\frac{L(y_2-y_1)}{M(x-x_1)+L(y_2-y_1)} \text{ and } \frac{M(x-x_1)}{M(x-x_1)+L(y_2-y_1)}$$


Picture 1


Picture 2

respectively. Thus, we have obtained

$$z=\frac{1}{M(x-x_1)+L(y_2-y_1)}$$

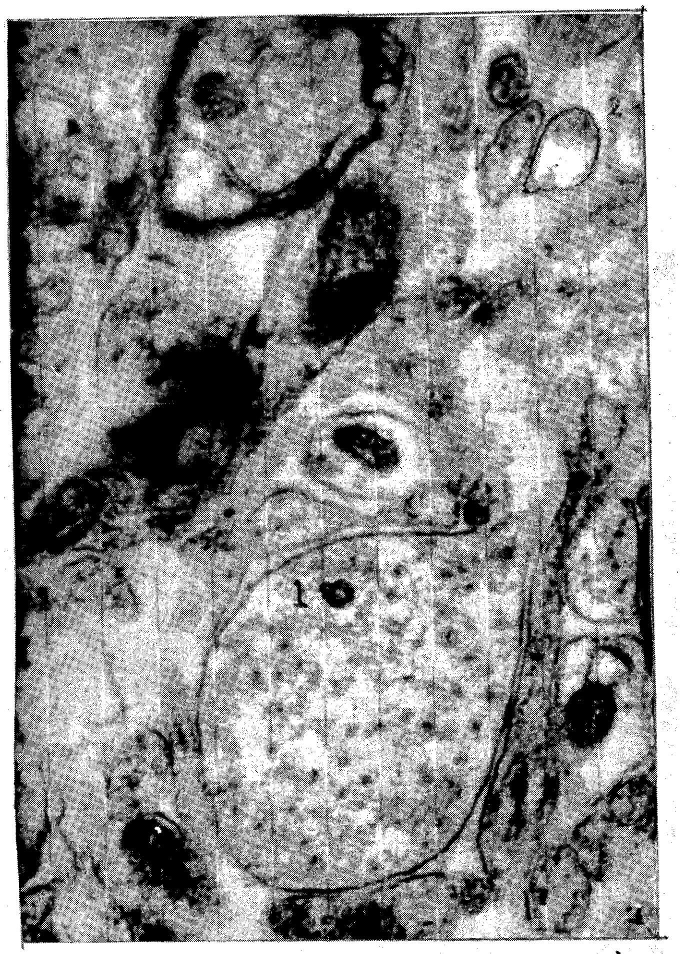
$$\left\{L[(y_2-y_1)z_1+(y-y_1)(z_4-z_6)+(y_2-y)(z_3-z_5)]\right.$$

$$\left.+\frac{M(x-x_1)}{y_2-y_1}[(y-y_1)z_4+(y_2-y)z_3]\right\} \quad (2)$$

as an interpolant analogous to Equation (1). The corresponding formulas for other stencils having a single missing boundary point can be written in a similar manner.

RESULTS

The pictures, except for computer generated Picture 1, are the results of processing a file on 7-channel magnetic tape. A microphotograph of brain tissue was



Picture 3

scanned and sixteen gray levels recorded on the tape. (A reproduction of the original photograph from which this tape was made can be found on page 8 of Reference 1.) The tape was processed on an IBM 360/40 (with FORTRAN under Version 13 of OS) and the resulting gray level values were output on 9-channel tape. The output tapes were read and recorded on film by a local vendor. Picture 2 is the recording of the original file. It consists of 1100 rows of 750 elements of the scanned photograph.

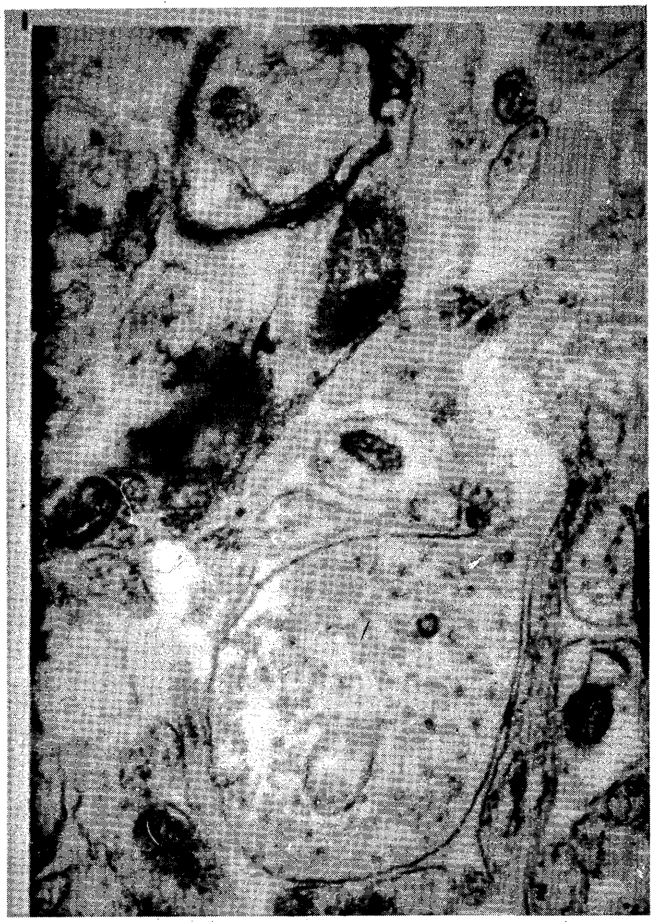
The local areas within the picture to be manipulated were outlined manually, and coordinates, B_k, of their boundary points were punched on cards. Picture 3 is a reproduction of Picture 2 with the two objects referred to in this discussion outlined.

Picture 4 is a reproduction after operations on a small section (300 rows) of the original image. Object 2 has been rotated counterclockwise 30 degrees about the indicated pivot point, P.

Picture 5 is a reproduction after several operations. Object 2 has been rotated 180° and translated to a position in the lower half of the picture. Object 1, has been rotated 60° and moved to the right. The areas formerly occupied by both objects have been filled in by the hole-filling technique described.

Owing to the relatively poor quality of reproduction of the pictures that have thus far been generated, it is not possible to speculate on the extent to which more precise algorithms would be required for any given implementation. In the geometric butterfly of Picture 1, the introduction of cyclical noise of both horizontal

Picture 4



Picture 5

and vertical nature makes it difficult to determine the extent to which staircasing was avoided by the weighting scheme. Yet in Picture 4, the detail in the original picture has been retained even to the fine, well-defined curved line on the right of the object. No staircasing, haloing, or other distortion in the surrounding area is observed.

CONCLUSION

Up to this point in time the literature demonstrating the effect of local manipulations of pictures has been limited, and comparative evaluation of our results is not possible. It has been demonstrated here that one can insert or delete items from a picture or change their appearance, position, and orientation using standard programming techniques and modest computing equipment, and that the modified pictures will be of adequate quality to support a variety of experiments. This paper treats the subject from a technique-oriented standpoint and establishes the validity of the techniques.

REFERENCES

1 B S LIPKIN A ROSENFELD Editors
Picture processing and psychopictorics
Academic Press New York 1970

2 A ROSENFELD
Picture processing by computer
Academic Press New York 1969

3 I H BARKDOLL B L McGLAMERY
An on-line image processing system
Proceedings of the ACM 23rd National Conference Washington DC August 1968 pp 705-716

4 R NATHAN
Digital video-data handling
NASA Tech Report No 32-877 Jet Propulsion Laboratory Pasadena California January 5 1966

5 R H SELZER
Digital computer processing of x-ray photographs
NASA Tech Report No 32-1028 Jet Propulsion Laboratory Pasadena California November 15 1966

The design of a meta-system*

by A. S. NOETZEL

The Moore School of Electrical Engineering of the University of Pennsylvania
Philadelphia, Pennsylvania

INTRODUCTION

The design and implementation of multiprogrammed, time-sharing computer systems continues long after the system is put to use. A tool is needed that will measure and evaluate the computer system while it is in operation, as an aid to further development or optimization for a particular usage. Research into the possibility of developing this tool was undertaken at the University of Pennsylvania's Moore School of Electrical Engineering. The research led to the design of the tool, which is presented in this report. It is called the *Meta-system.*

The uniqueness of the Meta-system is due to the coalescing of two widely used techniques—on-line measurement, and simulation—into one system. Measurement is performed by extracting raw representations of a computer system's operation (from that system) using software techniques only. Evaluation of the system is based on input of the measured performance characteristics to a simulation model that exercises modified hardware-software versions of the system. All the potential modifications to the system are evaluated in the context of the task load of the system, as extracted from the operational system.

PRELIMINARY DESIGN

Because of the novelty of the Meta-system, the description of the system will be preceded by a discussion of the design requirements of the system, and of the capabilities and limitations of various design alternatives. This should also make clear the area of applicability of the Meta-system.

The Meta-system was first conceived of as a feedback loop on an operating computer system, in which the functions of measurement, evaluation, and modification take place. These functions are discussed in the following paragraphs.

The measurement function of the Meta-system

The study of measurement techniques of operational systems resulted in the following set of requirements for the measurement function of the Meta-system:

1. It should be implemented by software techniques. The recognition and measurement of the logical or decision-making functions of the operating system will require decision-making capabilities in the measurement devices. Also, the measurement device must be capable of handling a variety of measurements and conditions of operation. A software device is therefore indicated. To avoid the expense of an additional processor, the measurement software will be multiprogrammed with the system being measured.
2. It should introduce little artifact.
3. It should record all information of interest. The complete specification of the information of interest will not be achieved until the entire Meta-system, including the system modifications to be evaluated, is specified.
4. It should be amenable to flexible off-line analysis (i.e., information must be detailed).
5. It should be flexible, so that the same general approach could accommodate new and more specific areas of investigation.

The choice of the measurement that meets all of these constraints is the *event trace.* This will be more precisely defined later. For the moment, the event trace is a record of the important occurrences in a computer systems operation. These occurrences are interrupts, activations of particular hardware devices within the system, or

* The work reported here was partially supported by the Information Systems Branch of the Office of Naval Research (Contract N0014-67-A-0216-0014).

calls on significant subroutines. The event trace is recorded at various points in the operating system by writing a small amount of data specifying the nature of the event, and the time at which the event occurs, into a buffer area in main memory. The buffer is written to external storage whenever it becomes full.

If the events constituting the event trace are properly defined, the event trace may be an extensively detailed record of the system's operation. It is, however, raw data, and will be processed by the evaluation function of the Meta-system.

The evaluation function of the Meta-system

Ultimately, the evaluation of the computer system will be provided by a system designer's response to the parameters of system performance obtained by measurement. And his response will depend upon the options available to him—which are determined by economic and political considerations. In order for the Meta-system to complete the evaluation of a system, it must include the system designer. Henceforth, 'evaluation function' will refer to the automatic part of evaluation—the processing of measurement data to be more helpful to the system designer.

The following possibilities for the evaluation function of the Meta-system are apparent:

1. If the event trace is a record of the time of the beginnings and terminations of each interaction of the user tasks, the evaluation function may condense this data to obtain the response time distributions for the user tasks.
2. If the event trace is a record of the activations and deactivations of the hardware devices, the evaluation function may condense this data to create a record of the utilization factor of each device.

In either of these cases, the actual evaluation will be obtained by comparing the condensed data with some standard. Since the purpose of the evaulation is to determine the modifications that will improve the system's performance, the standard for evaluation should be the same data taken from variant versions of the system, especially modified versions in which performance might be improved.

The evaluation function should satisfy these two requirements: (1) It should enable the system designers to identify potential performance improvements to the system, and (2) it should indicate the performance of the modified systems without the expense of performing the modifications.

The Meta-system described thus far has the form indicated in Figure 1. The evaluation function contains a 'trial modification' loop in which measurements of variant versions of the system are obtained. Another possibility for the evaluation function suggests itself:

3. The evaluation function may be a simulation model of the system, or modified versions of the system. Condensed data, such as (1) and (2) above, may be obtained from the model.

Selection of the simulation model as the evaluation function imposes special requirements on the measurement function. Consider the measurements of one system that are useful for simulating another system. The measurements must not be the final results, such as the utilization factors of various hardware components, since these will be obtained from the simulation model. Rather, they will be measurements that can be interpreted by the simulation model—*frequencies* of occurrence of the operations, for example. The simulation model may then allocate a different time interval for each operation, and different utilization factors will be obtained. New resource allocation techniques, as well as other system algorithms that influence the resource allocation, may be investigated in the simulation model.

The measurements taken from the operational system will therefore be measurements of the user task *demand* for various system resources, which is related to the *allocation* of the resources through the operation of the system.

Next, it must be noted that a task's demand for system hardware resources cannot be represented independently of the system on which the task is run, for two reasons. First, because the hardware resources (as well as the other resources—macros, algorithms,

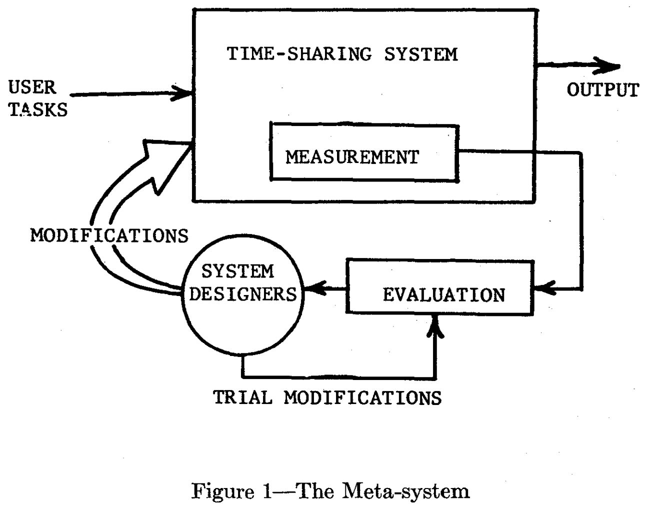


Figure 1—The Meta-system

tables, etc.) vary from system to system. Second, because it is the system as well as the user that generates the demand. Only at the highest level of demand specification—the level of machine-independent languages—is the demand purely due to the user. But even in this case, the demand for the execution of a program written in a high-level language cannot be correlated in a system-independent fashion, to the demand for hardware resources.

The result that is important to the theory of execution-simulation, is that it is possible to find representations of user task demand that are *relatively* independent of the system on which the task was run.

Relatively independent demand representations are representations which remain valid for a system that is within a specified class of modifications of the system from which the representations were taken. One concrete example may help make the concept of relative independence clearer. A task may be represented as a series of I/O operations. The number and frequency of the I/O operations are functions of the size of the data block that is involved in a single I/O operation. Then the sequence of demands for the I/O operations is a valid representation of the task's demand in every system which has the same block size in its external storage. It will remain valid even if the speed of the I/O device is changed, or if the configuration of the system or the scheduling of the device is altered, changing the wait time for the I/O operations.

It has been found possible to extract representations of user task demand for system resources from several different levels of operation. In each case, it is necessary to preprocess the event trace as taken from the system, before using the trace in the simulator. The preprocessor and the remainder of the Meta-system is described in the next section.

OVERVIEW OF THE META-SYSTEM

The Meta-system that was developed in detail was designed for operation on a hypothetized system which has the characteristics of three large time-sharing systems—TSS on IBM 360, TSOS on the RCA Spectra 70/46, and the Multics system on the GE 645. Specifically, it includes the features of recursive and reentrant operating system routines, demand paging, multiprogramming, multiple-I/O paths, multitasking and a Virtual Access Method. Reference to the details of the computer system in this overview of the Meta-system will be reference to the common, and commonly-known features of these systems.

An outline of the Meta-system is shown in Figure 2. The three parts of the Meta-system—the recording mechanisms, the preprocessor, and the simulation model—are described in the following paragraphs:

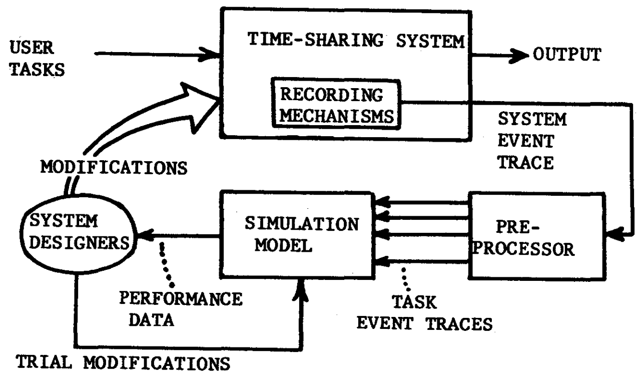


Figure 2—The developed Meta-system

The recording mechanisms

Measurement of the system's operation is performed by small open subroutines embedded within the operating system. The subroutines record the significant events in the system's operation. An event is composed of the following data items:

1. the *time*
2. an identification of the task (*task number*) for which the event occurred
3. the identification of the *type* of event
4. *data* associated with the event

A few examples of event types are the following: (In each case the *time* and *type* are recorded. *Task number* is recorded for each event type except 'idle.' The *data* field may or may not be recorded, depending upon the event type.)

The 'on' event, signifying a task gaining control of the processor. No data is associated with this event.

The 'idle' event indicating the beginning of a processor idle period. No task number or data need be recorded with this event.

The 'I/O-req' event, which is recorded when a task (or a system routine) requests a physical I/O operation. The data associated with this event is a unique representation of the physical address (e.g., device, cylinder, track) involved in the operation. An 'I/O-req' event is not synonymous with the initiation of an I/O device because the system may delay the actual operation.

The page fault of 'pf' event, indicating the necessity for a demand paging operation. The data, in this case, is the virtual address of the page required.

The event of a request for a Logical I/O operation, or 'LI/O.' This event is recorded when a task calls on a system routine to perform a Logical I/O operation. The data associated with this event is the logical specification, of the record required (e.g., file name, record name). The data is, essentially, the input parameters to the LI/O routine.

Other events complete the specification of the system's operation. The set of all events that occur during the operation of the system, recorded in the order they occur, is known as the system event trace.

The preprocessor

The system event trace is first preprocessed before becoming the input to the simulation model. Both the preprocessor and the model are run off-line.

The preprocessor accepts the system event trace in mass storage as input. The preprocessor has two functions:

1. To decompose the system event trace into event traces representing the resource demand of each task.
2. To 'purify' the task event trace. Since the task event traces will become input to a simulator of part of the system, the effects of that part of the system in those traces should be removed before the traces are used as simulator input. The preprocessor does this. Several examples of 'system' influence (i.e., the system to be simulated) in the representation of 'system' demand, will be shown later.

The output of the preprocessor is a set of task event traces—one for each task that was active during the period the event recording mechanism was operating. The events in the task event traces are much like those of the system event trace except that:

(a) The task numbers are not recorded in the events, since each event in a trace is of the same task.
(b) The time of each event is adjusted to be relative to the operation of that task only.

An example of the second function of the preprocessor —removing system influence in the event trace—is as follows: One event in a system event trace is a call on a Logical I/O (LI/O) routine. The LI/O routine calls on a Physical I/O (PI/O) routine, and the Physical I/O event is recorded. This call on the PI/O routine is not due to the task, because the task specified its I/O demand at the Logical level. The Physical I/O call must be considered due to the system, and is removed by the preprocessor of the system event trace.

The simulator

The simulator accepts the task event traces as input. The simulation model includes the operation of the system, from the level at which the events in the trace are recorded, to the hardware. The simulator consists of:

(a) A Clockworks, which selects the next event from the task traces and increments the simulation time.
(b) An Event Analyzer: the analog of the interrupt analyzer in the actual system.
(c) The Event Response Routines: models of the operating system routines.
(d) The Hardware Section: representations of the system hardware devices.

The output of the simulation model is the data that allows evaluation of the system and isolation of areas of possible improvement. This data consists of:

1. Utilization factors of the various devices.
2. Response time characteristics for the task interactions.

The utilization data is recorded in the simulation model by summing the simulated operating and idle times of each hardware device. Response times are calculated by the difference between the simulator time at which the first 'on' event of the task is accepted by the model, and the simulator time at which the 'terminate' event is accepted.

This data, obtained from the model, may be compared with the same data taken directly from the operating systems.

Levels of Meta-system awareness of system operation

It is obvious that the level of detail of the simulation will depend upon the class of modifications that is being contemplated.

Since the event trace, after some preprocessing becomes the input to the simulation model, the definition of the events in the trace will depend upon the extent of the simulation model. If the events in the trace are representations of some aspect of the original system's *operation* (such as the operation of a hardware device), and that aspect of the system is altered in the simulation model (i.e., the device characteristics are changed), then the event trace is irrelevant and useless to the simulation. To be useful, the events must rather be representations of the user tasks' requirements or *demand* for that aspect of the system's operation. The term 'system resource' which usually indicates the hardware devices of the system, may be extended to include any aspect of the system's operation that may be of interest—specifically, the system service macros, the scheduling routine, the loader, or a compiler. Therefore, the definition of the events in the trace are seen to depend upon the definition of system resource that is used for the specification of resource demand.

Lastly, the parts of the system that are of interest, and considered to be resources, will be included in the simulation model.

Thus, it can be seen that all of the following are inter-related:

- the definition of system resource used to specify resource demand.
- the class of trial modifications
- the extent of the simulation model
- the definition of the events in the trace

In the course of the design of the Meta-system it became apparent that these four entities could be specified at several different levels, which could best be differentiated by calling them different levels of *Meta-system awareness of the system operation.*

At the lowest level of awareness, only changes in speed or configurations of the hardware devices are potential modifications, and only the hardware and the scheduler of the hardware devices need be included in the simulation model. Any program calling for a hardware operation will be considered a user program, and the user programs' demand for system resources is the demand for hardware operations. The events in the trace, in this case, will be occurrences of the requests for hardware operations.

At the highest level of Meta-system awareness—total awareness of system and user programs—any modification to the real system may be made to the simulation model, since the simulation will be total—and the model as complex as the entire system. The events in the trace will be defined in terms of instructions or commands written at the terminals, and the system resource defined as all of the programs that respond to these commands.

Between these two levels, several more practical levels

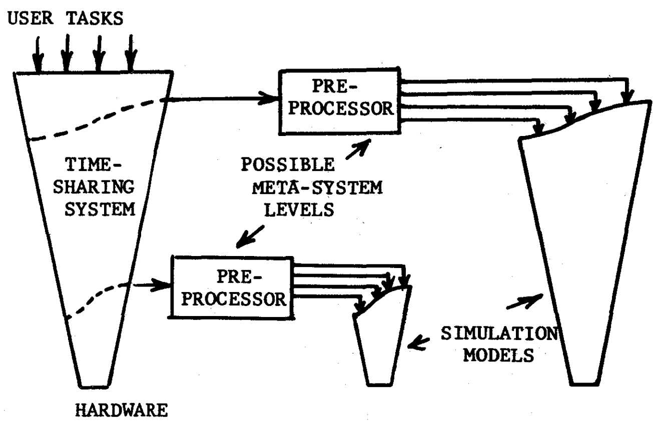


Figure 3—Conceptualization of Meta-system levels

of Meta-system awareness have been demonstrated in the detailed design of the Meta-system.

Meta-system levels of awareness are represented graphically in Figure 3. The representation of the time-sharing system in this figure is quite arbitrary. It roughly corresponds to the levels of logical complexity of the information-processing capabilities of the system, which are greatest for the parts of the system that directly communicate with the user, and least at the level of the hardware. Representations of system operation taken from one level are used as the input to a simulator of all parts of the system below that level, including the hardware. Several such levels of measuring and simulating the system are possible.

The design of any one particular Meta-system includes the determination of the Meta-system level. The factors determining the Meta-system level, listed above, must be selected to be mutually compatible.

DESIGN PROBLEMS OF THE META-SYSTEM

The analysis of the operation of time-sharing systems for purposes of implementing the Meta-system centered on isolating representations of user task demand for system resources that are independent of the allocation of the resources.

The representation of task demand taken from within the system are obtained by viewing the execution of any program above the Meta-system level to be due to the 'user,' even though the instructions being executed may have been coded by a system's programmer (as would occur during compilation of a user program) and the rest as the 'system.' The user task demand is given by the calls on the system functions.

These representations of task demand, however, are not easily separated from the operation of the system (below Meta-system level). There are feedbacks from system to task: allocation of system resources to the task modifies the task demand for system resources. Most of the problems encountered in the design of the Meta-system are due to the system influence in the 'pure' or system-independent representations of user task demand. The system influence is removed, in each case, by one or a combination of the following techniques:

(a) Construction and placement of the event recording mechanisms in the system to either exclude the system influence, or include supplementary information so that it may be removed later.
(b) Removing the system influence in the preprocessor of the event trace.
(c) Carrying the system influence into the simulator, but designing the simulator to neutralize its effect.

The following paragraphs outline some of the problems encountered in defining, extracting, and using representations of resource demand, and the solutions to them. Other problems, relating to the efficiency and practicality of the technique are also discussed.

Task identifications in the event trace

A basic function of a multiprogramming operating system is the scheduling of each task's use of system resources. Thus, the way in which the events of each task are intermingled in time is due solely to the influence of the system.

In another instance of system operation—specifically, the one provided by the simulator—one task may run faster or slower, relative to the performance of the others. The simulator must therefore view the representations of each task's demand separately.

The first way in which the representation of resource demand is purified, then, is to separate the resource demand of each task, into 'task event traces,' in a preprocessor of the simulator input. In order to do this, each event in the system event trace must be identified with a task.

The event recording mechanisms are therefore placed in positions within the operating system at which an identification of the task is known. This is no great obstacle to the implementation of the measurement portion of the Meta-system. In most cases, the Task Control Block for the task being operated on is immediately available to the operating system routine. When it is not, some unique representation of the task such as task number or TCB address, is always maintained by the system, and may be used as the task identification.

Some events are caused by the system only, and yet must be recorded in order to complete specification of task demand information. For example, the event of the processor beginning to idle need not be associated with any task, when it is recorded. Later, the 'idle' event will be used as a 'task relinquishes processor' event. The preprocessor of the event trace, having knowledge of which task is on the CP, will complete the specification of the event.

Representation of processor time requirements

A task's demand for the processor is given by the number and type of instruction executions it requires. Since it is neither practical nor necessary to count and simulate the execution of the individual instructions* a task's processor demand is taken to be the processor time required by the task.

The measurement of the time a task spends in the processing, however, is influenced by the amount of memory interference due to I/O operations into memory, taking place simultaneously with processing. Therefore, the task's processor demand will be defined to be the time the execution of the task would take if no memory interference were present. The system influence due to simultaneity is eliminated in the preprocessor of the event trace. The preprocessor calculates the 'pure' processing time, as follows: Let m be the memory speed (cycles per second) and c be the average fraction of memory cycles needed by the processor. Then with no memory interference, processing for a period of t seconds will spend ct seconds utilizing the memory.

Now suppose I/O operations taking k bytes per second are being performed in the background. The time for the processors use of memory will be expanded by a factor of $m/(m-k)$. The total time t' taken to perform the original t seconds of processing is:

$$t' = \left(1 - c + \frac{cm}{m-k}\right) t$$

The quantity $\{1-c+[cm/(m-k]\}$ will be called the memory interference factor f.

Each of the event traces taken from the system will contain the actual time taken on the processor, t'. But, in order to isolate the task requirement for processor time — t —, the trace must also contain an indication of the amount of I/O being performed simultaneously with processing, so that f may be known during each interval

* Modifications internal to the processing unit will not be considered here.

of processing time. Each event trace is constructed to contain some account of the I/O activity and the calculation of t from t' and f is performed in off-line processing of the event trace.

Specification of memory requirements

The specification of the hardware resource requirements made by the user programs—either directly or via a call on a system routine—are generally quite unambiguous. The specification of memory requirements is an exception.

A task's memory requirement is actually one word—instruction or data—at a time. For obvious reasons, memory must be allocated in larger units—in paging systems, one page or block at a time. The requirement for a page of memory—when the page is not allocated, will result in an unambiguous specification of demand: the page fault. But the system cannot know whether the demand still exists one memory cycle after the page has been allocated. Hence, the system itself specifies when pages should be de-allocated. It will generally do this by assigning a probabilistic value to the demand for the page and deallocating the block when either one of these conditions is met:

(a) When it becomes known that the page is no longer needed;
(b) When some other task has a demand for the memory block occupied by the page, which is greater than the probabilistic demand for the page;
(c) When it is known that the page will not be needed for a period, and it is likely that condition b) will be met before the end of the period.

If these deallocation judgments are made optimally, a page fault for a particular page will not occur soon after the deallocation of that page.

The record of page allocations and deallocations, then, is an inexact specification of the task's demand for memory: it shows a large degree of system influence. However, it is the only record of memory demand available without special hardware to monitor memory utilization. This example of system influence is not removed during preprocessing of the event trace, but is removed by the simulation model, and removed only when necessary.

The simulator will, in general, handle memory allocation differently from the allocation shown in the event trace. If, during the simulation, the task trace shows a page fault for a page that the simulator has already allocated, the page fault event is simply skipped. On the other hand, if the simulation model deallocates a page when it was not deallocated in the real system, the simulator must impose a potential page fault on itself. It replaces the page fault by evaluating the page re-use time as a random variable. The specification of the random variable is made from the average value of the page re-use times of the previous and next re-use times for the page that are available in the event trace.

Entrances and exits to system routine

Higher levels of specification of user demand for system resources are demands for system functions. The events indicating these functions are recorded at the entrances to the routines performing the functions. The operating system is written as a set of recursive subroutines so that a call on one system routine may result in calls on several others. If the original call on a system routine is taken to be an indication of user program demand, then these secondary calls, which are not made directly by the user program, may not be considered user demand. The events corresponding to these calls are system-contributed data, and must be eliminated from the event trace. A method of distinguishing user program calls from system program calls is required.

In order to distinguish system program calls from user program calls on the system functions, both the entrances to and exits from the system routines are recorded as events. Off-line processing of the event traces will remove the secondary calls that occur between the entrance event and exit event of a particular routine.

In order to place these event recording mechanisms in the system, the entry and exit points of the system routine of interest must be identified. The identification of entry points is straightforward. The identification of the exit points of system routines is, in general, a difficult problem. Each transfer of the following types must be analyzed to determine whether it should be considered an exit from the subroutine:

(a) Transfer to the return address provided by the standard subroutine call.
(b) Transfer to any address provided as a parameter to the subroutine.
(c) Transfer to an address taken from the pushdown stack of subroutine calls.
(d) A non-subroutine type transfer to another system function.

The methodology outlined in the design specifies at which of these transfers the event recording mechanism

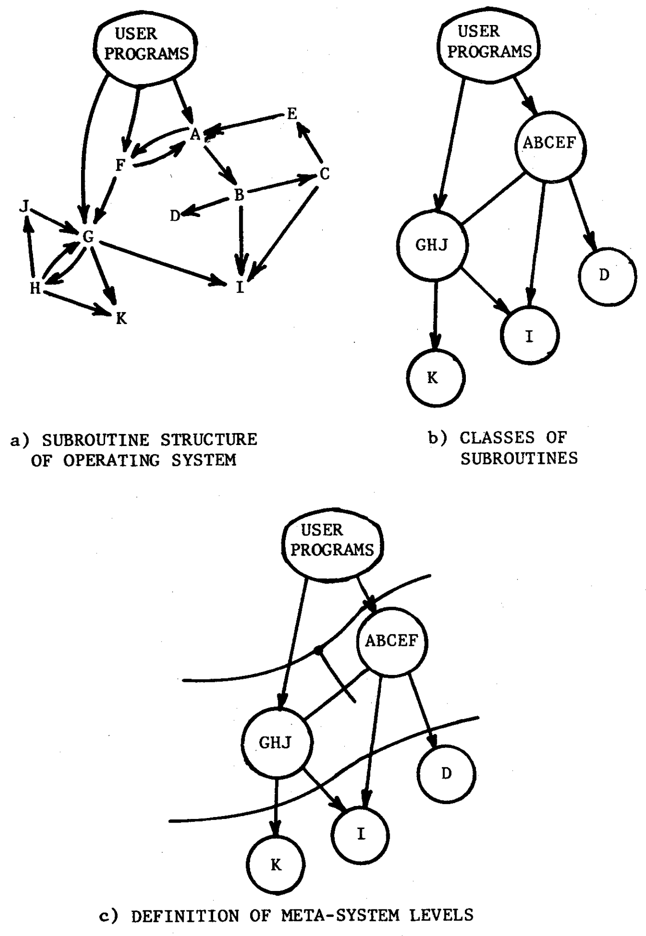

USER PROGRAMS
A
B
C
D
E
F
G
H
I
J
K
a) SUBROUTINE STRUCTURE OF OPERATING SYSTEM
USER PROGRAMS
ABCEF
GHJ
D
I
K
b) CLASSES OF SUBROUTINES
USER PROGRAMS
ABCEF
GHJ
D
I
K
c) DEFINITION OF META-SYSTEM LEVELS

Figure 4—Meta-system levels due to subroutine structure

(in some cases, a conditional recording mechanism) should be placed.

Volume of data recorded: the concept of class of subroutines

As the representation of user resource demand is refined, it becomes more a specification of logical functions to be performed than a specification of physical operations.

Because of the subroutine structure of the operating system, it may be necessary to record many subfunctions of one logical function. Also, it is necessary to carry some representation of the physical operations even at logical level of user demand specifications. Hence, high-level specifications of user demand require more events than lower-level specifications.

In order to generalize the technique of recording the event trace at higher levels of demand specification, the number of events must be somewhat independent of the level of specification—it cannot increase indefinitely as the specification level increases.

The generalization of the definition of higher level event traces must be made in such a way that the events indicating the operation of routines that are always called as the consequence of higher-level routines, are not included in the higher level traces. This requires analysis of the set of routines making up the operating system to identify classes of subroutines, that have a partial ordering imposed upon them by the nature of their calls.

The classes of subroutines are defined as follows: Routine A is in a class greater than or equal to Routine B if A calls on B, either directly or through another routine. If, in addition, routine B calls on routine A, then A and B are in the same class.

As an example, applying the definition of class to the subroutine structure of Figure 4a, in which the subroutine calls are indicated by arrows, yields 5 classes, whose partial ordering is shown in Figure 4b. The user programs are always a class by themselves, and always the highest class, since they are never called by the system as subroutines. From the fact that the user programs call on routines A, F and G, the class of user programs call on the classes ABCEF and GHJ.

Once the classes of the operating system routines are established, as in Figure 4b the set of levels at which the resource demand of the system may be measured (the level of Meta-system awareness of system operation), may be selected. The level is represented by the interface between the classes of routines that are considered user programs (higher in the ordering) by the Meta-system, and the classes of routines that are considered part of the system (the lower part of the ordering). A level is chosen by simply drawing a line across the arrows representing the calls on the subroutine classes.

Only the entrances and exits to the subroutines of the classes that are adjacent to the Meta-system level of awareness need be recorded as events.* The routines that are called only from higher-level routines within the Meta-system awareness need not be recorded, even though they may have been recorded in a previous, lower-level Meta-system.

The set of levels of Meta-system awareness that may be chosen for the subroutine structure of the example is

* This does not imply that the entrances and exits to *every* subroutine of such a class must be recorded, because a finer analysis (e.g., the operating system structure of Figure 4a) may show that only several of the routines of a class are called from above the Meta-system level. The analysis by class is a first approximation to specify a set of routines that need not be recorded. It is true, however, that if one routine of a class is included within the model, then each routine of the class, and all classes below it, must be included in the model.

the following:

(a) any of the 7 subsets of {K, I, D}, the lowest classes (excluding the empty set)

(b) {GHJ, I} or {GHJ, I, D}

If the class GHJ is chosen, then the lower class, K, need not be recorded since it occurs only as a consequence of GHJ. The other lower class, I, must be recorded, since it is called from above GHJ.

(c) {ABCEF, GHJ}

The user programs call on this set of system program classes. All other classes result from calls on this set; therefore, calls on these other classes need not be recorded.

It must be remembered, however, that the Figure 4b is a structural representation of Meta-system classes, and therefore provides only an estimate of the number of events which will actually be recorded. The volume of recorded data will depend upon the frequency with which control passes through the Meta-system level. Also, the calls on the subroutine classes are recorded at the entrance to the subroutines. Thus, if the Meta-system level crosses one arrow leading into a class, the level will in fact cross the other arrows into that class as well, whether or not this is intended in the definition of the level. For example, in Figure 4c, two levels are shown. One, the higher level, is inefficient, since some of the calls on the GHJ class result from the previously-recorded ABCEF class. In this case, a call sequence from the user program to F to G is recorded twice. The lower Meta-system is efficient.

Modeling system routines

The simulation model will include the models of some of the system routines. In order to preserve the economy inherent in simulation (as opposed to implementation and testing) the models of these routines are simplified. Yet the important aspects of their operation—in particular, the decisions that ultimately result in hardware resource allocation—must be duplicated within the model.

Simplified versions of system routines have been developed in the design of the Meta-system. It has been estimated that 60 percent of the code in the executive of a multi-programmed operating system exists for the purpose of error checking. It is assumed in the Meta-system design that the paths resulting from the error checks are taken rather infrequently, therefore, they are not a great influence on the resource allocation process. These error checking paths are omitted from the versions of the system routines in the model. Likewise, security considerations, in file operations, do not determine the location or identification of a particular data item. It may be assumed that the frequency of file operations being blocked for security reasons is small enough not to influence the utilization data obtained in the simulation. Security data has been omitted from the model.

SUMMARY

This report has been a summarization of the concept of the Meta-system, and a review of the problems that are encountered in the design of such execution-simulation systems, rather than simply a recounting of the details of the design.

The completion of the design of the Meta-system and successful trial runs of the system (insofar as they are possible on an unimplemented system) provide strong evidence that the Meta-system is technologically feasible, and will be an aid in the development of a time-sharing system. The question of its economic feasibility still remains since the implementation will be a considerable task. However, simulation models are employed during the development of most new computer systems. Development of the simulation model to operate in the Meta-system at the outset of the design is probably a feasible approach, since the full benefit of the Meta-system will be obtained in the later stage of design, and it may then be given to the user to optimize the system for his particular usage.

ACKNOWLEDGMENTS

The author is considerably indebted to his dissertation supervisor, Dr. Noah S. Prywes, for his aid, advice and encouragement during the period of the research. Also, many thanks are due to Dr. David Hsiao, who generously contributed his insight, experience and enthusiasm.

REFERENCES

1 G M AMDAHL B BLAAUW
Architecture of IBM S/360
IBM Journal of Research and Development Vol 8 No 2 April 1964

2 P CALINGAERT
System performance evaluation: Survey and appraisal
CACM Vol 10 No 1 pp 12–18 January 1967

3 D J CAMPBELL W J HEFFNER
Measurement and analysis of large operating systems during system development
AFIPS Proc FJCC Vol 33 pp 903–914 1968

4 C T GIBSON
Time-sharing in the IBM System/360: Model 67
AFIPS Proc SJCC Vol 28 p 61 1966

5 R C DALEY J B DENNIS
Virtual memory processes, and sharing in MULTICS
CACM Vol 11 No 5 p 306 May 1968

6 P J DENNING
Equipment configuration in balanced computer systems
IEEE Transactions on Computers Vol C-18 No 11 pp 1008–1012 November 1969

7 J R DENNIS
Segmentation and the design of multiprogrammed computer systems
J ACM Vol 12 No 4 pp 589–602 October 1965

8 E W DIJKSTRA
Structure of THE multiprogramming system
CACM Vol 11 No 5 May 1968

9 G H FINE P V McISAAC
Simulation of a time-sharing system
Management Science Vol 12 No 6 pp B180–B194 February 1966

10 D FOX J L KESSLER
Experiment in software modeling
Proc AFIPS FJCC 1967

11 *IBM System/360 time sharing system, concepts and facilities*
IBM Document C28-2003-3

12 *IBM System/360 time sharing system, dynamic loader—Program logic manual*
IBM Document Y28-2031-0

13 *IBM System/360 time sharing system, resident supervisor—Program logic manual*
IBM Document Y28-2012-3

14 A S LETT W L KONIGSFORD
TSS/360—A time-shared operating system
AFIPS FJCC 1968 pp 16–28

15 R A MERIKALLIO
Simulation design of a multiprocessing system
AFIPS Proc FJCC Vol 33 Pt 2 1968

16 N R NIELSON
The analysis of general purpose computer time-sharing systems
Document 40-10-1 Stanford University Computation Center December 1966

17 G OPPENHEIMER N WEIZER
Resource management for a medium scale time sharing operating system
CACM Vol 11 No 5 p 313 May 1968

18 E L ORGANICK
A guide to multics for subsystem writers
Project MAC Doc March 1967

19 *70/46 processor reference manual*
RCA Corp Document 70-46-62 March 1968

20 *TSOS executive macros and command language reference manual*
RCA Document 70-00-615 June 1969

21 J H SALTZER J W GINTELL
The instrumentation of multics
CACM Vol 13 No 8 August 1970

22 F D SCHULMAN
Hardware measurement device for IBM System/360 time sharing evaluation
Proc ACM National Conference pp 103–109 1967

23 V A VYSSOTSKY F J CORBATO R M GRAHAM
Structure of the multics supervisor
Proceedings FJCC pp 203–212 1965

An interactive simulator generating system for small computers*

by JOEL L. BRAME and C. V. RAMAMOORTHY

University of Texas
Austin, Texas

INTRODUCTION

This paper is concerned with a novel idea for the use of an interactive computer system. The system, called an Interactive Simulator Generator, is designed primarily for a student interested in learning about the structure of a computer. This is achieved through the use of design configuration in a simulation process. The Interactive Simulator Generator asks questions about the structure of the target machine the user wants and provides him finally with a simulator for his machine. It also provides a list of pseudo micro-operations that help the user to specify and develop assembly instructions and programs. Modest diagnostic and debugging aids are provided. Measurements such as frequency of use and average execution time, etc., are also available for comparing different computer configurations. The simulation of a subset of the PDP-8 Computer is illustrated. The system is programmed in System 360/APL.

This project consists of a computer system that acts as a tool to aid a user in simulating and optimizing a single purpose computer. The system could also be used as a teaching aid in computer design courses. The theme of the project is design initialization and reconfiguration in a simulation process.

The system is intended to be used primarily by students interested in learning the basic principles of computer design. The system allows the user to build the type machine that he has in mind, specify the instructions that he wants to use, and then execute a program written in the language he has specified.

The machine that the user builds can either be hypothetical, or a physical machine already defined.

The overall procedure that the user goes through consists of the following steps:

1. A user sits at an APL interactive computer terminal. The host computer will ask the user questions concerning his computer's operations. The idea behind the system is to break the simulated computer down into its basic operations, called pseudo-microinstructions.
2. As the user answers questions, these pseudo-microinstructions are generated to correspond with every operation of his computer. The user will also be asked to specify the time that he judges each operation will take. Only the knowledgeable user would be able to accurately estimate these values.
3. The user then uses each of these pseudo-microinstructions to define each of his machine's assembly language instructions.
4. The user will then be asked to enter a program in the assembly language that he has specified.
5. This program will be executed. A clock will be kept, and a report compiled, that will tell the user his execution time and where he spent his time.
6. The user will then be given the option of reconfiguring his computer. If he chooses to reconfigure, he must specify what changes (i.e., new pseudo-microinstructions) he wants, and which assembly language instruction macros that he wants to redefine. Execution will then revert back to step 5. If the user chooses not to reconfigure, then he can save his system, and sign off.

Thereby, the user can evaluate the hardware of his computer for a given need and financial restriction by looping through steps 5 and 6.

It should be noted here that the system does not attempt to exactly, step for step, simulate the opera-

* This work is supported by National Science Foundation Grant GJ-492.

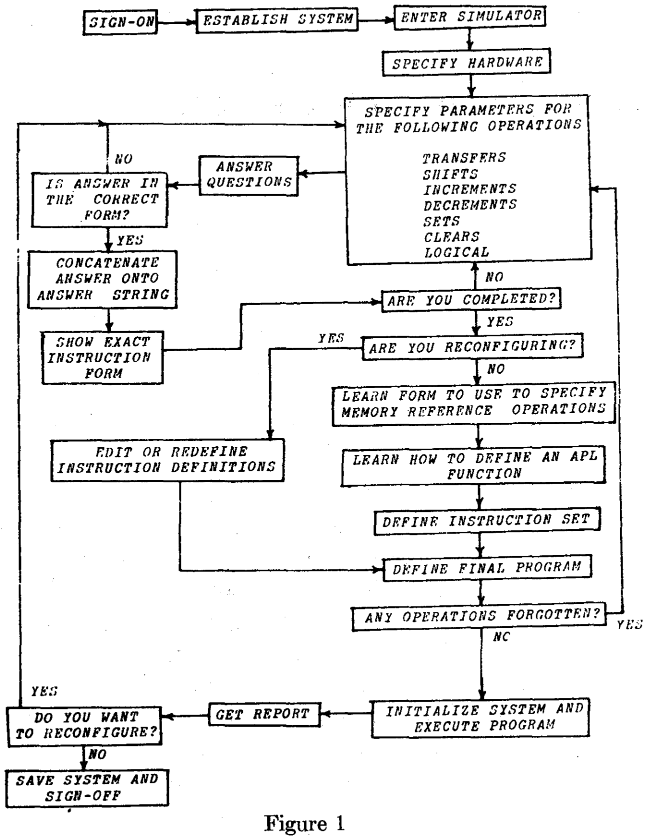

SIGN-ON
ESTABLISH SYSTEM
ENTER SIMULATOR
SPECIFY HARDWARE
SPECIFY PARAMETERS FOR THE FOLLOWING OPERATIONS
TRANSFERS
SHIFTS
INCREMENTS
DECREMENTS
SETS
CLEARS
LOGICAL
NO
ANSWER QUESTIONS
IS ANSWER IN THE CORRECT FORM?
YES
CONCATENATE ANSWER ONTO ANSWER STRING
NO
ARE YOU COMPLETED?
YES
SHOW EXACT INSTRUCTION FORM
YES
ARE YOU RECONFIGURING?
NO
LEARN FORM TO USE TO SPECIFY MEMORY REFERENCE OPERATIONS
EDIT OR REDEFINE INSTRUCTION DEFINITIONS
LEARN HOW TO DEFINE AN APL FUNCTION
DEFINE INSTRUCTION SET
DEFINE FINAL PROGRAM
ANY OPERATIONS FORGOTTEN?
YES
NO
YES
DO YOU WANT TO RECONFIGURE?
GET REPORT
INITIALIZE SYSTEM AND EXECUTE PROGRAM
NO
SAVE SYSTEM AND SIGN-OFF

Figure 1

tions of the target computer. Rather, it simulates only the results of the operations. The simulator cannot calculate the time that the operations take. It accepts the timing answer of the user to be true. If the user decides that he wants to spend more money for a particular piece of hardware, he can go back and specify a faster execution time. Then, he can determine how much this extra expense saves him in execution time. Furthermore, the system will obviously allow the user to develop the best programming method to achieve the result for which his single purpose computer was designed.

Most systems that the simulator is designed to handle could be developed by a knowledgeable person in half of one working day or less. The example, subset of PDP-8, took about one and a half hours to specify the machine, and one hour to define the instruction set.

Figure 1 presents this flow of action that the user goes through to define his computer.

The choice of APL

Iverson's *A Programming Language* is used to implement the Interactive Simulator Generator. The major reasons are: (a) It is easy to learn and get started; (b) Its statements are compact; (c) it is easy to debug programs since in system 360/APL the error messages are very meaningful; (d) it is easy to represent and manipulate structured operands; (e) its operators have remarkable power and flexibility. On the other hand, the ease of programming had to be compromised with long execution times. Since APL programs are hard to read, extensive comments and good program documentation had to be provided.

The simulator generator was programmed and debugged with the aid of an APL terminal in Austin using a IBM system 360/50 at Dallas. The current use of the generator is restricted mainly as a teaching aid to undergraduate students in electrical engineering and computer sciences learning computer organization, and in this the system has been very valuable.

OVERVIEW OF SYSTEM

Criteria for design

This system was theorized and implementation pursued with the following motivating principles in mind.

Primarily, the authors believe that a tool like this would be a great teaching aid. In a computer design course, an instructor could have the vehicle for converting his lectures into application demonstrations. This would be both stimulating and thought-provoking to the students as well as to the instructor. Furthermore, if the student could see the application of theory, he would certainly remember it longer and hopefully learn it faster. This kind of demonstration would be good not only for computer design courses, but also for any kind of computer science or engineering class where further knowledge of the computer and its operating procedures could make the students more aware of the problems and challenges that the computer presents.

Secondly, the authors believe that there exists a need for an "interactive" system to act as a tool to aid a computer designer in designing new or improving existing machines. There are existing computer-aided design systems, but few if any of them are in the interactive mode. The designer will get some of his best ideas sitting and working at the terminal. He can test out his ideas immediately instead of having to wait for a batch run job to be prepared and run. In this way when the designer has a brainstorm, and has the facts fresh in his mind, he can sit down at his terminal and explore his ideas. Often, if there is a time lag between thought and expression, a detail may be forgotten and this detail could be the difference between success and failure.

Structure of the overall system

We shall next outline the procedure that the user goes through to simulate his computer. There is an example in the appendix of the simulation of a small computer system that is a subset of PDP-8.

1. *Specification of Hardware*

The first thing that the Simulator must ascertain from the user is the physical description of the hardware of the computer that he wants to design. The topics that the user is asked about are the following:

(A) Memory Size
(B) Word Size
(C) Registers, Counters, and Special Memory Locations
(D) Method of Referencing Memory
(E) Appended Storage Devices

(A) Memory Size

The user is asked to state how many words of memory he will need to execute the assembly language program that he will enter at the end of the specification. If there is enough space in the Execute workspace, then the user is allowed to proceed. The amount of space he can use is approximately 12000 bytes. The simulator alone takes up about 32000 bytes of storage space.

(B) Word Size

The user is asked to indicate the bit length of the word size of his computer. If it is over 32 bits, then he is not allowed to proceed. This could be changed, but for space restrictions and because this system is primarily designed for a small single purpose computer, this is convenient and will allow most systems that it was designed to handle.

(C) Register, Counter, and Special Memory Location

The simulator must simulate the registers, counters, and special memory locations. (The system asks about special memory locations only if the user has a variable name or tag addressing scheme for his memory. If he simulates his memory, then the system treats his entire memory as special memory locations.) The Simulator asks the user to name all the registers, counters, and possibly special memory locations that his computer has. It gives the user a new name and number to refer to each device. The system is designed such that the user uses the numbers of the different types of storage devices (i.e., register, counter or special memory location) to define the exact operations that he wants. It also asks the user to give the bit length of each register and counter. The Simulator builds a word the exact length of each storage device. If the user wants, he can get a print out of the storage devices in the numerical order that they are identified with their respective bit length.

As an example, if the third register that the user enters is named ACC, the computer will tell him to refer to this device as REG3. Then when he is specifying an operation that involves ACC, he refers to it by the code that the Simulator tells him to use for the type of device that it is (REG), and its respective number which is 3.

(D) Method of Referencing Memory

The user has two ways that he can refer to memory locations in his final assembly language program.

1. He can refer to them by the register indexes which specify the location in memory that he has this value stored. If he does this, the array MEMLOC is used to simulate his memory.
2. He can refer to tags, i.e., names used to refer to specific memory locations. Tags are referred to as variables in the example at the end. The system keeps up with the location where these tags are stored by referring to the symbol table. He must specify the number of tags that he needs, and they are simulated in the array VARIABLE.

This is necessary information for the computer when it starts trying to execute his memory reference instructions, and it has to determine which array, i.e., VARIABLE or MEMLOC, that he is referring to.

(E) Appended Storage Devices

The Simulator has to find out if the user wants any storage devices appended together. This is needed for any shift operation that the simulated computer might have where a multiple number of devices shift as a whole. Also, the simulated computer might have two devices appended together to handle double precision arithmetic operations.

The Simulator does not record the devices that are appended together as such. It only needs to know if the specialist requests such a need, and tell him the limit of appended devices that the system will accommodate. This limit is set at three, but could easily be changed if the need existed. The procedure for shifts that involve more than one device is all taken care of in the way that the user specifies the parameters.

2. *Specification of Operations*

The user will be asked questions concerning his operations. The questions are broken down into two parts, that is, operate and memory reference operations.

(A) Operate Operations

The user is asked detailed information about the operations of his computer that are strictly internal. That is, the operations that work on data after it has been referenced from memory. These operations include:

1. Sequential transfers
2. Simultaneous transfers
3. Shifts
4. Logical operations
5. Increments and decrements
6. Sets and clears

There is a separate APL function written that will duplicate most any operation of the above types that the user could ask for. If the user is unable to specify an operation that he wants on his computer, then he could insert an APL subroutine for this purpose.

The user is asked to specify every detail of the type operation that he wants. As he answers the questions, he is building a list that is concatenated onto the main answer string, a vector named STRING. His answers acts as parameters to each separate function when he calls that function name with the correct indexes of STRING. As the user specifies the exact operations that he wants, the Simulator first checks to make sure that his answer is in the correct format. If it is not, then the answer is not accepted, and he is asked to reenter his answers for that operation. If his answer is in the correct format, then the Simulator gives the user the exact instruction that will perform the operation that he has just specified.

The following is a section taken out of the specification procedure. There is an instruction booklet that goes with the Simulator Generator. The user is told where in the booklet to refer for each section of operations. The questions and instructions were all presented to the user by the Simulator at first, but because of a space restriction, they had to be transferred into a user's booklet.

The following is an example on attaining information about the user's sequential transfers.

I NOW WANT YOU TO ENTER ALL THE SEQUENTIAL TRANSFERS THAT YOUR COMPUTER HAS.

THIS FIRST SET OF QUESTIONS IS CONCERNED ONLY WITH THE TRANSFERS THAT CAN BE DONE SEQUENTIALLY. IF YOU HAVE TRANSFERS THAT ARE DONE SIMULTANEOUSLY, BUT COULD BE DONE SEQUENTIALLY, PLEASE INCLUDE THEM IN THIS SECTION. IN THE FOLLOWING QUESTIONS WHEN IT ASKS FOR CODE NUMBERS OF DEVICES, USE THE FOLLOWING CODE NUMBERS:

1—REGISTER
2—COUNTER
3—SPECIAL MEMORY LOCATION

PLEASE ENTER ANSWERS TO THE FOLLOWING 7 QUESTIONS FOR AS MANY TIMES AS THERE ARE SINGLE BITS, OR STRINGS OF BITS INVOLVED IN EACH PARTICULAR TRANSFER. BE SURE TO SKIP A SPACE BETWEEN EACH NUMBER. IF ALL YOUR TRANSFERS ARE SIMULTANEOUS, AND MUST BE DONE THAT WAY, THEN ENTER THE NUMBER 9999.

1. CODE NUMBER OF DEVICE THAT BITS ARE TRANSFERRED FROM.
2. NUMBER OF DEVICE THAT BITS ARE TRANSFERRED FROM.
3. NUMBER OF LEFTMOST, COUNTING FROM THE LEFT, BIT OF THE STRING OF BITS BEING TRANSFERRED.
4. TOTAL NUMBER OF BITS IN THIS PARTICULAR STRING.
5. CODE NUMBER OF DEVICE THAT BITS ARE TRANSFERRED INTO.
6. NUMBER OF DEVICE THAT BITS ARE TRANSFERRED INTO.
7. NUMBER OF BIT, COUNTING FROM THE LEFT, IN THE RECEIVING DEVICE THAT LEFTMOST BIT OF THAT PARTICULAR STRING IS STORED INTO.

AFTER ANSWERING QUESTIONS 1-7 FOR AS MANY TIMES AS NECESSARY, PLEASE MAKE YOUR LAST ENTRY THE AMOUNT OF TIME, IN INTEGER NANOSECONDS, THAT YOU THINK IT WILL TAKE YOUR COMPUTER TO EXECUTE THIS OPERATION.

The Simulator will wait for the user to respond before execution continues.

Suppose the user has a single transfer operation, where he transfers bits 1-3 of REG1 to bits 5-7 of COUNT2 and bits 4-12 of REG1 to bits 1-9 of REG2. Suppose that the user enters the string:

1 1 1 3 2 2 5 1 1 4 9 1 2 1

The Simulator will tell him that his answer is in the wrong form. (He has forgotten to specify the TIME.) He will be asked to reenter his answers. Suppose this time he enters the string:

1 1 1 3 2 2 5 1 1 4 9 1 2 1 2000

The Simulator will respond with the following:

THIS OPERATION WILL BE REFERRED TO AS
TRANSFER STRING [8+ι15]
PLEASE MAKE NOTE OF THIS FACT.

Then when he wants to program an instruction that does this one operation, he defines an APL function with his instruction's name, and uses this instruction form as the first, and only in this case, function definition command. The string that the user entered is stored in locations 9-23 of the array STRING. The instruction he was told to use will issue the contents of these locations of STRING to the function TRANSFER, where they will be used as parameters to execute the operation he defined.

Suppose the user has an instruction that combines several of the elementary operations in one execution. He merely has to define a function with the name of his instruction and include each pseudo-microinstruction, in the form that the simulator asked him to use, in that one function definition.

(B) Memory Reference Operations

The user was given two choices for the way that he wanted to reference memory in his instructions that involved storing into or accessing from memory. The operate functions allow only registers, counters, and special memory locations to have operations performed together. It is easier for the user to program using tags and let the system keep up with where it is stored, instead of having to remember the memory locations that he stored a value into. However, if he simulates his memory, then he can perform operations on contents of memory instead of having to first load the contents of a tag into a storage device. This is because his memory is simulated by the array MEMLOC, which also simulates special memory locations (which can perform operations with registers or counters). The operate functions allow the user to use memory locations as if addressing is by absolute location of memory.

In other words, if the user has operations that take the contents of memory locations directly and perform operations between it and a storage device, then he wants to simulate his memory. His alternative, if he uses tag, is to add a register that he really does not have, and first load the contents of a tag into this dummy register before performing the operation. This alternative, though very clumsy, will give the user the same result.

He must use the LOAD and STORE commands to transfer the contents of tags or memory locations into and out of the machine, where necessary.

The commands that the system provides for memory reference instructions are the following:

READ
PRINT
LOAD
STORE
ADD
SUBTRACT
MULTIPLY
DIVIDE
DECLARE

The instructions, except for DECLARE, will handle values contained in either VARIABLE or MEMLOC. DECLARE is for only variable assignments.

The forms that the user must use are explicitly defined by the Simulator. The user is given a function name, and he defines the parameters, which can be either set numbers, or tag names. If he uses tags, he is responsible for assigning them the correct values in the course of his final program. He is told how to do all of this.

An example of the form provided is the READ command, which is defined as follows:

FOR THE INPUT OPERATION THAT YOU HAVE, USE THE FOLLOWING FORM:

```
READ        VARIABLE1        TIME
```

THE FIRST VARIABLE (VARIABLE1) REFERS TO THE NUMBER OF THE TAGS OR MEMORY LOCATION THAT YOU WANT TO INPUT A VALUE FOR.

THE PROGRAM WILL STOP AND ASK YOU TO ENTER THE VALUE OF THAT TAG OR MEMORY LOCATION EACH TIME THAT IT IS ENCOUNTERED.

TIME IS THE INTEGER AMOUNT, IN NANOSECONDS, THAT YOU THINK IT WILL TAKE YOUR COMPUTER TO EXECUTE THIS OPERATION.

3. *Definition of Instructions*

The user is now asked to take the instruction forms that were given him by the system, and use these forms to define the instruction set that he wants to use. To do this, he must first be instructed how to define an APL function. This is done by having him refer to the instruction booklet. Once he has learned this procedure, he can use the operate instructions that the system has given him and the memory reference instruction forms that he was instructed to use to define his instruction set in any manner he chooses.

The only exception to this is any jump instruction. Because of the absence of a needed APL operator, the jumps that the user requires must be written in APL code. There are several simple forms for him to use to achieve the type operation he wants.

4. *Entrance of Target Machine's Algorithm*

Now the user takes the instructions that he has defined, and the APL instruction forms that he has learned to use, and writes an APL function in about the same format as he would write a program for his machine.

5. *Execution and Reconfiguration*

(A) Execution

To execute his program, the specialist simply types its name. After execution, he can get two reports from the Simulator.

1. The times that this execution took.
   - (a) The accumulated time for each function that the specialist defined. This is the time that he predicted. If these times are accurate, this is the time that it would take for that program to run on his computer.
   - (b) The time that it took APL to execute this program.
2. A report on the number of times that each basic function was executed. From these two reports, he can determine not only how much time he spent, but also where he spent it.

(B) Reconfiguration

Now, if he wishes he can go back and specify more and/or different operations. He can then take the extra operations that he has defined, and either redefine the instructions that he has already specified, or he can define more instructions that he wants to use in his assembly language program.

Once he has done this, he can execute his same program, with the new specifications, and see if he has improved his execution time. Hopefully, with the loop that exists between these steps, he can build the best system that for a given investment will perform this one operation.

Once he is happy with the configuration of his machine, he can experiment to see if he can come up with a programming technique that will improve his execution time even more. In this way the user has two methods that he can work with to improve his machine for its application.

The user also has the option of simulating several machines for the same application and deciding which machine would be the best for his particular need.

6. *Programming Aids*

If the user has difficulty getting a program to work correctly on his simulated computer, he has the following options of programming aids.

(A) He can put a TRACE on his program that will allow him to specify the lines of the different functions that he wants to trace. This facility will print the name of the function being executed, and the line number, as well as the value assigned on that line. This can be done for each function, for as many lines of that function as needed with only one short, simple command. To remove the TRACE also takes only one command.

(B) He can stop his program at any point that he wishes, by issuing one command. When his program stops, he can ask for the contents of any storage device or memory location at that instant. Then with one simple command, he can start the program up right where it left off. The program will stop each time it hits this line number, and he can specify as many line numbers as he wishes. To remove this facility, takes one simple command.

(C) The user can get a memory dump by issuing one simple command. He can ask for all of memory, or only a particular part.

(D) The system has modest diagnostics to help the user get through the simulation moves. Between the booklet he has to refer to and the diagnostics offered by the system, it is not difficult to become accustomed to the operating procedures of the simulator.

SPECIFICS OF SYSTEM

The following divisions of this paper give the finer details of this system.

Operate operations

This is the heart of the Simulator. The Simulator has to be able to duplicate any kind of transfer, shift, logical operation, increment, decrement, set or clear operation that the simulated computer might have.

The original idea was to have a separate function for each micro-operation. However, this would have involved a great deal of redundancy to anticipate every operation that the user might want. In APL there are two modes of operation—execution and definition. One is unable to do the opposite operation in either mode. Also, one cannot copy literal characters as themselves when one transfers strings for one function to another. Therefore, the method adopted to handle names and variables, as discussed above, is to specify all the tags and let the system reidentify them and tell the user the new identification as a name and a number.

Then, the functions are programmed so that the user, in his parameters, specifies the code for the type storage device he is referring to and gives the number of that particular device. In this way, the user can define exactly the operation that he wants, and one function can be designed to handle all cases.

The main functions are defined very generally, and the user can specify exactly the operation he wants for each type operation definition. Also, in the definition of the parameters, the user enters the integer amount of time, in nanoseconds, that he thinks it would take that operation to execute on his computer.

All operate functions, except INCREM, convert the contents of the word used to simulate the storage device into binary before performing the appropriate operation, as defined by the user. Once the operation has been completed, the Simulator converts the contents of the location back into its decimal equivalence.

Before the system will accept the parameter string that defines an operation, it checks to make sure that it is in the correct form, that is, it has the correct number of elements. If it is not, then it asks the user to redefine the parameters for that operation. If the parameter string is accepted, then it is stored in an array, STRING. Then the user is told the exact form of the function call that will duplicate the operation he has just defined. It is his responsibility to record this instruction form and keep track of which instructions duplicate their respective operations.

Because of a space restriction, all parts of the simulation system that are strictly instructional in nature were removed from the system and put in an instructional pamphlet. The Simulator advises the user on the proper section of the pamphlet to refer to for each group of questions that must be answered.

(A) Transfers

The Simulator has to be able to transfer any bit or string of bits from any storage device to any other storage device. Not only does the Simulator have to take care of sequential transfers, but it also has to duplicate simultaneous transfers. That is where certain bits of a storage device are sending and receiving data at the same instant.

The Simulator accomplishes this need with the functions TRANSFER, for sequential transfers, and SIMULFER, for simultaneous transfers.

(B) Shifts

The Simulator must be able to make any kind of shift operation that the user might want. This includes any appended storage devices that might shift as a single unit.

This is implemented in the system by the function SHIFT. The user defines the exact shift operation he wants by specifying parameters. This parameter string is stored in the array STRING, and the user is told the exact form of the instruction that will duplicate this operation. The system as implemented in the prototype, will allow no more than three appended storage devices to be shifted as a whole unit. This limitation, however, is only a matter of convenience. If more is needed, it can easily be changed to handle the added facility.

(C) Logical Operations

The Simulator must be able to perform any kind of logical product, sum or complementation between storage devices. This capability is handled in the system by the function LOGOP. This function allows any logical operation between any two storage devices of the same length, or between any storage device and integer number. The operations that are available are: AND, inclusive OR, exclusive OR, NOT, NAND, inclusive NOR, and exclusive NOR.

(D) Increments and Decrements

The Simulator must be able to perform any kind of increment or decrement operation on any storage device.

This capability is handled in the system by the function INCREM. This function allows any storage device to be incremented or decremented by any amount.

(E) Sets and Clears

The Simulator must be able to set or clear any bit or string of bits in any storage device. This facility is handled in the system by the function SET. This function allows any bits in any storage device to be set with ones or cleared to zeroes.

Memory reference operations

The instructions, i.e., functions, defined up to now were those that would have no argument. Or in other words, those functions that operated on data that was already loaded into particular locations in the computer. The user will also need to define those functions that necessitate reference to memory, that is those instructions that require data to be stored into or accessed from memory.

The Simulator does not attempt to simulate the memory, as such, of the guest computer unless the user asks for it.

The memory reference instructions are defined so that the user uses set forms and specifies the parameters that these operations use. The parameters specify the particular register, variable, or memory location that he is referring to for that operation. These parameters, depending on the user's programming, can either be set numbers or tag names. If they are integer numbers then obviously, they stay the same throughout the execution of the program. If they are tags then the user is responsible for initializing and reassigning the values throughout the course of his program. He is instructed on how to do this in APL in his instructional pamphlet. In this way, he can work with arrays as easily as single tags. He must be careful to keep his index numbers straight for each respective tag name or array that he works with. The only restriction that the user has is his choice of tag names. He cannot use any tag name that the system uses, and all of his own must be unique. If he simulates memory then the locations are defined as index numbers from 1 to the total number he asked for.

The only exception to the rule of defining memory reference instructions is the JUMP instruction. This is fully explained in the section on jumps.

Because of a space restriction, the phases of the system that are strictly instructional on memory reference operations were also removed from the system and placed into the instructional pamphlet. The Simulator advises the user on the section of the pamphlet to refer to for memory reference instructions.

(A) Load

Any operation that the user's computer has that takes data out of a tag or memory location and places it into a register will be of the following form:

LOAD VARIABLE1 VARIABLE2 TIME

The first variable (VARIABLE1) refers to the number of the tag or memory location he wants loaded. The second variable (VARIABLE2) refers to the number of the register that he wants the value loaded into. TIME is the integer amount, in nanoseconds, that he thinks it will take his computer to execute this operation.

(B) Store

Any operation that he has that takes data out of a register and stores it into a tag or memory location will be of the following form:

STORE VARIABLE1 VARIABLE2 TIME

The first variable (VARIABLE1) refers to the number of the register whose contents he wants stored. The second variable (VARIABLE2) refers to the number of the tag or memory location that he wants the value stored into. TIME is the integer amount, in nanoseconds, that he thinks it will take his computer to execute this operation.

(C) Arithmetic Operations

Next are the arithmetic operations. They will be of the following form:

```
ADD
SUBTRACT      VARIABLE1     VARIABLE2     VARIABLE3     VARIABLE4     TIME
MULTIPLY
DIVIDE
```

The first variable (VARIABLE1) refers to the code as to whether this operation is between two registers, a tag and a register, or a memory location and a register. The code is the following:

0—REGISTER AND REGISTER
1—REGISTER AND TAG
2—REGISTER AND MEMORY LOCATION

The second variable (VARIABLE2) refers to the number of the tag, memory location, or register referred to in VARIABLE1. The third variable (VARIABLE3) refers to the number of the other device, a register, involved in the operation. The fourth variable (VARIABLE4) refers to the number of the register that the result of the operation is to be stored into. TIME is the integer amount, in nanoseconds, that he thinks it will take his computer to execute this operation.

(C) DECLARE

If the user has a tag to which he wants to assign a value without having to do a read statement, then he uses the following form:

VARIABLE1 DECLARE NUMBER TIME

The first variable (VARIABLE1) refers to the number of the tag that he wants to assign a value to. NUMBER refers to the value that he wants the tag assigned. TIME is the integer amount, in nanoseconds, that he thinks it will take his computer to execute that operation.

(D) READ

For the input operation that the user has, he will use the following form:

READ VARIABLE1 TIME

The first variable (VARIABLE1) refers to the number of the tag or memory location that he wants to read in a value for. The program will stop and ask him to input the value for each tag or memory location read, as encountered. All input must come from the user through the terminal. The system will tell the user the tag number it needs a value input for, for each read instruction encountered. TIME is the integer amount, in nanoseconds, that he thinks it will take his computer to execute that operation.

(E) PRINT

For the output operation that the user has he will use the following form:

PRINT VARIABLE1 VARIABLE2 TIME

The first variable (VARIABLE1) refers to the number of the tag or memory location that he wants printed. The second variable (VARIABLE2) refers to the number base that he wants that tag or memory location printed in. The system will print a header with each print instruction encountered, telling the number of the tag or memory location printed on that line. TIME is the integer amount, in nanoseconds, that the specialist thinks it will take his computer to execute that operation.

Jumps and labels

(A) Jumps

According to APL manuals, labels are global, and jumps can be made from any location of one function to any other location of any other function in that same active workspace. Unfortunately, in the APL version that the prototype is implemented in, labels are local and therefore jumps can only be made to labels located within the same function definition as the jump instruction. Another restriction with respect to jumps is that APL will not allow you to transfer a literal string in a string of literals like a tag name, from one function to another and then jump to that tag. APL recognizes only the value of a tag and not its literal self.

Therefore, the user will have to program his own jumps in APL, and program them in his assembly language type algorithm.

The system was originally theorized to be such that all instructions, both operate and memory reference, could be defined by micro-operations.

In implementation, the idea to define jumps like the other memory reference instructions, had to be abandoned. When, and if, the APL system personnel complete their work on the 'unquote operator,' i.e., allows tag to be transferred from one function to another as itself instead of its value, and if the system has global labels, then both of these original ideas could be easily implemented as originally designed.

The Simulator instructional pamphlet tells the user the two major forms, i.e., conditional and unconditional of APL jumps that he can easily learn and use. In both forms, he is given the option of branching to a label, or skipping over lines of instruction. The user has the prerogative of programming his jumps as he wishes. Any conditional jump instruction must be conditional on either the contents of a register, counter, or special memory location. This is mandatory.

(B) Labels

Label names have the same restrictions as tags, only that they must start with a letter and not have any operators in them. The only time that the specialist has a tag name on the left of the function name is when he plans to use it as a label. Lables can be on any line, but they must be followed immediately by a colon. Any label that is branched to must be defined as such.

The only other time that there is any other character at all on the left of a function name is in the DECLARE operation, and it is a number. Therefore, there should not be any confusion or difficulty with labels.

Procedure for defining an APL function

At this point, the user will have defined all his operate functions, and have been instructed how to define his memory reference instructions. He will then be asked to refer to the instructional pamphlet to learn how to define a function in APL. This is a very straightforward procedure and easy to learn.

Definition of instruction set

(A) Definition of Operate Instructions

Once the user has been informed how to define an APL function he will be requested by the instructional pamphlet to write APL programs that will act as macros, a function definition using the pseudo-microinstructions as noted by the Simulator as the instruction, for each of his assembly language operate instructions. The names of these instructions are his choice, except that they must be unique and obey the same rules as tag names. He defines these functions with the pseudo-microinstructions that he has already specified and generated. He defines separate functions for each of his operate instructions. He will specify the parameters for his memory reference instructions in his final program, using the set form that has been shown to him.

If he discovers that he has forgotten to specify a particular function then he gets out of definition mode and loads the Simulator again. The Simulator recognizes that he wants to add a function by certain flags that were set in the program the first time through. It then transfers him to the section of reconfiguration. This section is explained fully in RECONFIGURE.

(B) Definition of Memory Reference Instructions

The user has an option of how he wants to define his memory reference instructions.

1. He can use the exact forms that are given to him by the Simulator for these operations.
2. He can use the forms given him by the Simulator to write function definitions for the instructions that he has that perform this same operation.

Final program definition

After the user has entered his macros for his operate instructions, and possibly his memory reference instructions, he will be instructed by the instructional pamphlet to enter his final program function definition. The instructions that he will use in this final program are the following:

1. The function names that he defined to act as his operate instructions is the operate macros.
2. The set forms that he was told to use for his

memory reference instructions, or the functions that he defined with these forms.
3. The APL jump forms that he was instructed to use to simulate his jump instructions.

Execution

After the user has completed the final program function definition, he is asked to type "INITIATE." This function initializes the system before each execution. This function does the following:

1. Sets a flag so that the Simulator will know that execution of the final program has happened for the first time.
2. Sets the projected time that the user has specified throughout the system to zero. (It will be incremented by the value the user specified in the parameters for each function definition. This is the theoretical time that his program would execute on the guest computer.)
3. Get the present value of the amount of IBM/360 50 time that has been used in this session. This time is an integer amount and is expressed as 1/60's of a second. After execution, the function REPORT uses this value to determine the amount of APL time used to execute his program.

In this way the user can have two clock times that he can work with to optimize his system.

Due to the many levels of function calls, it takes APL a long time to execute this kind of program. The example shows specific times.

Reconfigure and function editing

(A) Reconfigure

After the assembly language final program has executed, and the user has had an opportunity to check his clock times, he may want to reconfigure his machine. That is, he may want to redefine how the operate instructions are executed by his machine, or he may want to add some extra microinstructions that will give him added capabilities before redefining his operations.

The operations needed, and their use, is left entirely up to the user. The Simulator only gives him the ability to define such operations that his machine could have, and test them. Only the user knows how much he is willing to pay for certain speeds, and the kind of speed necessary for each single purpose.

If he wants to add extra operate microinstructions to his list of operations, then he loads the Simulator back into the workspace. The Simulator will know by the flag he set before executing his program for the first time that he wants to reconfigure. It transfers him to this section. The user is referred to the instructional pamphlet, and asked to answer the question appropriately. From his answers, the Simulator detects what type operation he wants to add, and transfers him to the appropriate sections. He can add as many operations as he wants. When completed, he signals the Simulator, and it tells him to redefine, or edit, the functions that he needs to improve or add.

(B) Function Editing

The user now is referred to the instructional pamphlet where it explains how to edit an APL function. APL has some very simple yet powerful function editing capabilities.

Then he is asked to redefine the functions whose execution time he needs to try to improve. After he has done this, he reverts back to EXECUTION to re-execute his program. This loop between EXECUTE and RECONFIGURE can continue for as long as there is enough space in the active workspace for him to store the extra parameter strings and any extra instructions that he defines. This takes up very little space, so this loop should be able to continue for as long as he wants to work with that particular system.

Unanticipated needs

There is a good possibility that someone will want a certain operation for which the Simulator will not provide facilities. The Simulator provides for this situation by asking the user if there is any capability that he was unable to specify because of a lacking on the part of the system. If there is, then the user is asked to enter a description of the capability that he needs, as well as his name and phone number. The caretaker of the system can check the memory locations that the message would be loaded into for each work space. If he finds that there is a certain need that should be added, then he can provide the added facility and then contact the person asking for that improvement.

Preserving a defined system

This system would not be very useful if the user had to redefine his machine every time he signed on. The Simulator allows the user to save his own particular system by entering several simple APL system commands. These commands and the order to be given are all shown in the instructional pamphlet.

Limitations

(1) Since this system was designed primarily as an educational tool, there was little emphasis put on simulation of the I-O. There are plans to improve this situation in the future.

(2) The system as implemented now does not allow for the times for jumps on target computer to be accumulated in the total execution time. This is only because of the fact that jumps had to be programmed in APL. Hopefully future versions of APL will allow for this to be done. It could be done at the present but would necessitate the user picking up a little more APL than could be expected.

CONCLUSIONS AND EVALUATIONS

This system, for the most part, operates as it was originally envisioned. There are several phases of the system that could not be implemented as hoped. This was mainly because of the APL version that this system was implemented in; however *A* *P*rogramming *L*anguage is the only language that the authors know of that could have possibly accomplished the desired needs so completely.

The interactive system makes the simulation very convenient for the user and changes the task of trying to simulate a system from a boring job into a very stimulating procedure. After becoming accustomed to the operating procedure of the system, the Simulator is almost like a new toy that is extremely pleasant to manipulate. This system gives the operator a feeling of power in expressing his ideas that he never had before.

In implementing this system, there were several times that the final design was almost completed, and then it was discovered that there was a need that could not be facilitated. Therefore, if this system had to be redesigned, a version of APL that allowed the following would be very useful:

1. Global Labels, that is the ability to branch from any point in one function to any point in another function.
2. Variables to be transferred from one function to another as their literal self rather than their values.

REFERENCES

1 P BERRY
APL/360 primer
IBM Technical Publications Department
White Plains New York 1969

2 *Small computer handbook*
Digital Equipment Corporation
Maynard Massachusetts 1967

3 I FLORES
The logic of computer arithmetic
Prentice-Hall Inc Englewood Cliffs New Jersey 1963

4 H W GSCHWIND
Design of digital computers, an introduction
Springer-Verlag New York 1967

5 S S HUSSON
Microprogramming manual for the IBM System/360 Model 50
International Business Machines Corporation
Poughkeepsie New York 1967

6 *APL/360 user's manual*
IBM Technical Publications Department
White Plains New York 1968

7 T C LOWE
An educational tool for use in the introduction of digital computing
Informatics Inc 1967

8 J MARTIN
Design of real-time computer systems
Prentice-Hall Inc Englewood Cliffs New Jersey 1967

9 S PAKIN
APL/360 reference manual
Science Research Associates Chicago Illinois 1968

APPENDIX 1—PDP-8 SUBSET EXAMPLE

The PDP-8 is a small computer manufactured by DIGITAL COMPUTER CORPORATION. It has a word size of 12 bits, and a memory unit that will hold 4096 words. The PDP-8 has the following registers:

Accumulator	12 bits
Memory Address	12 bits
Memory Buffer	12 bits
Program Counter	12 bits
Link	1 bit
Switch Register	12 bits
Instruction	3 bits

The registers that it uses to perform its arithmetic operations are not included. The registers that need to be simulated because of programming operations that involve them are the:

Accumulator
Link
Switch Register

Memory reference instructions

The memory reference instructions that are simulated are the following:

AND Y

The AND operation is performed between the contents of memory location Y and the contents of the ACCUMULATOR. The result is left in the ACCUMULATOR, the original content of the ACCUMULATOR is lost, and the content of Y is restored. Corresponding bits of the ACCUMULATOR and Y are operated on independently. This instruction, often called mask or extract, can be considered as a bit-by-bit multiplication.

TAD Y

The content of memory location Y is added to the content of the ACCUMULATOR. The result of the addition is held in the ACCUMULATOR, the original content of the ACCUMULATOR is lost, and the content of Y is restored.

ISZ Y

The content of memory location Y is incremented by one. If the resultant content of Y equals zero, there is a skip over one instruction, and the next instruction is executed. If the content is not equal to zero then the next instruction in order is executed.

DCA Y

The content of the ACCUMULATOR is deposited in core memory at location Y and the ACCUMULATOR is cleared. The previous content of memory location Y is lost.

Operate instructions

The operate instructions that are simulated are the following:

IAC

The content of the ACCUMULATOR is incremented by one.

RAL

The content of the ACCUMULATOR is rotated one bit to the left with the content of the LINK. The shift is left, end-around.

RTL

Same as RAL, except that shift occurs twice, that is shift two bits instead of one.

RAR

The content of the ACCUMULATOR is rotated one binary bit to the right with the content of the LINK. The shift is right, end-around.

RTR

Same as RAR except that shifts two bits instead of one.

CMA

The contents of the ACCUMULATOR are complemented.

CML

The contents of the LINK are complemented.

CIA

This is a combination of two instructions. First CMA and then IAC. This operation converts the contents of the ACCUMULATOR to its equivalent negative value.

CLL

The content of LINK is cleared to contain a 0.

STL

The content of LINK is set to contain a 1.

CLA

The content of the ACCUMULATOR is cleared to contain all 0's.

STA

The content of the ACCUMULATOR is set to contain all 1's.

OSR

The inclusive OR operation is performed between the content of the ACCUMULATOR and the content of the Switch Register. The result is left in the ACCUMULATOR, the original content of the ACCUMULATOR is lost and the contents of the Switch Register is unaffected.

SKP

Skip the next instruction.

SNL

Skip the next instruction if the content of LINK is not equal to 0.

SZL

Skip the next instruction if the content of LINK is 0.

SZA

Skip the next instruction if the content of the ACCUMULATOR is 0.

SNA

Skip the next instruction if the content of the ACCUMULATOR is not equal to 0.

SMA

Skip the next instruction if the content of the ACCUMULATOR is negative.

SPA

Skip the next instruction if the content of the ACCUMULATOR is non-negative.

This example does not attempt to simulate the PDP-8 as a whole, but only a subset of it. Nor does it attempt to show the entire capabilities of the simulator system. This is intended only to provide a simple straightforward example of the use of the simulator with no reconfigurations, diagnostics, or programming aids included.

The questions that the user answers as well as the directions that he is to follow are in the instruction booklet. Space restriction would not allow for inclusion of this booklet in this paper.

```
                         INTRODUCTION

      ESTABLISH
      SIMULATE
THIS IS A SYSTEM THAT HOPEFULLY WILL ALLOW YOU TO GENERATE A
       SIMULATOR
FOR YOUR SINGLE PURPOSE COMPUTER.
NOW PLEASE REFER TO TO INSTRUCTION PHAMPHLET,SECTION 1, TO G
      ET AN OVERALL PICTURE OF WHAT
YOU ARE ABOUT TO DO.
ENTER THE RETURN WHEN READY TO PROCEED.


                  DESCRIPTION OF MACHINE


WHAT IS THE SIZE OF YOUR MEMORY BANK?.
⎕:
      4096
WHAT IS THE BIT SIZE OF YOUR INSTRUCTION WORD?
⎕:
      12
NOW PLEASE ENTER ALL THE REGISTERS THAT YOU HAVE, ONE AT A T
      IME, THEN THE BITLENGTH OF EACH.  WHEN YOU
HAVE ENTERED ALL YOUR REGISTERS, HIT ONLY THE RETURN KEY.
ENTER REGISTER NAME, HIT RETURN, THEN ENTER REGISTER BIT LEN
      GTH.
ACC
THIS REGISTER WILL BE REFERRED TO FROM NOW ON AS REG1.  PLEA
      SE MAKE RECORD OF THIS.
WHAT IS THE BIT LENGTH OF THIS REGISTER?
⎕:
      12
NOW ENTER THE NAME OF YOUR NEXT REGISTER. IF COMPLETED, ENTE
      R ONLY THE CARRIAGE RETURN.
MA
THIS REGISTER WILL BE REFERRED TO FROM NOW ON AS REG
      2.  PLEASE MAKE RECORD OF THIS.
WHAT IS THE BIT LENGTH OF THIS REGISTER?
⎕:
      12
NOW ENTER THE NAME OF YOUR NEXT REGISTER. IF COMPLETED, ENTE
      R ONLY THE CARRIAGE RETURN.
MB
THIS REGISTER WILL BE REFERRED TO FROM NOW ON AS REG
      3.  PLEASE MAKE RECORD OF THIS.
WHAT IS THE BIT LENGTH OF THIS REGISTER?
⎕:
      12
NOW ENTER THE NAME OF YOUR NEXT REGISTER. IF COMPLETED, ENTE
      R ONLY THE CARRIAGE RETURN.
PC
THIS REGISTER WILL BE REFERRED TO FROM NOW ON AS REG
      4.  PLEASE MAKE RECORD OF THIS.
WHAT IS THE BIT LENGTH OF THIS REGISTER?
⎕:
      12
NOW ENTER THE NAME OF YOUR NEXT REGISTER. IF COMPLETED, ENTE
      R ONLY THE CARRIAGE RETURN.
LINK
THIS REGISTER WILL BE REFERRED TO FROM NOW ON AS REG
      5.  PLEASE MAKE RECORD OF THIS.
WHAT IS THE BIT LENGTH OF THIS REGISTER?
□:
      1
NOW ENTER THE NAME OF YOUR NEXT REGISTER. IF COMPLETED, ENTE
      R ONLY THE CARRIAGE RETURN.
SWITCH
THIS REGISTER WILL BE REFERRED TO FROM NOW ON AS REG
      6.  PLEASE MAKE RECORD OF THIS.
WHAT IS THE BIT LENGTH OF THIS REGISTER?
□:
      12
NOW ENTER THE NAME OF YOUR NEXT REGISTER. IF COMPLETED, ENTE
      R ONLY THE CARRIAGE RETURN.
IR
THIS REGISTER WILL BE REFERRED TO FROM NOW ON AS REG
      7.  PLEASE MAKE RECORD OF THIS.
WHAT IS THE BIT LENGTH OF THIS REGISTER?
□:
      3
NOW ENTER THE NAME OF YOUR NEXT REGISTER. IF COMPLETED, ENTE
      R ONLY THE CARRIAGE RETURN.

WOULD YOU LIKE TO HAVE A HARD COPY OF THE REGISTER NAMES AND
       BITLENGTHS, IN NUMERICAL ORDER,
AS THE COMPUTER HAS THEM IDENTIFIED.   ANSWER YES OR NO.
YES
ACC MA MB PC LINK SWITCH IR
12  12  12  12  1  12  3
NOW, PLEASE ENTER ALL THE COUNTERS THAT YOU HAVE, ONE AT A T
      IME, THEN THE BIT
LENGTH OF EACH. WHEN YOU HAVE ENTERED THEM ALL, HIT ONLY THE
       RETURN KEY.
ENTER COUNTER NAME, HIT RETURN, THEN ENTER COUNTER BIT LENGT
      H.

WOULD YOU LIKE A HARD COPY OF THE COUNTER NAMES, IN NUMERICA
      L ORDER, AND THEIR BITLENGTHS
AS THE COMPUTER HAS THEM IDENTIFIED.   ANSWER YES OR NO.
NO
REFER NOW TO SECTION 1.5 OF THE APPENDIX FOR INSTRUCTIONS ON
       WHAT TO DO NEXT.

□:
    ( Refer to Booklet about Option of Memory Addressing)
```

```
⎕:
      1
WHAT IS THE SIZE OF THE BLOCK OF MEMORY SPACE THAT YOU WILL
      NEED TO EXECUTE YOUR PROGRAM?
⎕:
      50
THESE LOCATIONS WILL BE REFERRED TO FROM NOW ON AS MEMLOC1-M
      EMLOC50 THEY ARE 12

BITS IN LENGTH
THE FOLLOWING IS A LIST OF VARIABLES THAT YOU CANNOT USE AS
      VARIABLES OR LABELS IN YOUR PROGRAMS.
APLNOW APLNOW1 TERK TERK1 TERK2 TERK3 TERK4
TERK5 TERK6 TERK7 TERK8 TERK9 TERK10 TERK11 TERK12 TERK13 TE
      RK14
TIME STRING COUNTER REGISTER MEMLOC VARIABLE WORDSIZ BITLENG
      TH
BITLINGTH FLAG1 FLAG2
```

- APPENDED STORAGE DEVICES

```
ARE THERE ANY REGISTERS APPENDED TO ONE ANOTHER.
YES
THE PRESENT SYSTEM ALLOWS NO MORE THAN THREE STORAGE DEVICES
       TO BE APPENDED TOGETHER.  THAT IS
IT WILL ALLOW NO MORE THAN THREE STORAGE DEVICES TO BE SHIFT
      ED AS A WHOLE.  IF YOU NEED OR WANT MORE THAN THIS
SIMPLY CONTACT JOEL BRAME.

NOW, PLEASE REFER TO YOUR INSTRUCTION PHAMPHLET,SECTION 2, T
      O FIND OUT WHAT QUESTIONS
YOU ARE TO ANSWER AND THE FORM THAT THESE ANSWERS ARE TO BE
      IN .

⎕:    9999
DOES YOUR COMPUTER HAVE ANY TRANSFER OPERATIONS THAT MUST BE
       DONE SIMULTANEOUSLY?
NO
```

- SHIFT OPERATIONS

```
DOES YOUR COMPUTER HAVE SHIFT OPERATIONS?
YES
NOW PLEASE REFER TO SECTION 4 OF YOUR INSTRUCTION PHAMPHLET
      TO RECEIVE FURTHER DIRECTIONS
⎕:    1 5 6 4 1 0 1 1 1500
THIS SHIFT OPERATION WILL BE REFERRED TO AS
       SHIFT STRING[0+⍳9]
PLEASE MAKE NOTE OF THIS FACT.
IF YOU HAVE MORE OPERATIONS OF THIS TYPE, THEN ANSWER THE AB
      OVE QUESTIONS AGAIN.  IF YOU
ARE COMPLETED, THEN ENTER ONLY THE NUMBER 9999.
⎕:    1 5 6 5 1 0 1 1 1500
THIS SHIFT OPERATION WILL BE REFERRED TO AS
       SHIFT STRING[9+⍳9]
PLEASE MAKE NOTE OF THIS FACT.
IF YOU HAVE MORE OPERATIONS OF THIS TYPE, THEN ANSWER THE AB
      OVE QUESTIONS AGAIN.  IF YOU
ARE COMPLETED, THEN ENTER ONLY THE NUMBER 9999.
⎕:
```

```
LOGICAL OPERATIONS
      9999
NOW PLEASE REFER TO SECTION 5 OF YOUR INSTRUCTION PHAMPHLET
      TO RECEIVE FURTHER DIRECTIONS.
⎕:
      1 5 7 0 0 1 5
YOUR ANSWER IS IN THE WRONG FORM. REREAD THE INSTRUCTIONS A
      ND ANSWER THE QUESTIONS FOR THIS OPERATION AGAIN.

⎕:
      1 5 7 0 0 1 5 1500
THIS OPERATION WILL BE REFERRED TO AS
       LOGOP  STRING[18+⍳8]
PLEASE MAKE NOTE OF THIS FACT.
IF YOU HAVE MORE OPERATIONS OF THIS TYPE, THEN ANSWER THE QU
      ESTIONS AGAIN.  IF YOU
ARE COMPLETED, THEN ENTER THE NUMBER 9999.
⎕:
      1 1 7 0 0 1 1 1500
THIS OPERATION WILL BE REFERRED TO AS
       LOGOP  STRING[26+⍳8]
PLEASE MAKE NOTE OF THIS FACT.
IF YOU HAVE MORE OPERATIONS OF THIS TYPE, THEN ANSWER THE QU
      ESTIONS AGAIN.  IF YOU
ARE COMPLETED, THEN ENTER THE NUMBER 9999.
⎕:
      1 1 5 1 7 1 1 1500
THIS OPERATION WILL BE REFERRED TO AS
       LOGOP  STRING[34+⍳8]
PLEASE MAKE NOTE OF THIS FACT.
IF YOU HAVE MORE OPERATIONS OF THIS TYPE, THEN ANSWER THE QU
      ESTIONS AGAIN.  IF YOU
ARE COMPLETED, THEN ENTER THE NUMBER 9999.
⎕:
      9999

- INCREMENTS DECREMENTS

NOW PLEASE REFER TO SECTION 6 OF YOUR INSTRUCTION PHAMPHLET
      TO RECEIVE FURTHER DIRECTIONS.
⎕:
   1 1 4 1 1500
THIS OPERATION WILL BE REFERRED TO AS
       INCREM STRING[42+⍳5]
PLEASE MAKE NOTE OF THIS FACT.
IF YOU HAVE MORE OPERATIONS OF THIS TYPE, THEN ANSWER THE QU
      ESTIONS AGAIN.  IF YOU
ARE COMPLETED THEN ENTER THE NUMBER 9999.
⎕:
      9999
NOW PLEASE REFER TO SECTION 7 OF YOUR INSTRUCTION PHAMPHLET
      TO RECEIVE FURTHER DIRECTIONS.
⎕:
```

SETS AND CLEARS

```
      1 5 4 1 1 1500
THIS OPERATION WILL BE REFERRED TO AS
       SET STRING[47+⍳6]
PLEASE MAKE NOTE OF THIS.
IF YOU HAVE ANY MORE OPERATIONS OF THIS TYPE, THEN ANSWER TH
      E QUESTIONS AGAIN, IF COMPLETED
THEN ENTER THE NUMBER 9999.
⎕:
    1 5 5 1 1 1500
THIS OPERATION WILL BE REFERRED TO AS
       SET STRING[53+⍳6]
PLEASE MAKE NOTE OF THIS.
IF YOU HAVE ANY MORE OPERATIONS OF THIS TYPE, THEN ANSWER TH
      E QUESTIONS AGAIN, IF COMPLETED
THEN ENTER THE NUMBER 9999.
⎕:
   1 1 5 1 12 1500
THIS OPERATION WILL BE REFERRED TO AS
       SET STRING[59+⍳6]
PLEASE MAKE NOTE OF THIS.
IF YOU HAVE ANY MORE OPERATIONS OF THIS TYPE, THEN ANSWER TH
      E QUESTIONS AGAIN, IF COMPLETED
THEN ENTER THE NUMBER 9999.
⎕:
      1 1 4 1 12 1500
THIS OPERATION WILL BE REFERRED TO AS
       SET STRING[65+⍳6]
PLEASE MAKE NOTE OF THIS.
IF YOU HAVE ANY MORE OPERATIONS OF THIS TYPE, THEN ANSWER TH
      E QUESTIONS AGAIN, IF COMPLETED
THEN ENTER THE NUMBER 9999.
⎕:
      9999
WERE THERE ANY OPERATIONS THAT YOU WERE UNABLE TO SPECIFY FO
      R YOUR COMPUTER BECAUSE OF A LACKING ON
THE PART OF THE SIMULATOR.   ANSWER YES OR NO.
NO
PLEASE REFER TO THE PHAMPHLET YOU HAVE WITH YOU TO LEARN WHA
      T TO DO NEXT.  REFER TO THE MEMORY REFERENCE
INSTRUCTIONS,APPENDIX1,.  IT WILL EXPLAIN EVERYTHING THAT YO
      U ARE TO DO TO CONTINUE THIS SIMULATION PROCESS.
      )CLEAR
CLEAR WS
      )COPY EXECUTE:KEY
SAVED   15.22.05 07/25/70
      )COPY SIMULATOR:KEY NEEDS
SAVED   17.30.13 07/27/70
```

The following is an example of taking the pseudo-microinstructions set desired. (Many pseudo-microinstructions can be used to define the execution of any one instruction). The user is instructed how to define APL functions in the instruction booklet.

```
      ∇CIA
[1]   REGISTER[1]←-REGISTER[1]
[2]   ∇

      ∇CLA
[1]   SET STRING[59+⍳6]
[2]   ∇

      ∇CLL
[1]   SET STRING[53+⍳6]
[2]   ∇

      ∇CMA
[1]   LOGOP STRING[26+⍳8]
[2]   ∇

      ∇CML
[1]   LOGOP STRING[18+⍳8]
[2]   ∇

      ∇DCA KKK
[1]   STORE 1,KKK,1500
[2]   CLA
[3]   ∇

      ∇IAC
[1]   INCREM STRING[42+⍳5]
[2]   ∇
```

```
        ∇OSR
[1]     LOGOP STRING[34+⍳8]
[2]     ∇

        ∇RAL
[1]     SHIFT STRING[0+⍳9]
[2]     ∇

        ∇RTL
[1]     RAL
[2]     RAL
[3]     ∇

        ∇RAR
[1]     SHIFT STRING[9+⍳9]
[2]     ∇

        ∇RTR
[1]     RAR
[2]     RAR
[3]     ∇

        ∇STA
[1]     SET STRING[65+⍳6]
[2]     ∇

        ∇STL
[1]     SET STRING[47+⍳6]
[2]     ∇

        ∇TAD LLL
[1]     ADD  2,LLL, 1 1 3000
[2]     ∇

   SKP      →2+I26

   SNL      →(REGISTER[5]≠0)/2+I26

   SZL      →(REGISTER[5]=0)/2+I26
```

```
SZA          →(REGISTER[1]=0)/2+I26

SNA          →(REGISTER[1]≠0)/2+I26

SMA          →(REGISTER[1]<0)/2+I26

SPA          →(REGISTER[1]≥0)/2+I26

ISZ Y        INCREM 3,Y, 4 1 3000
             →(MEMLOC[Y]=0)/2+I26
```

THE FOLLOWING IS A SORT ROUTINE WRITTEN IN THE LANGUAGE THAT WE HAVE DEFINED.IT IS A SUBSET OF THE PDP-8 ASSEMBLY CODE OTHER THAN THE JUMP INSTRUCTIONS, AND ASSIGNING VALUES TO THE VARIABLES THAT ARE USED AS FUNCTION PARAMETERS, AND LABELS.

```
     ∇ PROGRAN;UU;SS;TT
[1]     UU←2
[2]     RR←1
[3]   LABEL:READ RR,1000
[4]     RR←RR+1
[5]     INCREM 3 1 4 1 3000
[6]     →(MEMLOC[1]=0)/2+⍳26
[7]     →LABEL
[8]     →ESTAB
[9]   LABEL1:INCREM 3 13 4 1 3000
[10]    →(MEMLOC[13]=0)/2+⍳26
[11]    →2+⍳26
[12]    →ESTAB
[13]    CLA
[14]    TAD SS
[15]    CIA
[16]    TAD SS+1
[17]    SS←SS+1
[18]    →(REGISTER[1]<0)/2+⍳26
[19]    →LABEL1
[20]    CLA
[21]    TAD SS-1
[22]    DCA 16
[23]    TAD SS
[24]    DCA SS-1
[25]    TAD 16
[26]    DCA SS
[27]    TAD 14
[28]    IAC
[29]    DCA 14
[30]    →LABEL1
[31]  ESTAB:CLA
[32]    TAD 14
[33]    →(REGISTER[1]≠0)/2+⍳26
[34]    →LABEL2
[35]    CLA
[36]    TAD 15
[37]    DCA 14
[38]    TAD 12
[39]    DCA 13
[40]    SS←2
[41]    →LABEL1
```

```
[42] LABEL2:PRINT UU, 10 2000
[43]  UU←UU+1
[44]  INCREM 3 12 4 1 3000
[45]  →(MEMLOC[12]=0)/2+ɪ26
[46]  →LABEL2
[47]  →0
```

```
      INITIATE
      PROGRAN

PLEASE ENTER THE VALUE FOR MEMLOC1
⎕:
      ¯15
PLEASE ENTER THE VALUE FOR MEMLOC2
⎕:
      3
PLEASE ENTER THE VALUE FOR MEMLOC3
⎕:
      9
PLEASE ENTER THE VALUE FOR MEMLOC4
⎕:
      6
PLEASE ENTER THE VALUE FOR MEMLOC5
⎕:
      15
PLEASE ENTER THE VALUE FOR MEMLOC6
⎕:
      1
PLEASE ENTER THE VALUE FOR MEMLOC7
⎕:
      8
PLEASE ENTER THE VALUE FOR MEMLOC8
⎕:
      10
```

```
PLEASE ENTER THE VALUE FOR MEMLOC9
⎕:
      20
PLEASE ENTER THE VALUE FOR MEMLOC10
⎕:
      2
PLEASE ENTER THE VALUE FOR MEMLOC11
⎕:
      7
PLEASE ENTER THE VALUE FOR MEMLOC12
⎕:
      ¯10
PLEASE ENTER THE VALUE FOR MEMLOC13
⎕:
      0
PLEASE ENTER THE VALUE FOR MEMLOC14
⎕:
      1
PLEASE ENTER THE VALUE FOR MEMLOC15
⎕:
      0
PLEASE ENTER THE VALUE FOR MEMLOC16
⎕:
      0


MEMLOC2       0  0  0  0  0  0  0  0  0  0  0  0  0
      0  1
MEMLOC3       0  0  0  0  0  0  0  0  0  0  0  0  0
      0  2
MEMLOC4       0  0  0  0  0  0  0  0  0  0  0  0  0
      0  3
MEMLOC5       0  0  0  0  0  0  0  0  0  0  0  0  0
      0  6
MEMLOC6       0  0  0  0  0  0  0  0  0  0  0  0  0
      0  7
MEMLOC7       0  0  0  0  0  0  0  0  0  0  0  0  0
      0  8
MEMLOC8       0  0  0  0  0  0  0  0  0  0  0  0  0
      0  9
MEMLOC9       0  0  0  0  0  0  0  0  0  0  0  0  0
      1  0
MEMLOC10       0  0  0  0  0  0  0  0  0  0  0  0  0
      1  5
MEMLOC11       0  0  0  0  0  0  0  0  0  0  0  0  0
      2  0
      REPORT

THE ACCUMALATED TIME THAT YOU SPECIFIED FOR EACH OPERATION FOR T
      HIS EXECUTION IS    0.0015825
THE FOLLOWING IS THE AMOUNT OF APL TIME, IN 1/60 OF A SECOND, TH
      AT EXECUTION
FOR THE PREVIOUS PROGRAM TOOK    14825
THE FOLLOWING IS A LIST OF ALL THE BASIC FUNCTION NAMES AND THE
      NUMBER OF TIMES THEY WERE EXECUTED

TRANSFER 0SIMULFER 0SHIFT 0LOGOP 0
INCREM 126SET 205ADD 249SUBTRACT 0MULTIPLY 0
DIVIDE 0LOAD 0STORE 96DECLARE 0READ 16
PRINT 10
```

Multiband automatic test equipment—A computer controlled check-out system

by TERUHISA KURODA

McKinsey & Company, Inc.
New York, New York

and

THOMAS C. BUSH

Sanders Associates, Inc.
Plainview, New York

INTRODUCTION

The major problem facing organizations using electronic devices is in test and maintenance of the equipment. Due to the high cost of the equipment and demands made on its maximum utilization, fast and reliable testing procedures are required to minimize downtime for repair and calibration.

The purpose of this paper is to describe the computer and software system developed for MATE, the Multiband Automatic Test Equipment. The MATE System was developed to test naval avionic equipments. Information related to these naval avionic equipments is both "classified" and irrelevant to the MATE systems in the software sense and, therefore, is not included in this paper.

SYSTEM OVERVIEW

The MATE System is self-contained and consists of all software and hardware necessary for its operation.

Basic processes in the use of the MATE System are shown in Figure 1. The test engineer first performs the necessary analyses to determine what tests must be made on a particular device. These tests are then organized by a 3-digit test-sequence number with a 3-digit test-series number. These tests are then flow charted and coded, taking into account the rules of the MATE programming language. Since this programming language is English-like, programs are called English Language Programs or ELPs. ELPs are punched on paper tape and catalogued on the disk, unit under test is connected to MATE; and tests are performed.

Figure 2 shows the components of the MATE System from the programming point of view. It shows how major programs such as the Executive Program, Language Processor, and Utility Programs interact with various input-output devices in the system. Measuring devices and stimuli-generating equipment appear as input-output units. The use of pertinent data on the disk such as indices, tables, and libraries have been indicated.

Hardware connections are shown in Figure 3. These connections are rather straightforward, except for the use of two busses to transfer data between test modules and the computer. The address bus is used to send the device address of the test module to be referenced in the input-output operation. Data from test modules are sent across the data bus.

Execution of tests is shown by Figure 4. It shows how various data stored on the disk are used.

The picture of the MATE station is shown in Figure 5. It can be seen how compact and self-sufficient the system is.

MANUAL CHECKPOINT INADEQUATE

The basic testing process consists of sending a simple stimulus to the unit under test, receiving the resulting response, and measuring the response against a standard. As the complexity of the devices to be tested increases, there is a corresponding rise in the number and interrelationships among stimuli and responses. The traditional manual testing approach, with the use of lash-ups and semiautomatic control devices, is totally

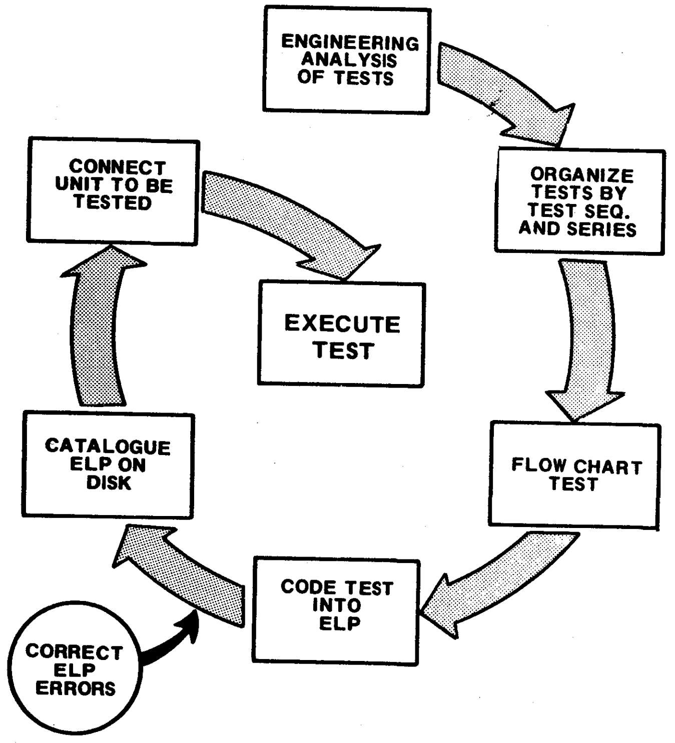

ENGINEERING ANALYSIS OF TESTS
ORGANIZE TESTS BY TEST SEQ. AND SERIES
FLOW CHART TEST
CODE TEST INTO ELP
CATALOGUE ELP ON DISK
CONNECT UNIT TO BE TESTED
EXECUTE TEST
CORRECT ELP ERRORS

Figure 1—MATE process chart

inadequate for testing avionic equipments because of the:

- Large amount of time spent in testing
- Inherent errors introduced in a manual process
- Shortage of required personnel and number of test stations to serve testing needs within a time period
- Tedious method of locating faults
- Inaccurate method of recording test performance.

AUTOMATED SYSTEM IS THE ANSWER

With the declining cost of computers and peripherals, and emergence of analog-digital technology, it has become possible to build a computer-monitored automatic check-out system. The MATE System is a self-contained test station consisting of test equipments, computer, and operator control stations. This system can be driven in automatic mode by the computer or in manual mode by paper tape or operator action. The manual mode of operation is not described, since it does not require the computer, and, therefore, is not relevant to this discussion.

In the automatic mode, the MATE test station is driven by **E**nglish **L**anguage **P**rograms (ELP), written by test engineers and catalogued on disk storage, which is operated upon by the MATE software. The test process is directed by the operator who can intervene and override any preplanned sequence of test or parameters.

The MATE System was designed in a modular fashion to facilitate modification and/or growth. Additional stimuli generators and measurement devices can be accommodated by adding new programs to the MATE software. New units under test, sequence of test or test parameters can be incorporated by writing new ELPs.

The MATE System, compared to manual methods, will:

- Reduce check-out time
- Apply a uniform test standard to units under test
- Reduce the number of check-out personnel
- Reduce training time for personnel
- Record test results on hard copy
- Decrease errors introduced by operator

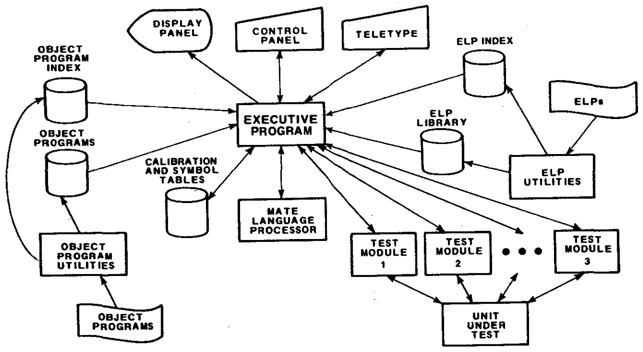


Figure 2—MATE system components

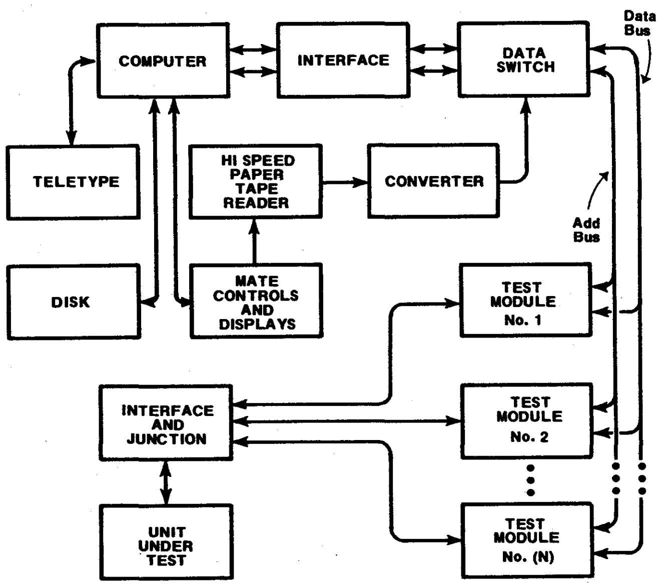


Figure 3—Hardware connections

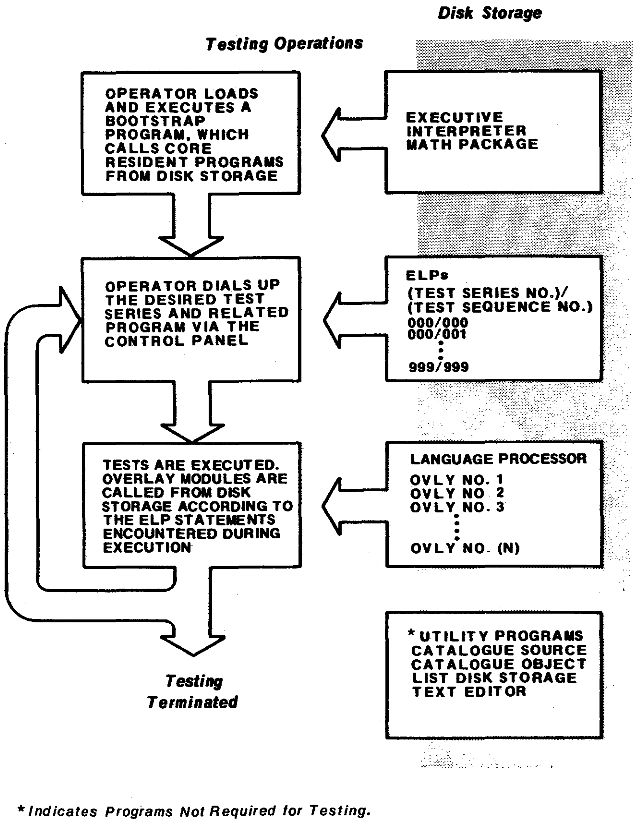
- Reduce skill level required of operator
- Isolate faults to a level suitable for further manual tests
- Permit the engineer to direct testing procedures via an easily usable test language
- Reduce the number of test stations required to handle the same load under a manual process.

HARDWARE DESCRIPTION

The hardware of the MATE System consists of a computer, various test modules and a control/display panel. These system components appear like any other I/O device to the computer. Thus, the MATE hardware operates a computer with many I/O devices.

Computer configuration

- Varian 620/i computer
- 8192 word storage (16 Bit word)
- KSR-33 Teleprinter
- 131K words of disk storage
- 200 char/sec paper tape.


Disk Storage
Testing Operations
OPERATOR LOADS AND EXECUTES A BOOTSTRAP PROGRAM, WHICH CALLS CORE RESIDENT PROGRAMS FROM DISK STORAGE
EXECUTIVE INTERPRETER MATH PACKAGE
OPERATOR DIALS UP THE DESIRED TEST SERIES AND RELATED PROGRAM VIA THE CONTROL PANEL
ELPs (TEST SERIES NO.)/ (TEST SEQUENCE NO.) 000/000 000/001 999/999
TESTS ARE EXECUTED. OVERLAY MODULES ARE CALLED FROM DISK STORAGE ACCORDING TO THE ELP STATEMENTS ENCOUNTERED DURING EXECUTION
LANGUAGE PROCESSOR OVLY NO. 1 OVLY NO 2 OVLY NO. 3 OVLY NO. (N)
Testing Terminated
*UTILITY PROGRAMS CATALOGUE SOURCE CATALOGUE OBJECT LIST DISK STORAGE TEXT EDITOR

Indicates Programs Not Required for Testing.

Figure 4—Test organization

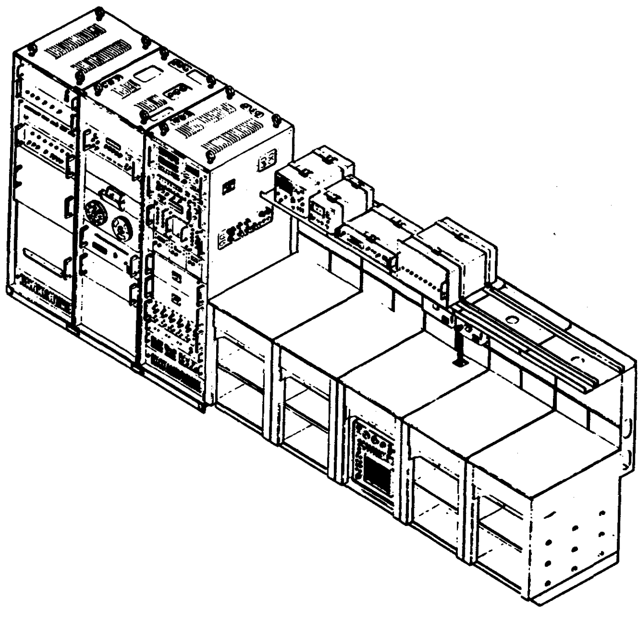


Figure 5—MATE station

Test modules

A test module is an electronic test instrument or stimulus generating device such as digital voltmeter or a sine square generator. These modules must all appear as similar devices which can:

- Communicate with the computer by digital word transfer
- Convert digital information to analog signals
- Convert analog measurements to digital for transfer to the computer
- Respond to and activate discrete signals for system control functions
- Respond to strobing by a unique device address.

Typical test modules contain the following features:

- Modular packaging
- Built-in self-test circuitry which continuously monitors the module outputs for failure
- Control panel programmed response indicators
- Manual or automatic operation
- Control panel alignment and self-check output.

Control panel

From the standpoint of the operator/user, the control panel is the central point of all testing activities. Consequently, the control panel was designed to accommodate the testing needs of the operator. It has no controls related to the computer. From the control panel the operator can:

- Initiate tests by retrieving and executing ELPs stored on the disk
- Override any faults which may be encountered while executing tests
- Repeat any test
- Repeat any measurement
- Interrupt the testing sequence, manually alter any of the test parameters and continue testing
- Acknowledge testing results (GO/NOGO, run, etc.)
- Execute testing ELPs one instruction at a time
- Reset the entire system.

Display panel

The display panel, although not a control function, is physically located adjacent to the control panel and contains all the visual displays required in testing. The panel can display:

- Test number currently being performed
- Operator instructions (i.e., operator decision, etc.)
- Next ELP instruction step to be executed (when in single step mode)
- Alarm indications to alert the operator of hardware malfunctions.
- Condition of I/O RF switching paths via an indicator lamp matrix
- Test results and values
- Indications and/or displays provided by the unit under test.

MATE PROGRAMMING LANGUAGE

The MATE programming language is a special-purpose language designed for use by test engineers. Since its format is cryptic English, we refer to it as ELP (**E**nglish **L**anguage **P**rograms). Familiar engineering expressions are retained while introducing some necessary programming concepts like "go to" and "execute subroutine." The language is easy to learn and requires no previous knowledge or exposure to programming.

The basic element of the language is a statement. An ELP statement takes the following skeleton format with the contents depending on the verb:

- Statement number
- Verb
- Contents.

The English elements of the MATE language consist of verbs (test actions), test module names (nouns), parameters (adjectives), and units of measure (other adjectives). In order to reduce errors in coding, all English words can be used in either abbreviated or full form. Abbreviations require using only the leading characters required to make the word unique. For example, the word "calibrate" can be expressed as CA, "step attenuator" as STEPAT, "RF band" as RFB, etc.

List of verbs

The verbs in the MATE language consist of the following:

MATE—VERB ACTION TABLE

ELP Delimiters	BEGIN
	TERMINATE
Input/Output	READ
	DISPLAY
	SETUP
	CALIBRATE
	UPDATE
	LOOKUP
	WAIT FOR
	SYSTEM RESET
Decision	IF
Subroutines	PERFORM
	DEFINE
	END
Computation	LET
Branch	GOTO

Function of verbs

Each verb in the MATE serves specific functions and requires certain information as described in the following:

ELP delimiters

The BEGIN and TERMINATE verb serves to define the start and end of a test sequence. It is identified uniquely by a 3-digit test-series and 3-digit test-sequence number.

Input output verbs

The following verbs perform I/O operations to the teleprinter, disk, test modules, control panel, or display panel.

The SETUP verb is used to send stimuli to the test module and the READ verb is used to obtain the resultant response from the test module reflecting a test made on the unit under test.

The DISPLAY verb is used to display information on the teleprinter or display panel.

Symbol tables are referenced by the UPDATE and LOOKUP verbs. These verbs are I/O related since symbol tables are stored on the disk.

The WAIT FOR verb is used to wait for the occurrence of hardware responses. Such as I/O complete or test module ready interrupts, and operator action like a Go/NOGO decision.

The SYSTEM RESET verb is used to initialize all test modules.

Examples of I/O verbs are:

```
001  SETUP, STEP ATTN (ATTN=59 DB)
002  READ, PWR OUT METER
003  DISPLAY, MESSAGE ('VOLTS', C, K,
     LO LIM)
004  UPDATE, TABl ('LABl') 'INC'
005  LOOKUP, TABl ('LABl')
006  WAIT FOR, RESUME
007  SYSTEM RESET
008  CALIBRATE, AUTO LEVEL (FREQ=
     'LABL')
```

Decision

The IF verb is used to ask the true-false question based on the mathematical expression greater than (>), less than (<), equal (=), greater than or equal to (≥), and less than or equal to (≤). If the answer is true, a jump is made to the specified ELP statement number. A false answer will cause the execution of the next sequential ELP statement.

An example of the IF verb is:

```
010  IF, (2.5)≤'LABEL'<(3.0), GOTO (S100)
```

Subroutines

Subroutines can be included in the ELP provided they are defined at the beginning by the DEFINE verb and at the end of the END verb. The DEFINE verb is used to list those parameters in the subroutine which can be varied by the calling program.

The execution of a subroutine is done via the PERFORM verb. It can also deliver values for parameters of the subroutine.

For example:

```
010  PERFORM (S100) (33.25)
100  DEFINE ('X')
101  LET, 'X'='X'+1
102  END
```

Computation

Computation is indicated by the verb LET. Operators available are add (+), subtract (−), multiply (*), divide (/), and exponentiate (↑). Any mathematical expression can be stated, using these operators. The interpretation of the expression is in left-to-right order.

Examples are:

```
020  LET, X=A↑B+CX
030  LET, X=A+B/C
```

Branch

A branch in the ELP is accomplished by the use of the GOTO verb. The destination can be specified as an ELP statement number, a new test sequence number, or a work indicating the end of a test series.

Examples of the GOTO verb are:

```
100  GOTO, (S200)
200  GOTO, (T125)
300  GOTO, (EXEC)
```

SOFTWARE DESCRIPTION

After a survey of the computers which could be used in the MATE System (in terms of size, cost, and system requirements), the Varian 620i was selected. The Varian 620i, being a mini-computer, has software and peripheral equipment which is either too general or too specialized for a given application. Often I/O devices, such as high-speed tape readers, are made by different companies and frequently require special interfaces to the computer. Thus, what could have been an applications programming task explodes into a major software development project. Further burden is imposed by the sparcity of mini-computer simulators. Software development, therefore, must be done directly on these computers which have only limited debugging aids. The MATE System was no exception.

The software developed included:

- Executive program
- Input/Output handler for disk, high speed tape reader and teleprinter

- MATE software (object program) catalogue and maintenance routines
- English Language Program (ELP) catalogue and maintenance routines
- Utility programs to print contents of core and disk storage
- Mathematical package to perform programmed floating point arithmetic, exponentiation, and data format conversion
- Interpreter program to operate upon ELPs
- Test module simulator.

MATE LANGUAGE PROCESSOR

Test procedures written by the engineers in the MATE language must be translated into machine instruction for execution. A detailed tradeoff analysis was conducted to resolve the question of off-line compiler application versus the on-line interpreter/compiler technique for the MATE System. The interpreter approach was chosen for the following reasons:

- Reasonable timing requirements by test equipment
- Engineers could work with the MATE language for programming, execution, and maintenance, without learning the computer machine instructions
- Due to limitation of external storage, there was no room to maintain both ELPs and software programs on the disk
- Quicker implementation of the total MATE System
- Ease of modification for extension of MATE language
- Ease of maintenance of operating system.

The MATE language processor is an interpreter. It consists of a resident and overlay programs. The resident part contains a floating point mathematics package, data format conversion programs (i.e., ASCII to binary, floating point to binary, etc.), and the general processor to identify, validate, and parse a source statement to be executed. When the ELP verb is identified, an appropriate overlay program will be invoked to interpret and execute the ELP statement.

Executive program

The executive program is the center of the MATE System. It is responsible for bringing together the operator, language processor, test modules, and test programs. The executive program is resident in core memory. The main components are:

- Interrupt processor
- Input/Output handler
- Table lookup routines
- Startup-shutdown routines
- Test program statement monitor.

The *interrupt processor* handles three types of interrupts: input-output alarm stop, and manual override stop. Input/Output interrupts are caused by operator action on the control panel, input data have arrived from test modules, or ready state reached by test modules.

When an alarm condition occurs on test modules, the alarm stop interrupt occurs. The computer stops and the system cannot be restarted until the alarm condition is reset. The manual override interrupt occurs whenever the test module, indicated on the control panel for manual intervention, is addressed. The computer remains in the wait state while the operator performs the necessary manual action on the control panel. The system is restarted when so directed by the operator.

The *Input/Output handler* performs all the Input/Output for the system. Devices handled are test modules control panel, display panel, teletype, and disk storage. In all cases, the calling program provides a parameter list which provides all the data to perform the physical input-output operation. There are two types of I/O operations for test modules and control panel. They are output pulse signal and data transfer. The output pulse signal is used to activate control panel lights and to describe the data about to be sent to test modules. Data transferred to test modules is in multiples of computer words and is either a device address or information word.

I/O for the disk storage and teletype is done by a specified number of words or to a control character terminating the record. For disk I/O, the physical track address must be specified. Since there is no wait state in the computer, the read from teletype instruction sequence is used to simulate a wait state when soliciting interrupts.

Due to the shortage of core storage, all tables are stored on the disk. These tables hold information relating to global and local symbols and their computed values, and calibrated standards referenced by test processes. Local symbols are symbols used only within a test sequence. Those symbols which are used among test sequences within test series are defined as global symbols. The local symbol table is reset at the beginning of a new test series.

The *test module simulator* is an important part of the

Executive Program. It permits the MATE System to be checked out without test modules. The teletype simulates the test modules. Inputs, described in octal code, are typed in by the operator and outputs are similarly printed. The simulator was invaluable in locating hardware-software problems and faults in test modules.

The *startup-shutdown routines* control initialization and termination procedures. At startup, the test series and test sequence numbers supplied by the operator, via the control panel or teletype, are validated. The appropriate ELP is located. Global and local tables are initialized. The language processor is called to begin interpreting the test ELP program. Shutdown simply involves readying the system for another test.

The *test program statement monitor* allows operator control of the current statement being executed.

Control panel options permit displaying the current statement number on the control panel, executing one statement at a time, repeating a given test sequence, restarting a test sequence from the last measurement read, or inhibiting entry into fault isolation program in case of errors. This monitor allows the operator to control the execution of a test statement at the source language level from the control panel and away from the computer.

Utility programs

An integral part of the MATE System are the utility programs. These programs consist of elementary routines such as core and disk dump programs, and comprehensive disk data handling programs.

Disk service programs maintain software programs and ELPs on disk storage. Further, they condense indices and data storage areas automatically when gaps develop as a result of deletions.

As both a startup and backup procedure, all data stored on the disk are stored on paper tape. There is an initialization program to set up the disk storage and a bootstrap program on paper tape to begin a test process.

CONCLUSION

The MATE automatic check-out system which has been described, has been in operation for more than 6 months. Its performance has given support for a bright future in computer-driven check-out systems.

The MATE System has demonstrated that:

- Test time can be reduced by a factor of 20 over manual methods
- It is economically and operationally feasible to use mini-computers in automatic check-out systems.
- Test engineer without previous programming experience can learn to write test programs in the specialized programming language, like the MATE language, in 2 days.

REFERENCES

A guide to atlas for test specification writers
ARINC Report 418 Aeronautical Radio Inc Annapolis Maryland May 15 1969

Coding techniques for failure recovery in a distributive modular memory organization

by S. A. SZYGENDA

Southern Methodist University
Dallas, Texas

and

M. J. FLYNN

The Johns Hopkins University
Baltimore, Maryland

INTRODUCTION

This paper considers various coding techniques which could be applied to a memory organization to achieve detection and correction of failures. In this discussion, we define a *fault* as a malfunction of a systems component and a *failure* as a manifestation of a fault. Notice that a single fault can result in multiple failures. Thus, techniques such as error-detecting and correction codes, when used alone, are limited in that they operate on failures—not faults.

Failure analysis of various memory organizations has been performed.[1] The material presented in this section of the paper summarizes these results. The analysis demonstrated that a fault in the accessing units, the address decoders or in the drivers, can introduce multiple failures in the memory. To insure against these types of faults we must prevent, or detect and correct the following conditions:

a. More than one of the decoder outputs becoming active during a memory instruction.
b. Major fluctuations occurring in the amplitude of the current pulse provided for the selection of memory words.

By digital simulation, it has been determined that, at most, two output leads from the memory drivers will be active at any one time, under a single fault assumption. Hence, a device capable of summing the current supplied by the drive lines can be used.

A simple parity check could perform this function; however, a better implementation would use a device capable of summing currents from a number of leads. Faults would be indicated by attainment of minimum or maximum threshold current values, i.e., this device would possess properties similar to that of a Zener diode.

By measuring the current, checking for two types of faults can be accomplished, first, detection of fluctuations of the amplitude of the current pulse on the correct line and, second, detection of no-drive-line or two-drive-lines being activated.

The current summing methods can detect the faults in the accessing units, but if processing is not halted while this check is made, data in the memory may be damaged. Since it is not desirable for processing time to be sacrificed for checking, except where absolutely necessary, it is assumed that data may have been erroneously changed in the memory due to checking being performed simultaneously with the memory operation.

Since it is necessary to locate the faulty word, the procedure of summing the current pulses can be used for that purpose. Under fault conditions, it is known from the current summing check that half of the address (more precisely, which half), has been decoded properly. Therefore, the faulty word must be one which is selected by coincident current pulse with the correct drive line.

Two questions remain to be answered:

1. How is the damaged memory word corrected?
2. How is the fault in the accessing hardware corrected?

It will be shown that correction of memory data can be accomplished if it is known that there will only be

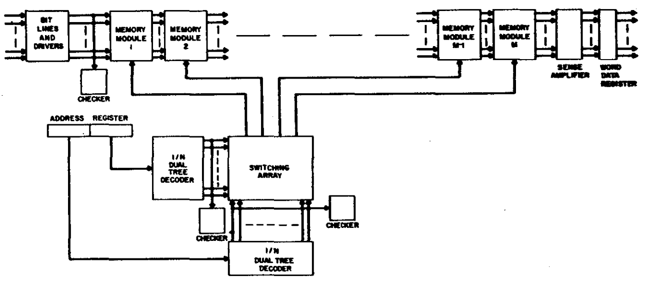


Figure 1—Decoder and memory module organization

one word damaged in a memory module. This correction will be accomplished by coding techniques to be discussed in the next section. Hence, the problem remaining is that of assuring that the type of faults being considered cannot affect more than one word in a memory module. This can be accomplished with the example organization shown in Figure 1. In general, the memory is divided into modules. For convenience, these modules may be logically divided into 2^N groups of 2^M words each. These groups will be referred to as the memory modules.

The switching array is made up of elements which are switched by half select currents applied by the horizontal and vertical drivers. The memory word desired is then indicated by the switched element of the array. For the read operation, a linear select current is applied to the lead emanating from the selected element. The bit lines act as sensing lines to sense the bits of the accessed word. The word is regenerated by having the bit line inhibit positions requiring "0." The word line is then driven to the "1" state.

A major feature of this organization is the manner in which the switching array is implemented. This implementation must meet the requirement of storing words in such a manner that at most one complete word in each module can be affected by a fault.

Figure 2 gives an example of this organization for a memory array of eight modules containing eight words each. As can be seen from the figure, lines $R_1, \ldots, R_8$ thread eight elements in a diagonal manner. (The symbology used on the R lines is for clarification only and has not been used on the L lines. The lines are all identical accessing lines.) This organization provides the desired failure distribution for single faults that could occur in the memory accessing units.

The question may arise as to the need for the current summing device if any single word in a memory module can be reconfigured. The reason for the summing is strictly for fault indication. That is, corrective action would not be taken unless a fault has been indicated.

Another possible area of faults would be on the bit lines. A fault affecting a bit line can in turn affect a bit in every word of the memory. However, it would only affect a single bit per word, and this condition can be corrected by techniques to be considered.

Faults in the accessing units can be eliminated by switching in a spare unit and then an indication of this condition will be given for replacement of the faulty unit.

As yet, we have not mentioned the possibility of transient faults in this organization. To insure against needless replacement of modules, because of transient faults, a retry feature will be used.

After a fault has been indicated and the memory words reconstructed, a retry order will be given. Under the retry condition, access to the switching array will be inhibited unless no fault is indicated. If a fault is again indicated, it will be assumed to be solid and the spare unit will be activated.

Two way read technique

An additional feature is to be provided for this memory organization. A two way read will be used on each memory module. That is, the capability will be provided to read an element in the same position of each word in a module.

This feature provides the capability of reconstructing a damaged word of data in the memory.

Figure 3 shows the basic organization with the two way read feature added. For reading, a linear select pulse will be provided on the bit line. This will switch all memory elements in the "1" state to the "0" state. The changes will be sensed by the word lines and stored in the bit register.

For rewriting, a half select pulse of opposite polarity

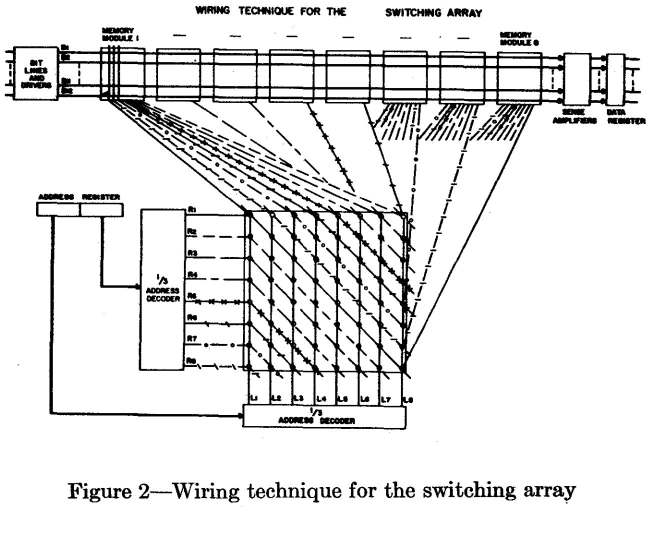


Figure 2—Wiring technique for the switching array

will be provided by the bit line and the word lines which sensed the original change.

By repeating the procedure of sequentially stepping through all the bit lines, we can read the contents of the memory module as rows of bits. This procedure will be essential to the discussion in the succeeding sections.

CODING SCHEME

Since all the memory modules are identical, we will restrict our attention to a typical module. The major techniques to be used are:

1. Virtual embedding of the address of the data word into the word itself (Figure 4).
2. Use of an iterated code for reconstruction of a memory word, or of a string of bits.

The scheme employed here uses virtual embedding of the address of the data word into the word itself, for the purposes of failure detection and correction. The embedding is virtual because the address of the data is never stored with the data. Rather, since its true value is presumably known during the storage phase, the coding for detection and correction treats its argument as the concatenation of the address and the data word itself.[2]

In conjunction with the vitrual embedding technique, an iterated code is used. A linear code is applied over the rows and over the columns of a memory module to produce the iterated code.

The use of these techniques will be demonstrated for the implementations adopted in this study.

Double error detection and single error correction

In this case, a Hamming minimum distance four code (applied over the address and data jointly) is

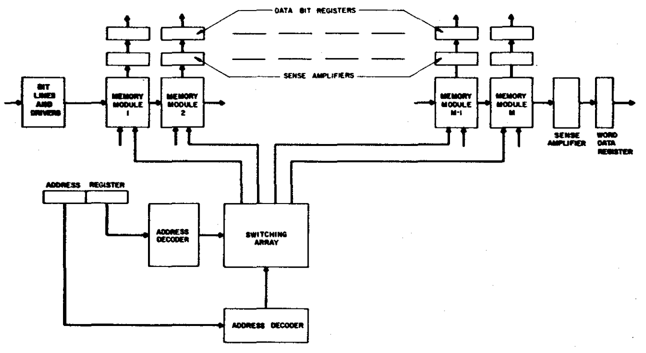


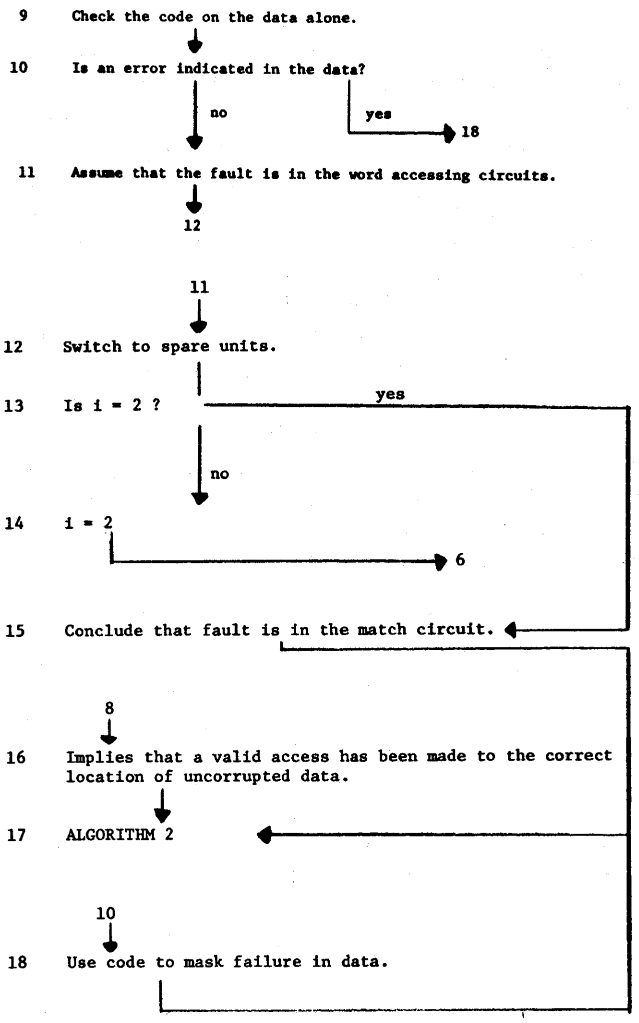
Figure 3—Memory organization with two way read

sufficient to obtain double error detection and single error correction.

With this system, the following can be accomplished. Assume an address has been moved to the address register (A.R.). The address is then decoded and a location is accessed. From the configuration of a typical system (Figure 5), we can determine whether the location accessed is the location that was specified.

When a word is moved to the A.R., the code bits are checked to determine if a fault exists in the A.R. or in the transfer. If a single failure has occurred, it will be detected and corrected by the code. Therefore, the validity of the contents of the A.R. can be assessed. When a word is accessed, a code will be generated jointly on the data portion of that word and on the address in the A.R. This code should match identically the code contained in the accessed location. If a match does not occur, there exists an error in the address, the data, or the match circuit. By decoding the accessed word, it can be determined if the error is in the data. If so, it can simply be corrected by the check circuitry. If the error is not indicated in the data, it is concluded that the fault is in the accessing circuitry. Correction of this fault would be by use of the spare accessing circuitry. Accessing of the word is then tried again. If, again, an error indication is given (not in the data), it is concluded that the fault is not in the address decoder but actually in the code matching circuit. The preceding discussion is formalized in Algorithm 1.

```
ALGORITHM 1:  MEMORY ACCESS WITH DOUBLE ERROR DETECTION
              AND SINGLE ERROR CORRECTION

1   i = 1.  Has a fault been indicated by the current summing circuit?
              ↓ No            Yes → 17
2   Move address, covered by code, to address register
              ↓
3   Validity of content of address register is checked by use
    of code bits.
              ↓
4   Has a single fault occurred?
              ↓ Yes           No → 6
5   Use code to mask single error, or indicate double error.
              ↓
      4    14
      ↓    ↓  ↓
6   Decode address and access designated word.
              ↓
7   Generate code bits on the concatenation of the data and
    address.
              ↓
8   Do these code bits match those contained in the accessed
    word?
              ↓ no            yes → 16
              ↓
```



```
9   Check the code on the data alone.
10  Is an error indicated in the data?   no → 11;  yes → 18
11  Assume that the fault is in the word accessing circuits.
    → 12

    11
12  Switch to spare units.
13  Is i = 2 ?   yes → 15;  no → 14
14  i = 2
    → 6
15  Conclude that fault is in the match circuit.   → 17

    8
16  Implies that a valid access has been made to the correct
    location of uncorrupted data.
17  ALGORITHM 2

    10
18  Use code to mask failure in data.   → 17
```

Iterated codes

An iterated, or product, code can be envisioned as one applied over the rows and columns of a binary array (Figure 6). The minimum weight of such a code is the product of the minimum weights of the individual codes. A proof of this theorem is given by Peterson.[3]

Two different product codes will be considered:

1. Minimum weight 2 code on columns and rows.
2. Minimum weight 4 code on columns and rows.

The minimum weight of the first product code would be 4. This code would be capable of double error detection and single error correction.

These codes will serve a double purpose in the memory organization proposed. They permit the reading of a memory word covered by a code or the reading of a string of bits covered by the code. Furthermore,

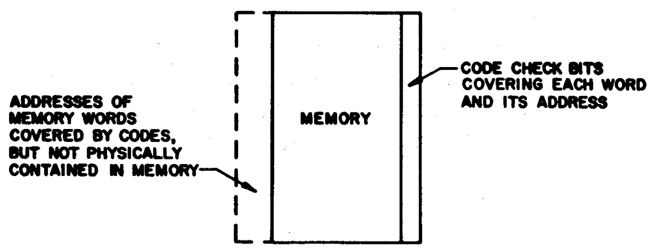

ADDRESSES OF MEMORY WORDS COVERED BY CODES, BUT NOT PHYSICALLY CONTAINED IN MEMORY
MEMORY
CODE CHECK BITS COVERING EACH WORD AND ITS ADDRESS

Figure 4—Virtual coding technique

these codes will permit the complete reconstruction of a memory word.

Reconstruction of a data word

The particular memory organization chosen in this study attempts to decrease the probability of multiple random type failures and increase the probability that multiple failures will be restricted to a single word in the memory module.

Under this condition, it is desirable to reconfigure a given word in a module. However, before failures can be corrected, they must be isolated. The detection is provided by two methods. First, there is the Hamming code to detect failures. However, multiple failures are not always detectable by a double error detection code. Therefore, some alternate method of detection must be provided. Since detection by the coding techniques would be the fastest, this will be attempted first. If no failures are indicated by the code, and a fault exists (knowledge of this fault would be provided by the current summing circuit), the bit read would be performed for the module, and the code which had been previously generated on the bits would be checked. If a failure exists in a word, it will be detected by this method. At

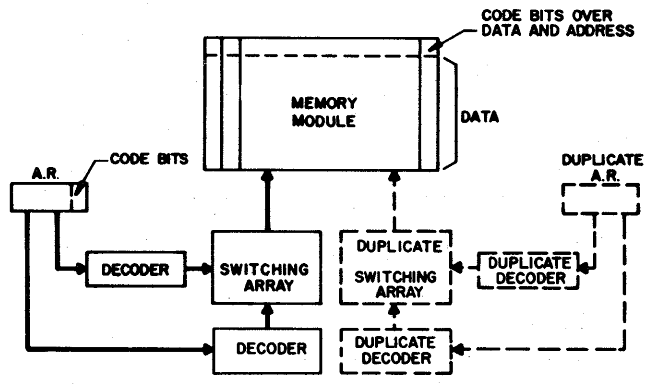


Figure 5—Memory configuration with virtual coding features

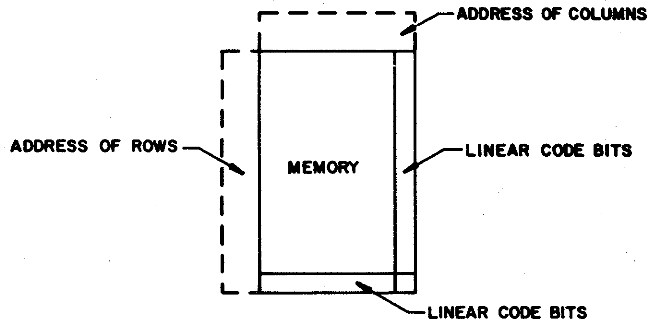


Figure 6—Iterated coding technique

this point, the failure is located in a given module of memory. Location of the failures to a particular word in that module is accomplished with the assistance of the summing circuit. Since a fault would have been indicated by this circuit, it is known that a failure for a given module will be at the intersection of the drivers specified. This will locate the word containing failures. Figure 7 illustrates a possible memory configuration utilizing these techniques.

For an example of this technique, consider (Figure 2) that drivers R_i and R_j are active and a fault is indicated by the checker. Also, driver L_k is active. The specified word of the module would be indicated by L_k and the modules being affected would be indicated by L_kR_i or L_kR_j.

This discussion has covered the types of solid faults detectable by the codes or the current summing check.

For a read only memory the use of a minimum weight four code may be acceptable. However, in a read write memory, where data is constantly being changed, it would not be practical to generate this type of code for the entire module each time a memory word is changed.

Although the use of a weight four code is impractical, virtual parity codes can be used efficiently. The encoding of a parity bit on a word of data is well known and will not be discussed here. However, it would be a costly procedure to recompute the parity on the bit strings by generating the parity after each access to the memory. Fortunately, the new parity on the bit string can be simply computed.

Consider a word (a_1, a_2, . . ., a_N) being entered into storage, and the storage word (s_1, s_2, . . ., s_N) which is to be replaced, also, the parity word (p_1, p_2, . . ., p_N) for the module.

The values of the parity word are assumed to be that of the memory module containing the storage word to be replaced. The following truth table provides the specification for the new values of the parity word as a function of the possible input combinations.

a_i	s_i	p_i	p_i^1
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	0
0	0	1	1
1	0	1	0
0	1	1	0
1	1	1	1

p_i^1 is the new required value of p_i. Hence, changes need only take place where there is a change between p_i and p_i^1 in the truth table.

The required function can be performed utilizing two exclusive-or operations. Verification of this statement can be shown by the following truth table.

a_i	s_i	$a_i \oplus s_i$	p_i	$p_i^1 = (a_i \oplus s_i) \oplus p_i$
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	0	0	0
0	0	0	1	1
1	0	1	1	0
0	1	1	1	0
1	1	0	1	1

Therefore, an exclusive-or can be performed between the new word being placed in memory and the word whose place is being taken. This would be followed by an exclusive-or with the parity word. The result would

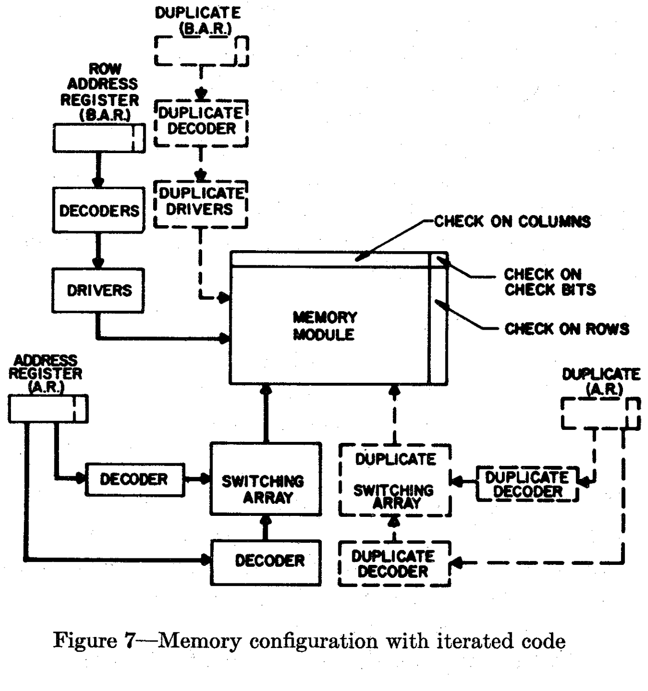


Figure 7—Memory configuration with iterated code

be the new parity word. This procedure would take place only on a WRITE operation. The complete method for reconstruction of a data word is formalized in Algorithm 2.

Next, we will consider faults in the checkers.

Checking the checkers

The question arises as to whether or not a malfunction can occur in the check circuitry, such that it could mask or introduce a fault into the data. The checking circuits of interest are the current summing circuits and the linear code decoders.

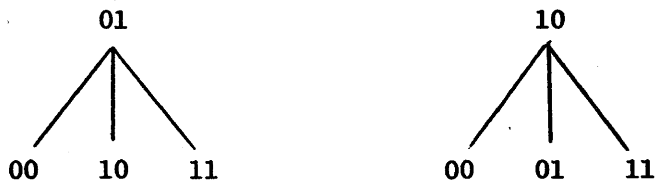
Faults occurring in the summing circuit could be of the type that a fault is indicated, but none actually exists. This case would be detected by the code circuit. If the summing circuit indicates a fault, the code circuit would attempt to locate the failures and correct them. If the fault was in the summing circuit, the code circuit would not be able to locate the failure. Hence, a fault indication is provided for the summing unit, and the faulty unit would be removed from the configuration. Whether or not a fault is in the code circuitry will be determined by a procedure described later. Another possibility is that a fault exists in a checker, such that it continually provides a fault free indication. One way to insure against this type of failure is to use a microprogrammed diagnostic sequence, containing failures to be detected by the checker, which will be executed periodically to verify the integrity of the checkers. Another alternative for this condition is to duplicate the check circuitry. Duplication is acceptable since this circuit is quite inexpensive. If a fault is indicated by either unit, the code check will determine if the fault is in the summing circuit. The precise unit can be deduced from the outcome of the code check.

A procedure will now be given to provide fault detection for the parity or Hamming decoding unit. For these units, duplication will again be employed. The inputs to the decoders will be identical. Assume that an output of 0 is the no fault indication and a 1 will be the fault indication. The possible outputs for decoders A and B would then be:

A	B	
0	0	No Fault
0	1	Decoder Fault
1	0	Decoder Fault
1	1	Fault, Not in the Decoder

If both units indicate a failure, it will be masked and processing will continue. If only one of the units indicate a failure, an attempt will be made at masking the supposed failure. After masking the data, it will again be processed through the decoders.

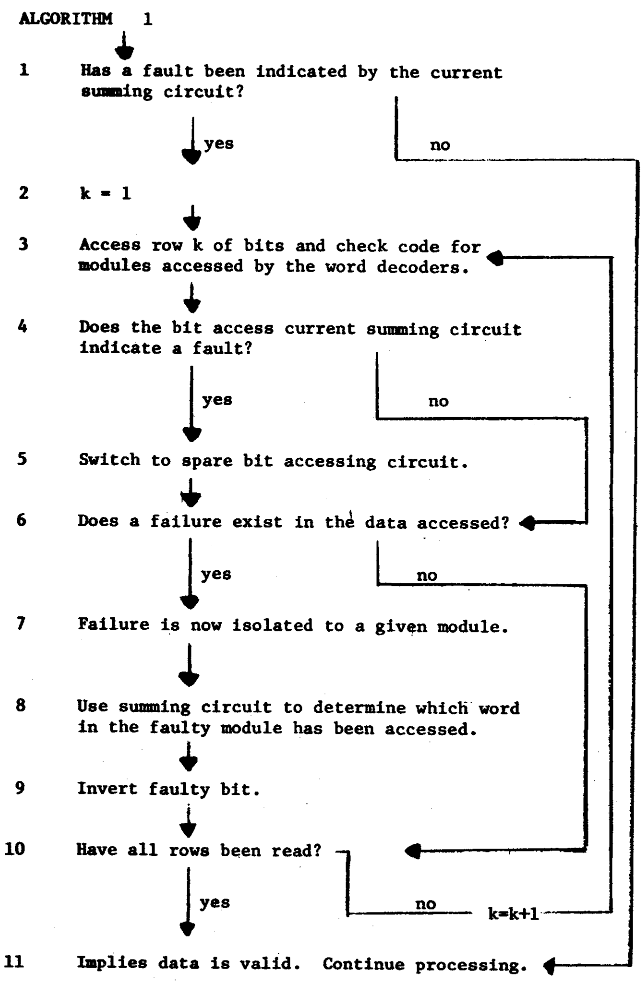
The following tree diagrams show, for both units, the possible output states that could be observed upon reprocessing.



Consider the outputs originally being 01. After masking and reprocessing, it could not again assume the 01 state due to the single fault assumption. If the outputs assumed the 00 state, the fault would be isolated to the A unit. The 10 state and the 11 state would indicate the B units.

If the original outputs were 10, the possible outputs could be: 00 indicating a fault in unit B, 01 indicating a fault in A, or 11 indicating a fault in A. These conditions are summarized below:

ALGORITHM 2: RECONSTRUCTION OF A DATA WORD



ALGORITHM 1

1 Has a fault been indicated by the current summing circuit? (yes → 2; no → 11)

2 k = 1

3 Access row k of bits and check code for modules accessed by the word decoders.

4 Does the bit access current summing circuit indicate a fault? (yes → 5; no → 6)

5 Switch to spare bit accessing circuit.

6 Does a failure exist in the data accessed? (yes → 7; no → 10)

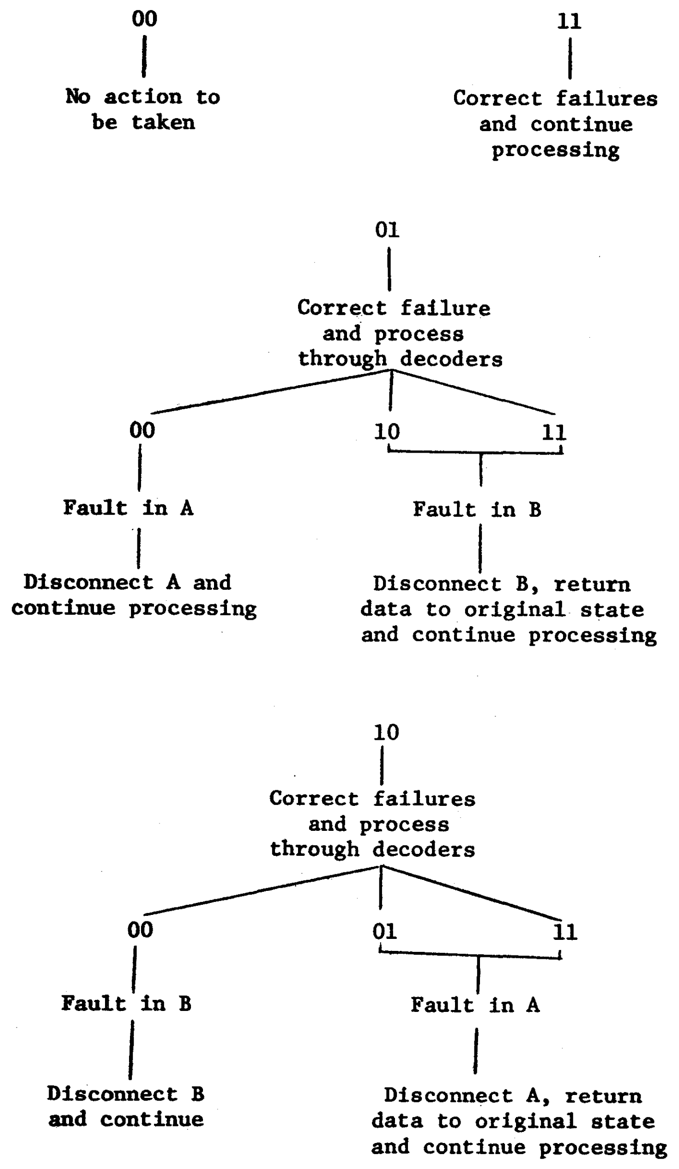
7 Failure is now isolated to a given module.

8 Use summing circuit to determine which word in the faulty module has been accessed.

9 Invert faulty bit.

10 Have all rows been read? (yes → 11; no: k=k+1 → 3)

11 Implies data is valid. Continue processing.


00
No action to be taken
11
Correct failures and continue processing
01
Correct failure and process through decoders
00
10
11
Fault in A
Fault in B
Disconnect A and continue processing
Disconnect B, return data to original state and continue processing
10
Correct failures and process through decoders
00
01
11
Fault in B
Fault in A
Disconnect B and continue
Disconnect A, return data to original state and continue processing

These techniques provide the capability of fault detection and isolation for the checking circuits.

Memory configuration

A major objective of this study is to reduce, as much as possible, the amount of redundant hardware used in the system. The configurations shown in Figures 5 and 7 have utilized duplication of the accessing units, resulting in an undesirable level of redundancy. Since the proposed memory would be modular or expandable in nature, a configuration was adopted which decreases the total amount of redundant hardware.

This configuration includes one or more read-only memory (R.O.M.) units, one or more read-write memory (R.W.M.) units, and one spare address register

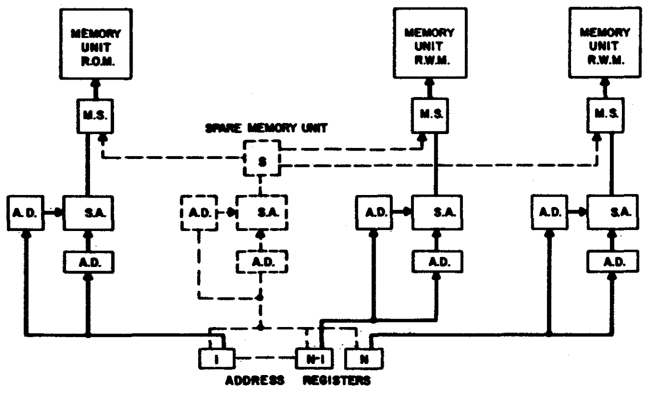


Figure 8—Memory configuration with redundant accessing

(A.R.), address decoder (A.D.) and switching array (S.A.), as shown in Figure 8. This approach eliminates the need for duplication of the accessing units for each R.O.M. or R.W.M. module.

Whenever a fault is indicated in any A.R., A.D. or S.A., the spare system is switched into use, the appropriate address is decoded and accessing of the word is accomplished by switching to the correct unit. To insure that the spare units are functioning, they are not left idle under fault free conditions. A counter is provided so that every nth access to a memory unit will be processed by the designated unit, as well as the spare. The spare contains the same checking features as the other units. Hence, corresponding faults will be indicated in a similar manner. To insure the functioning of the switches, the outputs of the spare and the designated units are compared. This process would be performed sequentially through each of the memory units, to check the functioning of the interconnections between any unit and the spare.

CONCLUSION

The techniques proposed in this paper provide the capability of failure recovery in the memory organization described. All failures resulting from single solid faults can be detected and corrected. In addition, fault detection in the accessing circuitry and the checkers is considered. A primary objective of this study has been to demonstrate that multiple failures could be corrected before they cause malfunctions, without resorting to duplication or triplication of hardware. A total memory system has been configured using the techniques described in this paper. This configuration uses less than 25 percent redundant hardware and no diagnostic software. The system is presently undergoing fault analysis by the use of digital simulation.[4]

While the codes considered have fulfilled the specified objective, it may be possible to use other codes resulting in a more efficient implementation. Two possibilities are presently being considered, the first is coding techniques utilizing more efficient decoding procedures or less redundant hardware. Second, the possibility of utilizing residue codes, such that common decoders could be utilized for codes applied to data undergoing arithmetic operations, as well as memory data, is being considered.

REFERENCES

1 S A SZYGENDA M J FLYNN
Failure analysis of memory organizations for utilization in a self repair memory system
IEEE Transactions on Reliability Feb 1971

2 R W DOWNING J S NOWAK L S THOMENOKSA
No 1 ESS maintenance plan
The Bell System Technical Journal Vol 43 Sept 1964 Part 1 of 2 pp 1961-2019

3 W W PETERSON
Error correcting codes
MIT Press 1961 pp 81-85

4 S A SZYGENDA D ROUSE E THOMPSON
A model and implementation of a universal time delay simulator for large digital nets
AFIPS Proceedings of the SJCC May 1970

Recovery through programming system/360—system/370

by DONALD L. DROULETTE

International Business Machines Corporation
Kingston, New York

INTRODUCTION

Recovery Management can be defined as the operational control of those system facilities (both program and machine) which strive to effectively deal with detected machine malfunctions within an operating system. Its primary concern is to maintain total system operation with minimum impact upon the availability of system resources.

Recovery Management, defined above and treated in this report, refers to recovery from an unscheduled system interruption resulting from a machine malfunction. As such, Recovery Management can be viewed as a consideration which leads to a higher degree of total system reliability, serviceability, and availability.

Effective Recovery Management is not a luxury; on the contrary, it may, in a given system, be a necessity. Without it, what need only be a minor problem becomes a major problem, possibly a catastrophe.

Recovery Management facilities service unscheduled system interruptions originating within an I/O device/unit, channel, processor storage unit, or central processing unit. The presence of such an interruption is indicated by a device/unit, channel, or machine-check condition. No individual Recovery Management facility services all machine malfunctions.

Recovery Management facilities attempt recovery at different levels; these levels differ with respect to the consequences imposed upon the system during the recovery process. Not all of the Recovery Management facilities have the capability of effecting recovery at each level. Some Recovery Management facilities are optional, and as such, must be specified by the user at system generation time. Considering that Recovery Management facilities are directed at specific types of failures, only after the thorough analysis of an installation's applications and requirements should a Recovery Management package be structured.

THE RECOVERY MANAGEMENT OBJECTIVE

The objective of Recovery Management is to provide the user with a higher degree of system availability (more time for more jobs) by minimizing the impact of machine malfunctions upon the user's operations. This objective is realized with the successful achievement of the following goals:

- Reduce the number of unscheduled system interruptions resulting from machine malfunctions.
- Minimize the impact of such interruptions in the event they do occur.

Through Programming, interruptions to the user can be reduced, their impact minimized and their causes isolated. There are a number of functions which can be performed to achieve these objectives of Recovery Management. Some of these are:

- *Instruction Retry*—The concept of instruction retry is not new. It is something IBM has been doing for years, particularly in the I/O area. Instruction retry has been standard procedure whenever an error was encountered in reading or writing a tape. It is possible to extend this retry capability and to employ it when a CPU or main storage malfunction occurs. A relatively large number of malfunctions are intermittent in nature, rather than being solid failures; therefore, there is a high probability of success of execution and recovery if an instruction retry can be attempted.
- *Refreshing Main Storage*—If instruction retry cannot be accomplished, one function which could be of value would be the ability to refresh main storage. Through this damage which either caused or was caused by a malfunction could be repaired.

This function could be accomplished by loading a new copy of the affected module or "Csect" into main storage or by a process known as check summing.

- *Selective Termination*—This function would enable the system to examine the failing environment, determine what problem program was executing and then proceed to terminate this program while entering all other jobs which were executing at the time of the malfunction. This is really a type of job which "frees" the resources of the system allocated to the job and makes them available for future use. This process results in the loss of a specific job but it keeps the system alive.
- *I/O Recovery*—The above functions have been directed mainly to errors which occur in the CPU or main storage. From an examination of system incidents, it is evident that a certain portion of errors occur in the I/O area. Recovery could be accomplished by I/O retry which is available through the error recovery procedures for the different I/O devices. Another group of I/O errors—channel control checks, channel data checks, and interface control checks—may be analyzed and under certain conditions a retry can be attempted. The I/O device or medium can malfunction and if retry is not successful the ability to switch data sets may be provided and then retry the operation on the new drive. Another is to try alternate routes to the same device, that is by addressing a device through a different channel or control unit.
- *Operator Awareness*—A group of system incidents is due to procedural and operator errors. Several things can be done to decrease these errors such as better trained personnel, minimal control information and clear and concise operator messages.

All of these functions are aimed at continuing the operation of the system. This is not always possible to accomplish. Therefore, the next best thing is to minimize the effect of the malfunction. This can be done by attempting to preserve information concerning the malfunction and to make it available to assist personnel to determine what caused the error and what can be done to correct it. Recording, therefore, is a major part of recovery management.

Recovery Management support has provided a number of these functions in the operating systems. RMS has provided a hierarchy of recovery which involves four levels of error recovery.

I Functional Recovery—Retry the interrupted operation
II System Recovery—Terminate the affected task
III System-Supported Restart—Prepare for Re-IPL
IV System Repair—Require stop for repair

Functional Recovery

Functional recovery is achieved when an interrupted operation is successfully retried. Such recovery is extremely desirable from a system point of view, because it makes the entire incident transparent to the user.

System Recovery

System recovery is achieved when system operation is maintained although an interrupted operation has not been successfully retried. This effort involves: an analysis of the failure's environment, a repair of the damage associated with the malfunction to prevent further interruptions, and/or an attempt to associate the malfunction with a particular task in order to allow selective termination of the affected job and continued processing of the unaffected jobs.

System-Supported Restart

System-supported restart is achieved when a stop for repair is not required and system operation is restarted using an Initial Program Load (IPL) procedure supported by System Restart facilities. (System Restart facilities aid the IPL procedure by preserving and using system job and data queues.)

System Repair

System repair, the lowest but most critical level of error recovery, consists of stopping the system and repairing a malfunction which cannot be serviced by the particular recovery facility at any of the previous levels. Recovery Management facilities aid maintenance personnel by providing them with detailed error analysis records. There is always, however, the possibility that system damage will be severe enough to preclude retrieval of the error records. In those cases, personnel will have to make use of the System/360 diagnostics available to them.

The levels of error recovery applicable to IBM Operating Systems operations are illustrated in Figure 1; the outcome of recovery procedures I, II, or III determines the level at which recovery will be effected. The bracketed information on a given flowline indicates the consequences of recovery at that level.

USER PERSONNEL INVOLVEMENT

The successful operation of a Recovery Management package is directly proportional to the planning for and use of specific facilities in a given operating system.

Once a user has determined what his needs and requirements are, the amount of specification required to tailor his Recovery Management package is minimal. The selection of some recovery facilities is made during the system generation process. Modifications can be made during the IPL/NIP process.

The programmer's responsibility varies greatly with respect to the Recovery Management options available to him:

- He may code actual error recovery routines which will receive control through macros specifying user exits (see *Optional User Written Routines*).
- He need not involve himself at all with regard to certain Recovery Management facilities.

Once the system has been set up and is running, it is the operator's responsibility to be aware of and responsive to the parameters required by, and the messages and wait state codes issued by particular Recovery Management facilities.

Maintenance personnel should acquaint themselves with the scope and operation of those Recovery Management facilities incorporated into the systems for which they have responsibility. They must be familiar with the messages and wait-state codes issued, and the error records produced, if they are to make effective use of the information available to them.

SUMMARY DESCRIPTION OF FACILITIES

This section briefly describes the available Recovery Management facilities. Included are discussions of the Machine-Check Handler (MCH), the Channel Check-Handler (CCH), and I/O Recovery Management Support (I/O RMS). The individual recovery facilities are discussed as they apply to specific types of failures, or to specific recovery functions. The topics of discussion are:

- I/O Device/Unit Recovery Facilities
- Channel Recovery Facilities
- I/O Recovery Management Facilities
- CPU/Processor Storage Recovery Facilities
- System Associated Recovery Facilities
- Error Record Retrieval Facilities

The following points are made to clarify the function

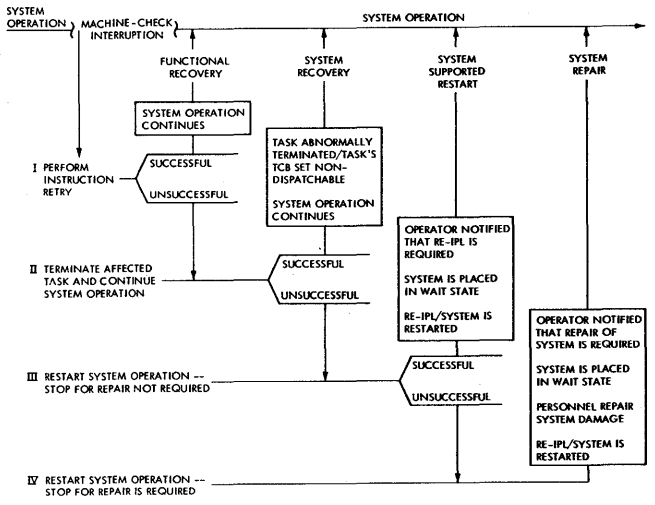

SYSTEM OPERATION
MACHINE-CHECK INTERRUPTION
SYSTEM OPERATION
FUNCTIONAL RECOVERY
SYSTEM RECOVERY
SYSTEM SUPPORTED RESTART
SYSTEM REPAIR
SYSTEM OPERATION CONTINUES
TASK ABNORMALLY TERMINATED/TASK'S TCB SET NON-DISPATCHABLE
SYSTEM OPERATION CONTINUES
I PERFORM INSTRUCTION RETRY
SUCCESSFUL
UNSUCCESSFUL
OPERATOR NOTIFIED THAT RE-IPL IS REQUIRED
SYSTEM IS PLACED IN WAIT STATE
RE-IPL/SYSTEM IS RESTARTED
II TERMINATE AFFECTED TASK AND CONTINUE SYSTEM OPERATION
SUCCESSFUL
UNSUCCESSFUL
OPERATOR NOTIFIED THAT REPAIR OF SYSTEM IS REQUIRED
SYSTEM IS PLACED IN WAIT STATE
PERSONNEL REPAIR SYSTEM DAMAGE
RE-IPL/SYSTEM IS RESTARTED
III RESTART SYSTEM OPERATION -- STOP FOR REPAIR NOT REQUIRED
SUCCESSFUL
UNSUCCESSFUL
IV RESTART SYSTEM OPERATION -- STOP FOR REPAIR IS REQUIRED

Figure 1—Levels of error recovery applicable to IBM operating systems

and scope of those recovery facilities which cross the bounds of two or more failure types:

- The *Optional User Routines* receive control from the IBM supplied Error Recovery Procedures (ERPS) on permanent I/O device/unit errors in order to determine whether their associated tasks are to be terminated.
- The *System Environment Recording Routines (SER0, SER1 and the Machine-Check Handler (MCH) program* can perform recording functions for channel and machine-check conditions. However, the limited SER1 and extensive MCH recovery capabilities deal only with machine-check conditions. Therefore, if one desires channel recovery, he must also make use of the *Channel-Check Handler (CCH)*. CCH may be used in conjunction with MCH, SER0, or SER1.
- The *System Environment Recording Editing and Printing (SEREP) program* may be used to record, edit, and print I/O device/unit, channel, CPU, and processor storage conditions. SEREP will be used when no automatic recording facility has been invoked, the facility invoked has failed in its operation, or the recorded records cannot be retrieved by the *Environment Record Editing and Printing (EREP) program*. EREP is a utility which edits and prints those error analysis records placed on the *SYS1.LOGREC* data set. This data set resides on the system residence device and is reserved for the exclusive use of all those recovery facilities which generate error analysis records.

I/O DEVICE/UNIT RECOVERY FACILITIES

The problem of malfunctions occurring within I/O device/units has been a concern for quite some time. The facilities available for the servicing and detection of these failures are:

- IBM Standard Error Recovery Procedures
- Optional User Written Routines
- On-Line Test System

IBM standard error recovery procedures

Standard error recovery procedures (ERPs) exist for I/O devices/units in order to maintain device performance and to provide uniform recovery procedures for all failures. The three types of IBM-supplied error routines are:

- Device-dependent routines
- Common routines
- I/O Recording routines

The device-dependent routines attempt functional recovery for particular device types by retrying operations a specific number of times. If functional recovery is not possible, control is passed to an optional user-written routine for further determination. Device-dependent routines exist for:

- Teleprocessing Devices
- Unit Record Devices
- Tape Devices
- Direct Access Devices
- Graphic Devices

The common routines are used by the device-dependent routines to analyze the type of error, to issue console messages, and to update the statistics table.

The I/O recording routines are the outboard recorder (OBR) and the statistical data recorder/channel-check recorder (SDR/CCR). OBR produces records for permanent I/O device failures on the SYS1.LOGREC data set. SDR/CCR updates the statistic counters on the SYS1.LOGREC data set whenever one of the error statistics counters in the statistics table overflows, and places I/O inboard records produced by the optional Channel-Check Handler (CCH) on the SYS1.LOGREC data set. The records placed on the SYS1.LOGREC aid maintenance personnel at the System Repair level.

Optional user-written routines

Should an installation determine that available Recovery Management facilities do not fill a need unique to the installation's requirements, user-written routines may be added to the system. When in the system, user-written routines are given control through the DCB macro instruction (SYNAD and EROPT). The user routine can determine on certain I/O device conditions if its associated task should be terminated.

On-line test system

The purpose of the On-Line Test System is to test the functioning of I/O devices in a controlled environment with minimum interference to the operating system. The On-Line Test System consists of an executive program, a series of tests for I/O devices/units, and a special SVC to perform functions required in the OS nucleus. The executive program serves as an interface between the operating system and the unit tests. It schedules and controls the running of the tests and provides communication with the operator. The use of the On-Line Test System serves to insure the integrity of the system's I/O devices. It might be considered preventive Recovery Management since its use should lead to the repair of faulty equipment prior to failure during system operation.

CHANNEL-CHECK HANDLER (CCH)

The Channel-Check Handler is designed to increase machine availability by minimizing the effects of channel malfunctions for 2860/2870/2880 and System/370 Model 155 channels. Without CCH, such malfunctions would be system incidents. The Channel-Check Handler will (1) determine the effect on the system of particular conditions that may have occurred, (2) set error indicators in the Error Recovery Procedure Interface Bytes (ERPIB) for the Error Recovery Procedure (ERP), and (3) create a record of the channel-error condition.

Unlike MCH, which is model dependent, CCH is only channel dependent because the Channel I/O Logout area is the analysis material used by the CCH program.

CCH includes the Dynamic Loading feature, which enables the main part of CCH (channel and model independent) to link to the various channel-dependent analysis routines. (See Figure 2.) Dynamic Loading also allows dynamic configuration for the specific channels on-line at NIP time, even if more channels were specified at SYSGEN time.

The Channel-Check Handler receives control from the I/O Supervisor (IOS) after detection of a channel control check, channel data check or an interface control check. CCH then completes its analysis to the error condition by setting up the ERPIB for the ERP or by indicating that immediate retry or termination is necessary. If termination is indicated, the error is recorded on the SYS1.LOGREC data set and a wait-state condition is set. If immediate retry is indicated, control is then returned to IOS who performs the retry and passes control to the next processing program on a successful retry. This retry is for special I/O operations such as SENSE. If an ERPIB has been created, IOS schedules the appropriate device ERP which operates in the Error Transient Area and receives a pointer to the ERPIB. (See Figure 3.) Based on the ERPIB information, the device ERP can determine whether a retry of the failing operation can be attempted or if the operation must be considered a permanent error.

For permanent error conditions, a message to the operator is printed (WTO Error MSG), the statistical data counters (STAT Update) for the devices are updated, a record of the permanent error condition is made on the SYS1.LOGREC data set by the Outboard Recording Routine (OBR), and an exit is taken. For errors marked as retryable, a retry is attempted and, if successful, control is passed to STAT Update to update the statistical data counters and then to OBR, which records the successful Channel-check recovery.

Functional Recovery is achieved on channel errors that can be successfully retried by CCH or the device ERP. CCH enhances the performance of OS/360 by reducing the number of system incidents resulting from channel malfunctions.

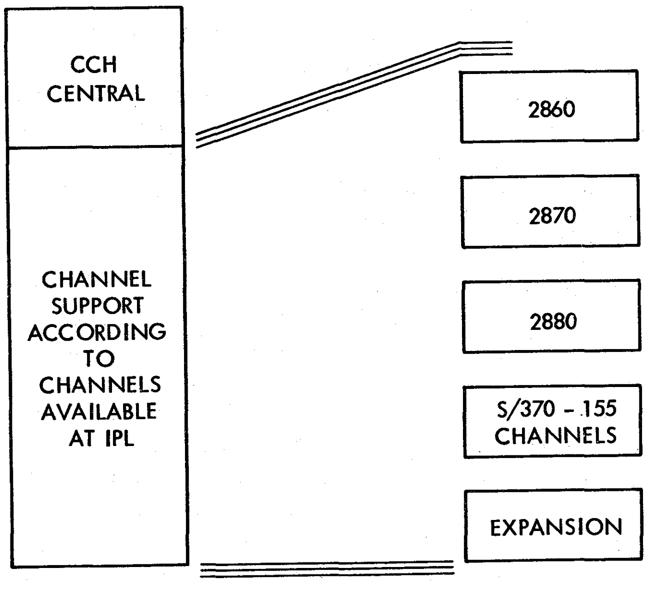


Figure 2—CCH dynamic loading

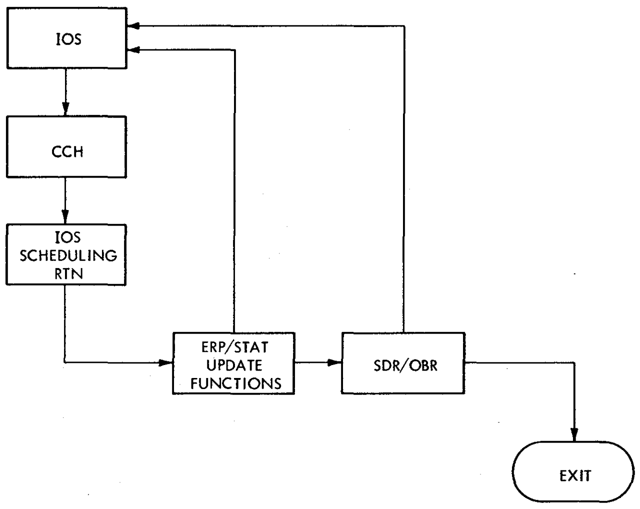


Figure 3—CCH processing

I/O RECOVERY MANAGEMENT SUPPORT

I/O Recovery Management Support (I/O RMS) is an extension to existing functions of the Operating System that address the availability and reliability needs of IBM customers that may not be realized due to channel, control unit, device, and medium failures.

Initially, these functions encompassed only the Device Dependent Error Recovery Procedures (ERP's), which were designed to effect a retry of a device failure on a particular path after a unit-check condition. Subsequently, with the implementation of the Channel-Check Handler (CCH), the utility of the ERP's was extended to effect a retry of channel failures (channel checks). In order to meet the continuing need for higher availability and reliability, I/O RMS provides two additional optional system functions that may be used to address the problem of I/O errors: Alternate Path Retry (APR) on the channel level and Dynamic Device Reconfiguration (DDR). (See Figure 4).

Without these functions, when an ERP is unable to successfully retry an I/O operation, permanent error is indicated. When a program encounters a permanent I/O error, it either accepts the error and continues, or ABENDS. If a critical supervisor function encounters a permanent I/O error, the system terminates.

APR

I/O RMS extends recovery from an I/O error with APR by ensuring that a different channel will be tried

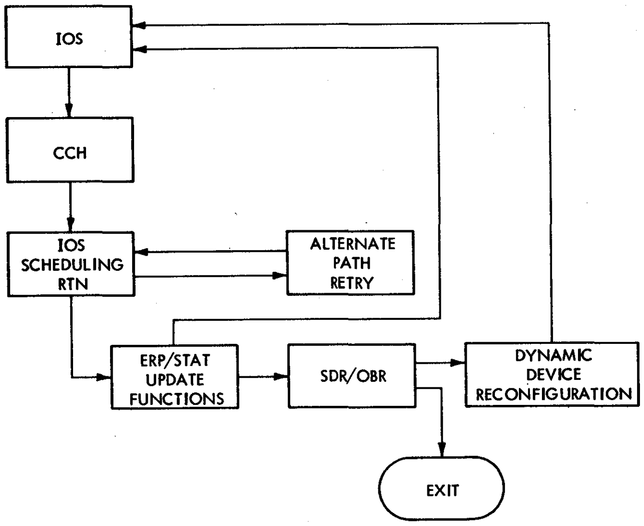


Figure 4—APR/DDR processing

(if one exists) during error recovery on a channel-detected error. If a permenent error exists on a device with a demountable volume, I/O RMS will extend recovery with DDR by requesting that the volume be moved to another device and the I/O operation retried.

The maximum number of paths supported to any one device will be four. APR will ensure that a different channel will be tried (if one exists, is on-line, and ready) only on retry of channel-detected errors. Retry on other errors will be handled as in the past. APR does not support tp.

In addition, APR provides an operator Command-VARY PATH. Through this command an operator can select a specific channel path and remove it from the system. Also, a path that has been removed can be put back on line through this command.

Alternate Path Retry is an extention of the Channel-Check Handler.

DDR

DDR extends I/O recovery when a permanent error develops on a device with a demountable volume by causing the system to request that the volume be moved.

The operator may also request DDR during normal execution to allow a volume to be moved from one device to another. A DDR can be operator-requested for volume cleaning, etc.

DDR can also be requested by the operator during "intervention required" conditions on readers, printers, and punches.

DDR will support the 2400 tape series, the 2420-7 tape, the 2311 and 2314 disks, the 2321 data cell drive, and readers, punches, and printers.

DDR can be requested by the operator anytime during execution, or by the system after a permanent error for all 2400 (including 2420-7), 2311, 2314, and 2321 devices. DDR can be requested only by the operator for readers, printers, and punches during "intervention required" conditions. "Intervention required" is either indicated by the system or may be caused by the operator. (The operator may cause an "intervention required" condition by making the unit "not ready.")

DDR's support of the 2314 allows the operator to move a volume to a drive on another 2314. It also allows the operator to move all data cells from the failing 2321 to another 2321. DDR will *not* allow the swapping of data cells on one device.

If the SYSRES option is selected, the SYSRES volume may be moved from one device to another at the request of the system or of the operator. The system will not request SYSRES swap unless a critical I/O operation is involved. (A critical I/O operation is one which involves the SVC library.)

If high availability is important to the installations, a duplicate SYSRES volume would be advisable. In order to use such a volume, writing on SYSRES would have to be prohibited except for the SYS1.LOGREC data set. Therefore, no libraries on SYSRES could be updated, no work data sets could be allocated on the SYSRES device, and SYS1.SYSJOBQE would have to be on a volume other than SYSRES. If the installation had such a duplicate volume, as well as an additional available SYSRES device, it would be possible to recover from both a device error and a media error.

SYSRES Option: Since some users do not have a demountable SYSERS device, DDR support SYSRES will be an option at SYSGEN time. THUS, the resident code necessary for SYSRES DDR is included only when the option is taken.

Dynamic Device Reconfiguration is an extention of IOS as it applies as much to device errors as channel errors.

With I/O RMS, a device encountering an error channel prone path may be able to continue operating on a different channel path. A volume on an error-prone device may be used effectively on a different device. Specifically, bus-out checks, and data checks, along with other error types, will have a higher degree of recovery, since a path to the volume may be made available that excludes the source of error.

I/O RMS is not model dependent.

In summary, I/O RMS will extend device performance in areas that may have previously rendered a job or the system inoperative.

CPU/PROCESSOR STORAGE RECOVERY FACILITIES

Machine-check conditions which arise within the CPU or processor storage are serviced by the mutually exclusive recovery facilities MCH, SERO, and SER1. If none of these are chosen at SYSGEN time, the default condition is a wait state. That is, when a machine check is encountered, the machine goes into a wait state. If such a wait state condition occurs or should a facility fail in its recovery attempt, SEREP may be used to access the CPU logout. (MCH is mandatory in the System/360 Model 85 and System/370 Models 155 and 165.)

MACHINE-CHECK HANDLER (MCH)

The primary function of the Machine-Check Handler is to attempt recovery from main storage or CPU failures which ECC or HIR has not previously corrected. An important additional function is to record each failure. The goal of MCH is total recovery, achieved when the interrupted program is enabled to continue processing at the point where the interruption occurred. When total recovery is not possible, MCH attempts to terminate the effected task without halting the entire system. If, however, a stop in system processing cannot be avoided, the error records produced by MCH aid manual repair.

MCH processing is inseparable from the operations of the machine recovery facilities, ECC and HIR. Upon detection of a hardware failure, either ECC or HIR (depending on the type of error) receives control. Only after these circuits make their recovery attempt does a machine-check interruption occur. MCH receives control at the interruption by means of the machine-check new PSW which contains the address of the MCH Resident Nucleus. Figure 5 illustrates the sequence of operations performed by MCH.

The path followed by MCH processing depends on whether or not the machine facilities were successful in their recovery attempt. If so, MCH only records the error, after which control is returned to the system. If the recovery attempt was unsuccessful, MCH analyzes the error and attempts recovery. If recovery is achieved, MCH records the error, notifies the operator, and returns control to the system. However, should recovery not be effected, MCH attempts to record the error, informs the operator of the condition of the system, then enters the disabled-wait state.

NOTE: In System/360 Model 65, Instruction retry and single bit error correction are performed by the program.

System environment recording (SER0 and SER1)

These optional recovery facilities record machine malfunctions of the CPU, processor storage, and channels in System/360 Models 40, 50, 65, 75, and 91 (SER1 only). After an error record has been placed on the SYS1.LOGREC, the system is placed in the wait state. If system repair is not required, a message is issued to the operator requesting him to re-IPL (System-Supported Restart). In addition to the recording function, SER1 attempts to associate the failure with a specific task. If the failure affects only the job step associated with the current task, the job step can be terminated without requiring a complete stop of the system (System Recovery).

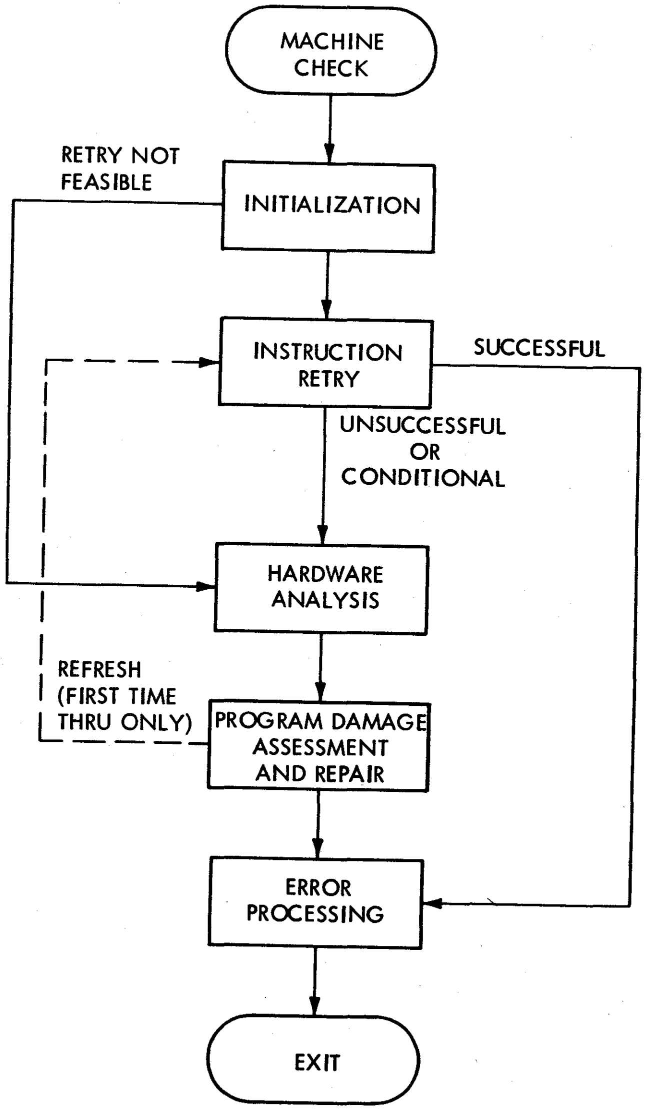


Figure 5—MCH gross flow

SYSTEM ASSOCIATED RECOVERY FACILITIES

While the following facilities do not actually record or analyze errors, they are an integral part of the Recovery Management scheme in that they further reduce the time involved in recovering from a malfunction which has caused an interruption in system operation:

- System Restart
- Checkpoint/Restart

System restart

The system restart facilities aid the IPL procedures by allowing the system to resume operation without having to reenter jobs that have been enqueued. This is especially time-saving in the case of those malfunctions which require a halt of system operation without a stop for repair. Information concerning input work queues, output work queues, and jobs in interpretation, execution, or termination is preserved for use when the system is reloaded. When the system is restarted, a message is written to the operator describing the status of each job in the system.

Checkpoint/restart

The checkpoint/restart facility provides the capability of restarting program processing subsequent to an I/O device/unit error, machine check, channel check, intentional operator intervention, or similar event. Job step information is recorded at user designated checkpoints in a problem program; if restart becomes necessary, it can be initiated from an available checkpoint. Checkpoint/restart can be invoked subsequent to system restart or subsequent to the abnormal termination of an effected job by one of the recovery facilities.

Use of this facility minimizes time lost in reprocessing a job step that has been terminated. It is used to best advantage in programs of long duration, or with programs where restarting from the beginning would be difficult.

ERROR RECORD RETRIEVAL FACILITIES

Although automatic recovery procedures are extremely desirable, such recovery is sometimes impossible, and human intervention on the part of maintenance personnel is required. The following facilities are part of the Recovery Management scheme, in that they facilitate system repair by providing a means of accessing failure data:

- Environment Record Editing and Printing (EREP) utility
- System Environment Recording Editing and Printing (SEREP) program

Environment record editing and printing utility

EREP, running under the operating system, edits and prints error records generated by OBR, SDR/CCR, CCH, SER0, SER1, and MCH and recorded on the SYS1.LOGREC data set.

The EREP utility program can edit and print:

- Combinations of the above records
- Records that were generated within a specific period of calendar time
- I/O outboard or statistical count records, or both, related to a specific channel or unit
- I/O outboard or statistical count records, or both, related to a specific I/O device type

EREP normally clears each selected record to zeros in the SYS1.LOGREC data set when processing of that record is complete. However, an option can be specified to prevent the clearing of selected records. Thus, a log of specific error conditions can be retained in the data set.

EREP output provides information for interpretation by the people performing the repair function.

A standard operating procedure in a Computer Center using MCH and/or CCH should be to execute EREP on a regular basis and then the information would be available to repair personnel as an aid or indicator to anticipate serious trouble. Upon review, if a particular pattern appears—indicating possible degradation, preventative maintenance may be performed before the occurrence of a serious incident.

System environment recording, editing and printing program

SEREP, is used to access failure information when:

- No automatic error recording facility (SER0, SER1, CCH, MCH, OBR, SDR/CCR) has been invoked
- An automatic error recording facility has failed in the performance of its function
- The SYS1.LOGREC data set cannot be accessed to obtain the error analysis records

SEREP is manually loaded using the standard IPL procedure. The program prints the information regarding the failure's environment on an online printing device. The SEREP procedure is aimed at improving the overall performance by minimizing unscheduled downtime. The program allows maintenance personnel to take full advantage of the machine diagnostic capabilities of the system in analyzing and correcting the following types of machine malfunctions:

- I/O Channel Failure
- I/O Device Failure
- I/O Test Channel Failure
- I/O Device Not Operational
- Machine Check Failure

RMS/65 RELATIONSHIP TO THE OPERATING SYSTEM

The RMS/65 package is comprised of two components, the Machine Check Handler (MCH) and the Channel Check Handler (CCH). For System/360 Model 65, both components are optional and a user at SYSGEN time may choose (1) CCH only, (2) MCH only, or (3) both MCH and CCH, depending on the needs of the installation. For System/360 Model 85 and System/370 Models 155 and 165, the MCH and CCH are an integral part of the Control System and, therefore, are not an option.

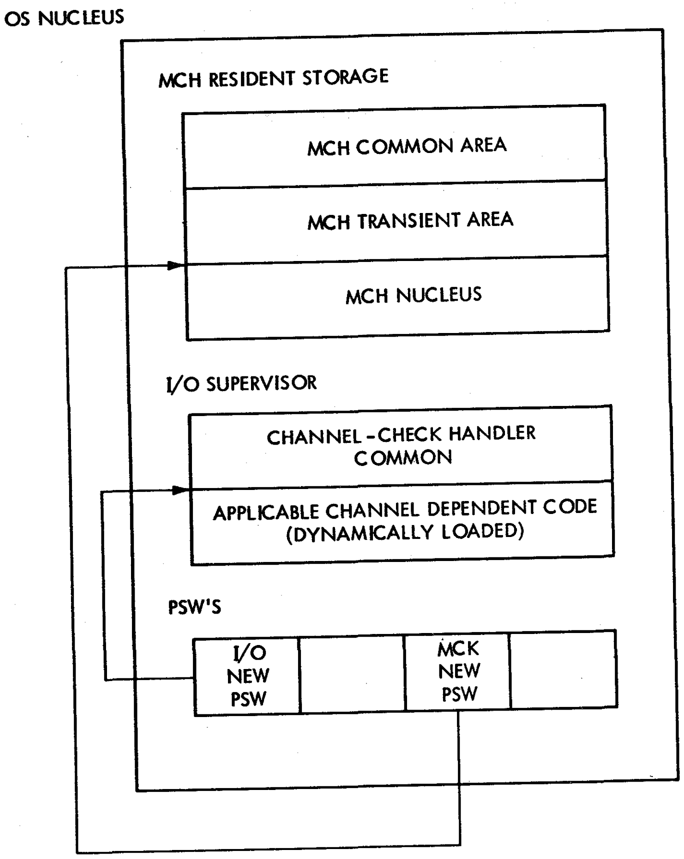


Figure 6—RMS relationship to OS

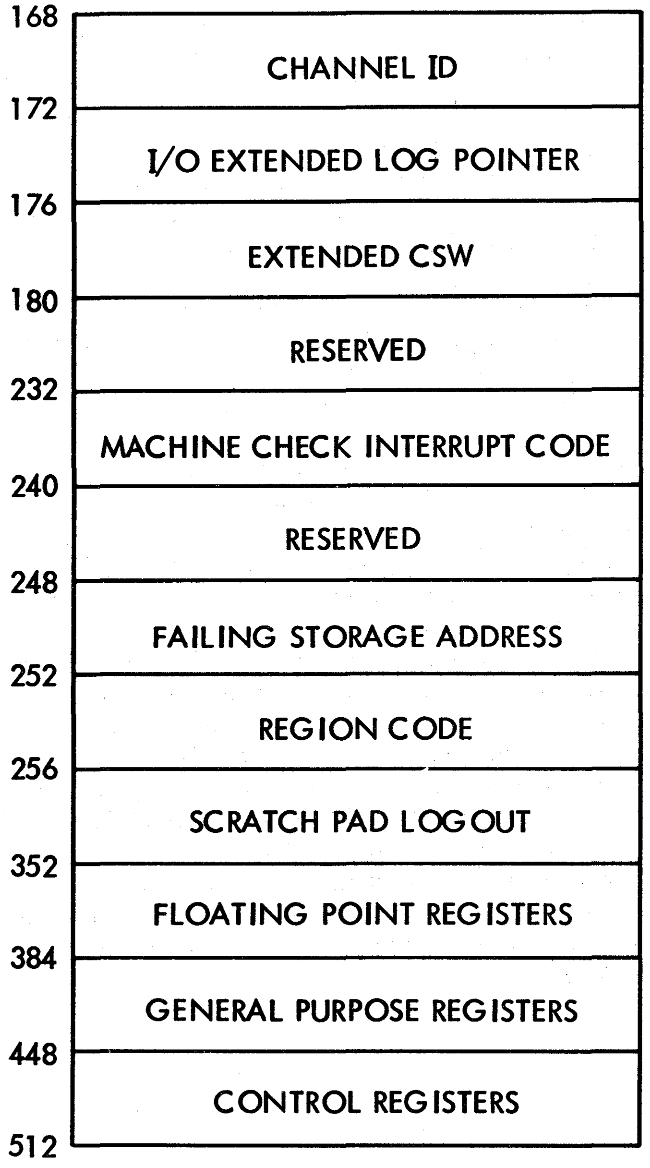


Figure 7—Permanently allocated storage locations

When selected at SYSGEN time, the components of RMS are included as part of the resident OS Nucleus. See Figure 6.

SYSTEM/370 CONSIDERATIONS

The current program status word (PSW) bit 13 has taken on more significance in System/370. In System/

360, bit 13 had sole control of Recovery Management functions. In System/370 there are recovery submasks in the control registers area which function in conjunction with bit 13 of the current PSW. Therefore, if bit 13 of the PSW is one submasks and the subclass mask bit in the control register is another, the associated condition will initiate a machine-check interruption. If either bit is zero, an interruption would not be initiated. Some subclass condition masks are system damage, timer damage, system recovery, etc.

Permanently allocated storage locations have been extended in System/370 for machine-check handling. Storage locations 168 thru 512 contain the added information for handling machine checks. (See Figure 7.) This information is supplied to assist in performing the recovery function. Such information consists of Channel ID, I/O extended by log-out pointer, limited channel log-out, I/O address, machine-check interruption code (discussed below), failing storage address, floating point, general and control registers as well as model dependent areas.

The Machine-Check Interruption Code is a double word starting at location 232. It contains such information as the time of interruption occurrence, machine-check intended log-out length, and subclasses. A subclass identifies the machine-check condition which caused the interruption. Some subclass conditions that can be indicated are system damage, instruction processing damage, timer damage, external damage, automatic configuration (when performed by hardware) and storage error type (whether corrected or uncorrected).

CONCLUSION

I believe that effective error recovery is a partnership between engineering and programming and these two must form a partnership and attack the problem together in order to provide a satisfactory solution. Recovery Management Support is a step in the direction which Error Recovery must take if the requirements of computer technology are to be met in this area. Every sign indicates that this is being accomplished.

It appears that some meaningful steps are being taken toward the goal of reducing the number of interruptions to which a user is exposed and to minimizing the impact of these interruptions when they do occur.

REFERENCE MATERIALS

IBM System/360 Operating System—System Reference Library

Concepts and facilities	*GC28-6535*
Operator's reference	*GC28-6691*
MFT guide	*GC27-6939*
MVT guide	*GC28-6720*

IBM System/360 Operating System—Program Logic Manuals

I/O supervisor	*GY28-6616*
MVT job management	*GY28-6660*
MCH for model 65	*GY27-7155*
MCH for model 85	*GY27-7184*

IBM System 360 Operating System

Machine check handler for the IBM System/370 Models 155 and 165, systems logic *GY27-7198*

On automatic testing of on line real time systems

by JON S. GOULD

Computer Consultants Corporation
Denville, New Jersey

INTRODUCTION

One of the major problems confronting the development of on-line, real-time information systems is overall system debugging—the final testing of all the integrated pieces working in concert under load conditions. An indicative analogy can be drawn from a look at similar project schedules (Figure 1), for systems of equal magnitude when one system is a conventional development and the other is on-line, real-time system (OLRT), our analysis will reveal the following:

1. Both systems require essentially the same amount of time and effort in the first three (3) phases.
2. During unit testing they are also quite similar with OLRT leading slightly.
3. In the final phases of integration, acceptance and cutover testing, the OLRT system extends several lengths over a conventional system.

The problem is quite simple. Although many would argue that OLRT systems are more intricate and difficult, I think most experts would agree that the larger non-OLRT systems are every bit as intricate and difficult and in many cases, perhaps more so. The problem, reduced to its simplest terms, is:

THERE ARE ESSENTIALLY NO
COMMERCIALLY AVAILABLE
OLRT DEBUGGING TOOLS!

There is essentially nothing available to the OLRT developer to help him debug the real-time aspects of a system, loading effects, testing of time critical events, throughput, and the list goes on. By-and-large, the extensive facilities available through various commercial operating systems in the form of dumps, traps and traces are of little or no value to the OLRT system during the final debugging stages of a system.

Therefore, the OLRT system developer is left to fend for himself, to develop whatever aids can be done within the scope of available resources. This effort, if it is undertaken at all, generally amounts to some weak attempts to build simple trap and trace subroutines, and perhaps some appropriately placed 'core dump' routines. This is integrated into the actual OLRT system and tailored specifically to this task and its associated programs, to the extent of rendering the routines useless to any other similar subsequent systems development effort. Therefore, when this project is completed, this 'test' code will be discarded or inactivated and probably never reused. Since test code is almost always required and never planned, it contributes greatly to project over-runs and in some cases system failure.

Let us look at the two types of projects (conventional and OLRT) from a testing and integrating viewpoint. A large scale conventional 'batch' processing system will involve several data files spread over disks and tapes, and will probably involve a variety of SORTS and MERGE runs, sprinkled liberally with processing programs. To test this system we must make up several dummy data files, or perhaps live data if it is available, layout a set of transactions which will test the sundry error conditions, and then run the system program-by-program, printing and analyzing the intermediate data and structures file as we go. Everything has been done neatly and locked-in to the processing unit in the computer room. Each phase or sub-task of the test is under the direct control of the programmer or operator. Contrast this with the OLRT system: To test a system, which may look like those in Figure 2, we must now arrange to have operations personnel standing by terminals in various parts of the country to enter certain specified data, in a specified sequence, at our command (We have sent them the instructions several days ago and after many phone calls they now understand what they have to do.) We must establish communications, have the data entered, and after the test have the corresponding results from the remote terminals sent in for analysis. Obviously, we have very little control over this type of testing. As the size of the system increases, the debugging problem grows exponentially. In summary, OLRT systems development is generally constrained and may fail because of:

1. A lack of sophisticated debugging aids.
2. System load testing to an over-capacity state is generally not attempted until cutover.
3. The test tools that are developed are custom tailored to an application and virtually useless for subsequent reuse.
4. Time compression capabilities to create real-time situations are not usually available.

It is desirable to have a method of testing, debugging, and analyzing an OLRT system that may be used over and over without extensive modification. A system should contain the following attributes:

1. It must be an easy-to-use debugging aid.
2. It must be reusable, which means it must be generally transparent to the application undergoing the test.
3. It must be able to subject the OLRT to data loads which are at or near its capacity.

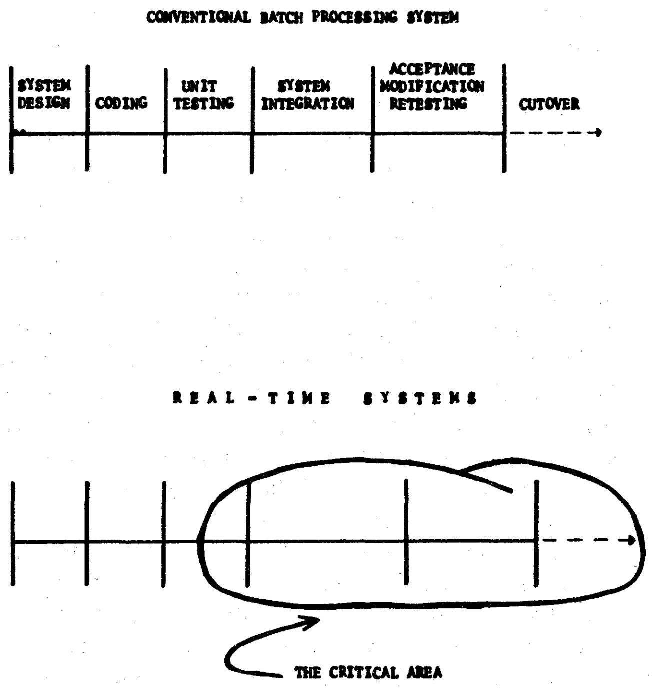


Figure 1—Project schedules

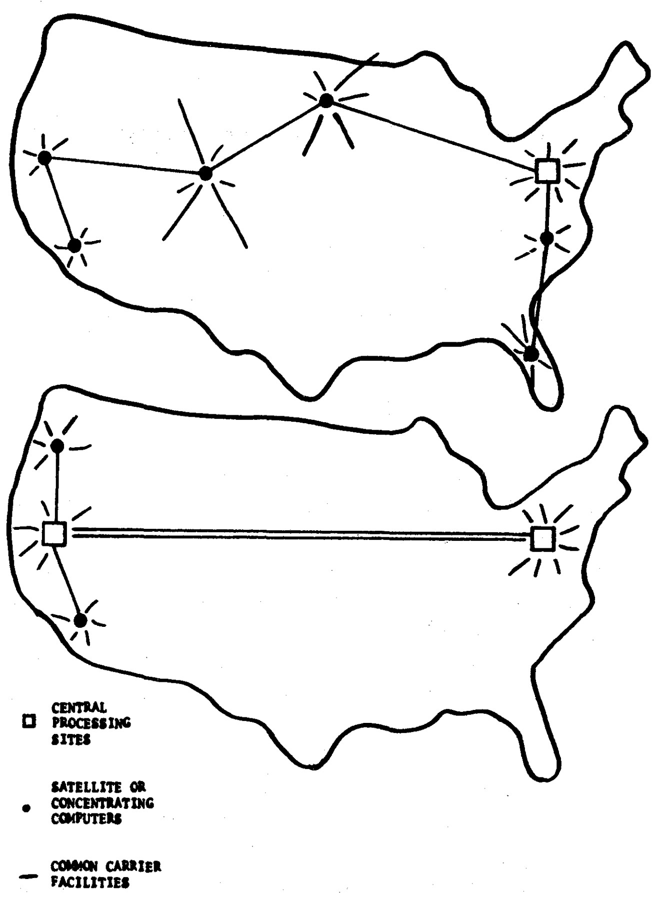

CENTRAL PROCESSING SITES
SATELLITE OR CONCENTRATING COMPUTERS
COMMON CARRIER FACILITIES

Figure 2—Communications networks

4. It must be readily modified to insert special system functions.
5. It must be capable of compressing time—to test now, those events scheduled several months or years away.
6. It must be useful for debugging modifications and new applications after cutover, as well as during the initial development effort.
7. It must be available at the right time in the development schedule.
8. It should provide a useful training aid for maintenance programmers, operators, etc.
9. It must be 'external' to the OLRT system so that its influence or overhead is not additive to system occupancy.
10. OLRT should not require any special coding whatsoever to use the test vehicle.

This is a tall order, to say the least. However, the application description which follows provides these things and more. This is primarily because the development of this system addressed itself to this problem head-on, it is not a by-product of an on-line system.

The technique itself is straightforward and simple; we assume that to provide meaningful tests of an OLRT system, especially during the integration phases of the system, the test vehicle should not reside in the same hardware system as the OLRT system. The OLRT system must be exercised in precisely the same manner as the 'real' network, including interrupt loads, etc. The primary differences between the technique described here and most others is:

1. THE TEST VEHICLE DOES NOT RESIDE IN THE ON-LINE SYSTEM.
2. THE TEST VEHICLE DOES NOT REQUIRE ANY MODIFICATIONS TO THE OLRT SYSTEM.

EQUIPMENT

The test vehicle, hereafter referred to as ATOLS, resides in a completely independent computer complex, which may be the back-up hardware for the OLRT system, or it may be contained in one of the popular mini-computers, which could be used to provide this function solely.

Regardless of the method selected, since the systems are independent, there is no requirement for any special engineering to effect the interface hook-up. Some extra common carrier supplied hardware, such as patch panels for communications lines and line 'battery' supplies may be required. However, this is standard off-the-shelf equipment of which most communications systems have an abundance. The application is described here, using the back-up computer system, since it was a completely redundant installation.

The communications carrier provides the equipment installation of phone lines and terminal board hardware which is commonly referred to as their DEMARKATION POINT or DEMARKATION STRIP. The common carrier terminates his equipment on the strip and provides a corresponding termination point for the attachment of 'foreign' equipment, such as the computer system. This is the termination point for the carrier's maintenance responsibility. The computer hardware vendor wires to the corresponding termination and the lines are connected and ready for on-line operations.

ATOLS is also terminated on the DEMARKATION STRIP in a similar manner as the on-line connections. The effect of this connection is to have both hardware systems 'split-wired' to the telephone lines. Figure 3

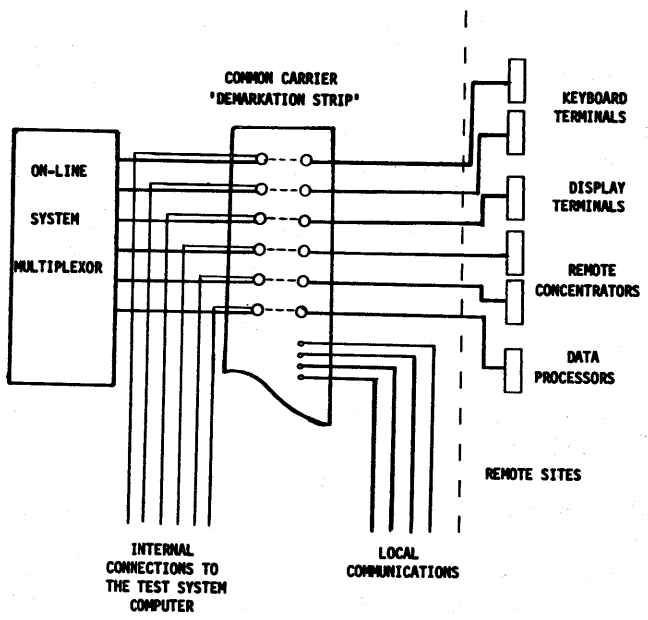

COMMON CARRIER 'DEMARKATION STRIP'
ON-LINE SYSTEM MULTIPLEXOR
KEYBOARD TERMINALS
DISPLAY TERMINALS
REMOTE CONCENTRATORS
DATA PROCESSORS
REMOTE SITES
INTERNAL CONNECTIONS TO THE TEST SYSTEM COMPUTER
LOCAL COMMUNICATIONS

Figure 3—The test system installation

illustrates a typical connection at the DEMARKATION STRIP. It is important to note that the installation of the common carrier's equipment is not necessary to provide a simple ATOLS-to-OLRTS interface. If there is a delay in the common carriers installation, the machines can be temporarily wired together on a limited number of lines.

After these line connections, we are ready for testing. ATOLS is a fully implemented OLRT system with some special features added for the test environment. It has both on-line and off-line components for test data generation and data analysis, as well as the on-line operation.

THE ATOLS OFF-LINE SYSTEM COMPONENT

The off-line system provides for the generation of test data to provide the inputs for a test run, and the analysis of data that has been produced as outputs from the test run, after transmission through the OLRT system.

TEST DATA PREPARATION

One of the obvious problems of working with one computer driving another is to generate enough test data in the test vehicle to be able to sustain a volume level to be transmitted to the on-line system over a period of

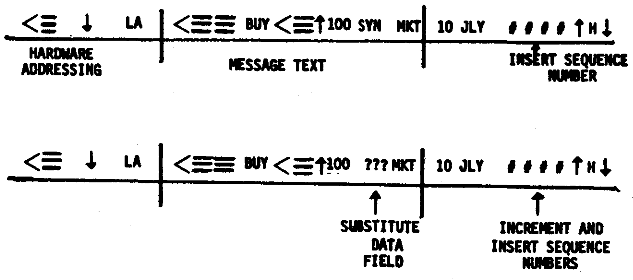


Figure 4—Test message data

time. Running at full speed, a system with several hundred lines would process several thousands of messages in an hour. Therefore, the system must have the capability to generate massive amounts of input information.

This is accomplished by a set of data preparation programs which accept punched card images and generates magnetic tape with the finished message formats and their destinations coded into the tape records. Data preparation is essentially free-form with the exception of a few control parameters, such as the number of messages or copies of the message to be generated and the hardware addressing characters, if any. Several fields are permitted to be imbedded within the data for the data generator program to update, such as sequence numbers, account numbers, and items of this sort that would be incremented or updated on message-by-message basis.

As the tape is generated, a copy of each message can be printed on the high-speed printer for verification of the final data format. The data generator function runs in the background of the on-line system and is able to produce test data while the switching system is in operation. However, it is generally done off-line so that the data can be verified before transmission to the OLRT system.

The recommended method of data preparation is off-line. The analyst determines the type of tests to be run in advance, codes the data, and it is keypunched and subsequently generated by the system. There are certain types of inputs that you would like to enter into the system while testing is in progress, such as special message types and console orders. For these inputs, a standard input device is available (the computer console, teletype, etc.) to enter traffic directly into the on-line system for transmission immediate action. Therefore, the data need not be completely prepared off-line, and may be entered at will from a supervisory station.

Figure 4 is an illustration of several input message formats that the data preparation programs would accept, expand, and generate to a magnetic tape file.

After the tape has been generated and the data content has been verified on the high-speed printer, ATOLS is ready to bring this tape into the system to queue the messages to the appropriate destinations. To initiate this process, the computer operator enters several commands through the operators console to 'close' the file and to 'load' it in for data transmission. At this point, the on-line data preparation programs take over, read the tape in, and queue the information to a disk or drum in preparation for transmission. For descriptive purposes, the generation and transmission have been separated; actually, these functions may be going on simultaneously.

When the data is loaded and queued to the lines, the system notifies the operator of the number of messages queued, queue reports to appear, etc. The complex is ready for on-line testing of the real-time system. At this point, the OLRT system is loaded in the on-line computer complex, and is awaiting input data or initiating some network action.

DATA ANALYSIS

The current implementation has several basic functions. First, the inputs to the data analysis package are the ATOLS journal tapes from the on-line run and the initial input data tape that contains the initial test data. These tapes are processed, merged, and sorted together to produce a single tape file containing the chronological events of the test. This file is input to a series of COBOL and FORTRAN data processing programs to reduce the data and to check its content. Some of the items that are checked are:

1. Message routing in a switching application is checked. The message sent by the test vehicle is matched against the message received to verify that it was transmitted on the correct line, and the time differential to transit the on-line system.
2. A check is made character-by-character of the message sent against the message received to see if any characters have been dropped or in some way modified from the original message. Characters are checked within boundary limits that are coded in the input data. This permits processing programs on the on-line system to modify the messages and still have some sections verified.
3. ATOLS contains the difference in the time the message was sent and the time it was received, the line it was received on, and the sequence it

was received. This is used to verify the sequence of priority messages in a system where several levels of priority transmissions exist.

The off-line system produces these reports on the high-speed line printer for further analysis. The significance of this is that a test of several hours under a full load, where thousands of messages are transmitted, is a virtually impossible clerical function to analyze for minute errors. ATOLS reduces this to a meaningful task, since it reports only on errors. The analyst need not only concern himself with the error traffic, rather than the total.

Other off-line functions, which are very useful are a series of utility programs to generate the communications network. The network to be generated for a given test is specified symbolically on punched cards. It is quite simple to run several completely different tests just by changing the network, and switching terminals. This facility allows us to compress time in an implementation schedule when remote installations are scheduled to cutover during several months or year periods. By generating a network now that will look like the network two years from now, for instance, we can test the system and debug it under that load and be assured today that when that 200th terminal is cutover some 18 or 20 months from now, the system is not going to collapse when the first message is sent. By providing network flexibility, these functions become available quite readily.

THE ATOLS ON-LINE SYSTEM COMPONENT

The on-line system component that provides the actual test vehicle is quite similar to most message switching systems. It is in fact a complete message switching system with all of the bells and whistles, plus a few subtle differences. The first being that only a minimal amount of message processing is done in the switching system because of the flexibility required. Second, error conditions are handled somewhat differently than they would be in a conventional on-line system in that ATOLS is more concerned with identifying time stamping and saving error conditions than it is in correcting them. Therefore, the test vehicle will time stamp and collect on file any data and or error condition that is detected on the lines or in the response to its stimuli. Aside from this, it has all of the components of a store-and-forward message switching system.

The software structure contains five main line

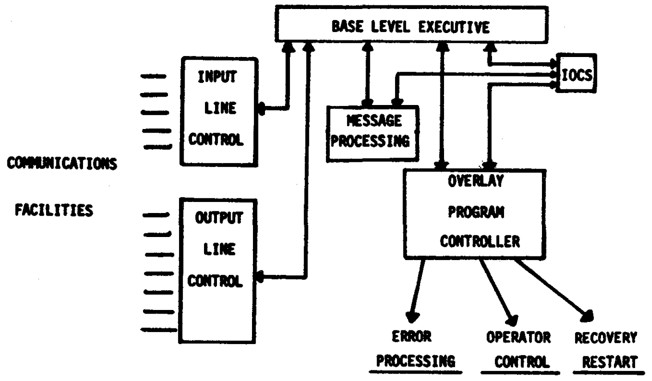


Figure 5—Software structure

program subsystems, which are:

1. The Base Level Executive.
2. Communications Control.
3. Message Processing.
4. Input/Output Control System (Non-communications).
5. The Overlay Controller.

These are the mainline resident programs. There are other sub-systems for checkpoints and recovery, operator commands, and several utility and report programs to complete the system software. The latter are mentioned briefly to clarify the text as required; however, they are not worthy of a great deal of detailed explanation at this time, since they are not concerned with the main concept to be presented.

Figure 5 is a diagram of the software structure of the five mainline programs.

The Base Level Executive is the main control program of the test vehicle. Its functions are to determine the priority of the other major elements of the test vehicle and to pass control to these elements as required. Priority of execution of each element is established by its relative position in a monitor control table which is set up at system assembly time. All programs release control to the Base Level Executive, which maintains the status of this table and allocates resources accordingly.

When the Base Level Executive Program has serviced all the work it has to do in its control table and nothing is outstanding, the Executive loops in an idle timing loop, waiting for another activity to require action. In this state, it keeps track of this idle time, which is provided as one of the statistics in the system.

The Communications Control Programs are logically divided into input communications and output communications. The responsibilities of these are essentially that of adaptive teleprocessing line control. The Communications Control Program has the following responsibilities:

1. To determine for each line in the system whether to send or receive on that line.
2. To address it or poll the line.
3. Determining if it must be dialed-up or it is private wire.
4. Receiving and logging any data other than the normal line addressing characters.
5. Notifying the logging and journaling programs that data has been received.
6. Providing all of the error testing and reporting for the transmission control units, lines, and the data.

The Line Control Programs are responsible for providing the characteristics of each line discipline that the system is testing. Also, it provides for the other functions of lines and terminals, such as to set lines up and down if excessive errors occur. Another function is to keep the system operator informed of network status, lines that are in trouble, and the characteristics that are on those lines. It also provides the basic timing functions for open lines, stuck transmitters, inter-character time-outs, no message time-outs—to protect the system from lines that may hang up for one reason or another. Again these occurrences are posted to the system operator.

The Message Processing Program is simple and straightforward. Since the test vehicle is essentially data transparent, the only requirement of the Message Processing Program is to properly route a message to its output queue. This is done from information supplied by the analyst on the data cards, indicating the line and terminal this information is to be queued to. The imbedded text of the message is not modified or analyzed in any way. On incoming messages that are generated by the on-line system, the Message Processing Program time stamps, sequence numbers, and writes the message to the journal tape for further processing.

The Input/Output Control System (IOCS) in the test vehicle schedules and executes all I/O operations for programs under the control of the Base Level Executive. With the exception of the operator's console and those communications line operations previously mentioned, IOCS contains all of the I/O channel and hardware device dependent error recovery procedures, addressing, and is initiated by a request from one of the Base Level Programs, after which it is interrupt-driven. That is to say that IOCS is dormant until a request is issued by some program under the Base Level Executive. It then schedules the requested operation and starts the physical I/O process. It is dormant again until further processing is triggered by input/output interrupts from the I/O channel. Inter-program communication between IOCS and the application program is accomplished through data control block linkages and the data per se. The programs currently support the disk drives, magnetic tapes, card reader punches, and high-speed printers. The IOCS program is queue driven and has several levels of priority so that it can be made sensitive to a particular type of data if that is desirable for the test.

The Overlay Controller Program and its buffer area is used in ATOLS to reduce the main core requirements for the system. Generally, Overlay Programs are used for supervisory controls, special processing routines, error procedures, and seldom used routines—such as retrieval, statistics programs where there is no restriction on time and/or space required. Any number of Overlay Programs may be included in the system since they are resident on external storage. Any Overlay Programs that are executed in the system under the direct control of the Overlay controller are initiated by the Base Level Executive Program. The Overlay controller is queue driven, such that any routine wishing to request an Overlay Program places the appropriate request information in a common queue for the Overlay Controller Program. When the request reaches the top of the queue, the requested program if it is possible at the time will be pulled in off the external storage device and executed.

This method of using the Overlay Program is under the direct control of the controller module, which permits some specially tailored routines for a particular on-line system to be included into the overall system without a great deal of effort required for programming. Since all of the mechanics of loading, executing, and scheduling are there, the only thing that need be provided is the actual processing program. This method allows for a certain amount of personalization or custom tailoring of software to a particular system to be tested.

The test vehicle is equipped with a command console to enter various orders to the system to re-configure networks, start up and shut down, recovery, retrievals, and things of this nature. In general, the types of commands which can be entered are to two categories. First there are network orders and second, general orders. In the area of network orders, we have line command orders and terminal command orders. These orders provide for dynamic network modifications, such as lines up, lines down, queue status reports, terminals up, terminals down, terminal status reports, and those functions associated with network parameters. The general orders group is a catch-all for everything else—set the time, set the date, start the system up, shut the system down, etc. These functions are provided so that the test director or the programmer debugging the on-line system has the ability to direct the test vehicle to perform in various degrees, depending on what options he is debugging at the time. He may want to run just one line, or he may want to run a hundred lines; he may want to stagger them, or produce instantaneous peaks. The programmer debugging the on-line system needs a great deal of control over the test vehicle to make sure that it provides the external inputs that he wants, when he wants them, and in the manner that he wants them.

Now, with all in readiness, the on-line system to be tested waiting for an external stimuli, the test vehicle primed with the test data to be transmitted, the programmer then brings up the test vehicle in a fashion that is suitable for the test—those lines and terminals that he requires. At this point, the test vehicle takes over and begins driving the on-line system in the way the 'real' network will when cutover takes place. The programmer then sitting across the room on the on-line system can then proceed to debug various features including load capacity, graceful degradation, etc. As the test system is driving, the programmer has at his command the ability to increase the load, run in an ambient state, or decrease the load by entering commands into the test vehicle as the test proceeds. While the testing is in progress, the test vehicle is logging the activity that is going on in the network to magnetic tape. This magnetic tape will become one of the inputs to the off-line programming system for the analysis of the run, if required. In addition, as the system is running, a constant stream of information is printing on the operator's console to inform the programmer of how the network is reacting, responding, what the queue status is, that the alarms are going off, and what errors have been detected.

The network and system constraints of ATOLS are primarily the amount of core memory available, and the band width capability of the processor. The network specifications for this implementation provide for:

1. 256 communications lines.
2. Several thousand terminals.
3. Supporting most of the popular line disciplines, on a line-by-line basis (adaptive).
4. Master/slave mode of operation—adaptive by line.

OTHER ATOLS USES

One of the extra benefits of the system has been to monitor and, indeed, substantiate modifications made to the on-line system several months after cutover. These modifications have been completely tested before being incorporated into the on-line system. We were able with a reasonable amount of insurance to go on-line with 'new' programs. In addition, the process of training operations and maintenance personnel has been greatly enhanced since we can provide real situations for analysis; we can overload the system, have the operators go through the fall back and recovery procedures (this is especially useful when an OLRT system stablizes—the operations personnel tend to get rusty over a period of several months). ATOLS provides for continuous training throughout the life of the system, as required.

One benefit derived from the ATOLS system that was not anticipated at the outset of the project was monitoring of the on-line system. The test vehicle was modified to 'ride' the lines of the on-line system and to capture and time stamp the activities on those lines—sort of a 'big-brother' concept. We were time stamping data transmitted on the lines and monitoring the on-line system performance without having to introduce a large amount of test or monitor equipment. The test vehicle itself deals with the collection, and the off-line package provided the results.

BENEFITS

I believe at this point, many of the benefits are obvious. However, to mention a few:

1. The throughput and communication line error rate can be measured.
2. All of the error paths of the on-line system can be exercised randomly and over a long period of time.
3. Special features that may be built into the on-line system that may not occur very frequently may be tested.
4. Peak load and duration analysis can be made on the system.
5. Behavior patterns can be developed.
6. Routing information can be checked.
7. The entire problem of fall back, recovery switch-over under various load conditions can be completely tested.
8. The so-called degradation factors can be checked and the entire system configuration of the on-line hardware can be put to the test even though the lines and/or terminals may be several months from installation.
9. Time compression—The entire system for many months and many years can be run continuously;

those things that are planned in the future can be checked out now while the system and the software is fresh in the development terms.

SUMMARY AND CONCLUSIONS

The implementation of an on-line system can be a traumatic experience. The failure rate is high, largely due to the lack of tools available for testing and debugging large, complicated, time-dependent systems. It is my opinion that the only real way a large on-line system can be satisfactorily debugged is by another computer; as one computer to another supplying the inputs as they are required. The test vehicle described here is a giant step in providing such a tool. At the same time, it falls far short of being a panacea. However, experience to date has proved it to be a very liable solution to an extremely difficult problem. OLRT systems are going to be around for a while and there will be evolutions of systems even larger and more complicated. The cost of developing a test vehicle such as this becomes trivial and can be written off over several systems to be developed by the company over a period of time. It has proven to be more than simply a good, sophisticated debugging tool to test out a system before it goes on the air. It has also proven to be an invaluable training aid on a continuous basis to train maintenance programmers joining the organization to get some experience without upsetting the on-line environment, to train rusty operators, and to test other applications that are not necessarily involved with the on-line environment but run in the background of the on-line system.

Considerably more work should be done in this area to reduce the mortality rate of OLRT systems. The costs are tremendous; the probability of success must be increased. This is the challenge for the real-time systems community.

PORTS—A method for dynamic interprogram communication and job control*

by R. M. BALZER

The RAND Corporation
Santa Monica, California

INTRODUCTION

Without communication mechanisms, a program is useless. It can neither obtain data for processing nor make its results available. Thus every programming language has contained communication mechanisms. These mechanisms have traditionally been separated into five categories based on the entity with which communication is established. The five entities with which programs can communicate are physical devices (such as printers, card readers, etc.), terminals (although a physical device, they have usually been treated separately), files, other programs, and the monitor. Corresponding to each of these categories are one or more communication mechanisms, some of which may be shared with other categories.

The "alphabet soup" in the following example is used only to indicate how diverse communication mechanisms have become. In IBM's OS/360,[1] communication with physical devices is through either BSAM (**B**asic **S**equential **A**ccess **M**ethod) or QSAM (**Q**ueued **S**equential **A**ccess **M**ethod); terminals use BTAM (**B**asic **T**elecommunications **A**ccess **M**ethod), QTAM (**Q**ueued **T**elecommunications **A**ccess **M**ethod), or GAM (**G**raphics **A**ccess **M**ethod); files utilize BSAM, QSAM, BDAM (**B**asic **D**irect **A**ccess **M**ethod), BISAM (**B**asic **I**ndexed **S**equential **A**ccess **M**ethod), or QISAM (**Q**ueued **I**ndexed **S**equential **A**ccess **M**ethod); communication to other programs is through subroutine calls, and to the monitor through Supervisor Calls. There are ten different mechanisms for the five categories; each mechanism has different commands for the utilization of the communication mechanism.

We propose that Ports offer a single unified mechanism for communicating with any of the five entities. Besides simplifying communications, this unification allows the dynamic specification of the entity being communicated with at execution time. This delayed binding can be effectively utilized for both debugging and building more flexible programs, and as a means for creating modular programs that can be easily plugged together to form systems. The remainder of this Report is devoted to defining Ports, explaining their use, and justifying the above claims.

EVOLUTION OF PORTS

The concept of Ports evolved several years ago from work on a somewhat mistitled paper called "Dataless Programming."[2] In that effort, we tried to develop a programming language that would enable representation for data structures to be selected after a program was completed rather than before it was begun. Selection of a representation after a program is written is much more appropriate because at that point the programmer knows exactly how the data are used; beforehand he must predict the actual usage. The different syntactic forms used in common programming languages for the different representations force the decision to be made at coding time. "Dataless Programming," by using a common syntactic form and by extending the operations across all the representations, allows the decision to be delayed until after coding is completed. In addition to the chosen set of standard representations, the user could create his own

* This study is part of RAND's ARPA sponsored research to improve man-machine interaction under contract DAHC 15-67-C-0141.

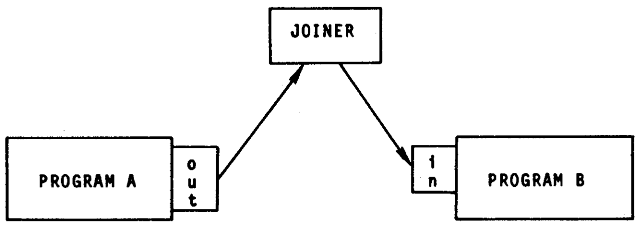


Figure 1—JOINER example

representations by supplying the necessary manipulative routines for use by the compiler in accessing, updating, adding, deleting, or inserting an element from the representation, or obtaining the next or previous one.

Because "Dataless Programming" was never implemented as a system, we tried other ways to test its ideas. The key concept was the ability to invoke a routine, either standard or supplied by the programmer, whenever a data structure was used. Not desiring to write a compiler, we looked for a centralized mechanism that could be controlled to invoke the proper manipulative routines. Such a mechanism exists in IBM's OS/360,[3] the Data Control Block (DCB) used for files. Whenever an action is required on the file, such as read or write, the address of the appropriate routine is obtained from the DCB. These addresses are placed in the DCB at the time the file is opened. The open process was modified so that for selected files, the address of an interface program, JOINER, was placed into the DCB rather than the address of a standard OS access method.

The JOINER program acted as an interface and controller between two DCBs that it had logically connected together. Thus, the output of one program was available as input to another program. Each program acted as the access method for the other. Consider Figure 1. Program A has a DCB, called OUT, used for output that has been joined to a DCB, called IN, used for input in Program B.

Assume JOINER has loaded Programs A and B, and has started A. Program A will open DCB OUT, and the address of JOINER will be placed in the DCB. Eventually, A will try some output through the OUT DCB, invoking JOINER. JOINER now starts B, and when B performs an input operation on its IN DCB, JOINER gives B the output from Program A. When B asks for the next input, JOINER suspends the program and restarts A to obtain more output to give B as input. JOINER thus coordinates the two programs and allows each to be used as the access method for the other. Notice that a type of co-routine[4] relationship is established between the programs. This relationship is called Data-Directed Co-Routines because control is switched back and forth between the two programs as data are produced and required. Also note that the connection between the two programs exists outside of each of them, and that they are unaware of who they are communicating with.

The JOINER system described contains the key elements of Ports (defined in the next section). However, we need to demonstrate some practical uses for this system because it tests the ideas in "Dataless Programming."

We first add some macros to IBM's assembly language, which gives it a control block structure. These macros are IF, ELSE, and ENDIF.[5] The IF macro begins a control block that is executed if and only if the condition tested by the macro is true. This control block is ended by either an ELSE or ENDIF macro. The ELSE macro ends the IF control block and starts an ELSE control block that is executed if and only if the condition tested by the IF macro is false. These macros can be nested, and hence a non-interactive control structure analogous to those of PL/1 or ALGOL is created. We find that these macros are very heavily used and that the nesting levels often extend ten levels and beyond. Hence, to make the program more readable, we build a formatting program that names the levels and indents the listing according to these levels.

Then, with JOINER, we connect the output of the assembler with the input of the format program. The connection is specified to JOINER and neither program is altered. By joining these two programs, we reduce both our CPU and I/O charges and the elapsed time needed to run the job.

The second application is even more important as it is the basis for an entire time-sharing system built under O/S. The RAND-built system is called Simultaneous Graphics System (SGS).* When a job is to be started, SGS joins the input of an O/S reader to the output of a spool program. The spool program is necessary because the source files are kept on the disc in compressed form as a linked list so that they can be very rapidly updated. The spool program follows the linked list and converts the file to the required sequential set of 80-character card images. When the job is running and requires input from or output for the SGS file system, its DCBs are joined with the spool program to provide the needed conversions. In this way, we are able to run, unmodified, standard OS/360 programs that utilize the SGS file system, including such IBM processors as the PL/1 compiler and the assembler.

* SGS is an internal RAND time-sharing system.

DEFINITION AND IMPLEMENTATION

As presented in the preceding sections, Ports can be defined as a data element used for communication with files, terminals, physical devices, other programs and the monitor. Four basic operations can be performed on Ports. They can be CONNECTed to or DISCONNECTed from another Port, and data can be sent (SENDed) or RECEIVEd through a Port. One compound operation, REQUEST, consisting of a SEND followed by a RECEIVE, and used for requesting certain data, also exists. The reverse sequence, RECEIVE followed by a SEND, used for replying to a REQUEST, does not exist as a single operation because an arbitrary amount of processing may be done between the RECEIVE and the answering SEND.

This definition, although containing the essence of Ports, does not answer many questions about Ports and the way they operate. We need to know how data are passed through a Port; when control is transferred to the co-routine; what happens if two SENDs occur before the first one is processed by the co-routine; if two Ports can be connected to a third; and how Ports are connected to a terminal, physical device, or file; etc. Ports can be logically implemented in many different ways, each providing different answers to the above and similar questions. Each way is a logical implementation—one that produces logically different behavior as a result of the operations. We describe Ports in terms of one such logical implementation, ISPL,[7,8] rather than JOINER, in which we are severely limited by the environment.

Incremental System Programming Language (ISPL) is both a language and an environment for programming. The ISPL language is an incrementally compiled PL/1-like language designed to run on the ISPL machine, which is designed specifically to run programs written in the ISPL language, and is intended for implementation through micro-code. As of this writing, the ISPL system is being implemented by a RAND development team. All further discussion of Ports is in terms of this logical implementation.

In this implementation, Ports are defined in terms of "data semaphores," an extension we have made to Dijkstra's semaphores[8] allowing data to be associated with such semaphores. We have extended his definition as follows (the extensions are in italics):

Semaphores are a basic language data type used for synchronization. A semaphore logically consists of a count of the available resources of a particular type. The only legal operations on a semaphore are the *P*, *V*, *and conditional P operations.* The *P* operations request one resource. The semaphore's count is decremented by one, and if the result is non-negative, the requestor continues. Otherwise, the requestor must wait until the resource is made available. The *V* operation makes a resource available. It increments the semaphore's count by one and if the result is non-positive, one of the waiting requestors is reactivated. *The conditional* P *operation performs a* P *operation only if the requested resource is available, and returns an indication of whether the resource was obtained or not. Semaphores may, in addition, have a datum associated with the available resource. Such semaphores are called data semaphores, and the legal operations for these semaphores are* P-*data,* V-*data, and conditional* P-*data, which are like their non-data counterparts except that the* V-*data operation must also supply the data to be associated with the available resources, and the* P *data and conditional* P-*data operations must specify a variable to which the data associated with the requested resource will be assigned. The data can be any item in the language to which the assignment operator applies, or a structure of such items. The data can be buffered in a stack or a queue, providing respectively, LIFO and FIFO availability. They may also be stored unbuffered for those data semaphores whose count is never greater than one.*

Using the definition for data semaphores, we define Ports as a basic language data type used for communication. They consist logically of a pointer to the Port to which the connection is made, and a data semaphore representing the availability of and the actual data being passed through the Port. The only legal operations on Ports are CONNECT, DISCONNECT, SEND, RECEIVE, conditional RECEIVE, and REQUEST.

Because Ports are used for a type of co-routine call, we feel the same mechanism used for transmitting data to a subroutine should be used for Ports. Thus, the data physically passed through the Port and its data semaphore is a pointer to an actual parameter list, the contents of which are accessed by the receiver through a formal parameter list. As with subroutines, the data logically passed through a Port and its interpretation are established as a convention between the communicating programs.

The CONNECT command interconnects two Ports by setting their pointers to reference each other. DISCONNECT sets the two pointers to NULL.

When two Ports are connected, the Port specified in a SEND, RECEIVE, or REQUEST command is referred to as the local Port, and the Port it is connected to as the remote Port.

The SEND command builds an actual parameter list from the data specified in the command, and performs a V-data operation on the remote Port's data semaphore with a pointer to the actual parameter list as the data. The data in the actual parameter list is now available to be received through the remote Port. The RECEIVE command performs a P-data operation on the local Port's data semaphore specifying an internal cell to which the parameter-list pointer will be assigned, and which will be used by the language's standard mechanism for accessing formal parameters. If no data is available, then the requestor is suspended until one is available. The conditional RECEIVE is similar, except that a conditional P operation is used. The REQUEST command is simply a SEND followed by an unconditional RECEIVE.

We have, so far, described the operations on Ports in situations where two Ports are interconnected, but have not handled the cases where a Port is connected to a terminal, physical device, or file. Terminals and physical devices are handled by connecting the Port to a Port in a device-dependent system program for the terminal or physical device that transforms the communication into I/O commands appropriate for the device, and which then requests the supervisor to perform the I/O through the MONITOR Port (see the following section).

Files are handled similarly, except that the determination of the program to which the connection should be made is based on the type of file specified. The ISPL file system[9] is based on the "Dataless Programming" principle that representation-extension capabilities should be provided by allowing the user to supply the manipulative routines necessary to implement the new representation. Thus, corresponding to each type of file, there exists a set of manipulation routines for creating, destroying, connecting, disconnecting, and communicating with files of that type. When the CONNECT command is issued, the file name is found in the master directory and its file type is used to access and execute the connect routine, and to access the communication routine that is connected to the specified Port. Ports are thus always connected to other Ports. For terminals, physical devices, and files, the remotely-connected Port is in a program selected by the system on the basis of the characteristics of the terminal, physical device, or file.

The questions on detailed Port behavior posed in this section have now been answered except for specifying when control is transferred to the co-routine. To provide the flexibility we require, the control structure of ISPL is necessarily complex. Scheduling decisions are made at three levels. First is the process level. In ISPL, a process is a set of independent tasks that share a separate, unique, addressing space. It roughly corresponds to a job. Processes are scheduled by their supervisors that are informed via an interrupt when one of their processes, which is waiting for some resource, is again able to run. Nothing more can be said about process scheduling because each supervisor can use its own arbitrary scheduling algorithm. All scheduling within a process is controlled by the ISPL machine. Each task within a process is a logically independent flow of control that could be executed simultaneously with other tasks if multi-processors were available. Each task has a relative priority, and the task with the largest relative priority that is not waiting is scheduled by the ISPL machine. Tasks, in turn, are composed of exclusive-execution blocks that are separate flows of control, but only one of which can logically be executing at once, even in a multi-processor system. As with tasks, the ISPL machine schedules exclusive-execution blocks within a task on the basis of their relative priority among those not waiting. The important difference between the two is that if an exclusive-execution block is interrupted by a higher priority one, it will not be resumed when the higher priority one waits for some resource, as is the case for tasks, but must wait for the higher priority exclusive-execution block to exit. This control structure is required for the implementation of co-routine and the on-units of PL/1.[10] An exit occurs when a program completes or does a P operation on a synchronous semaphore—one which will not asynchronously be Ved. Because it will not be Ved asynchronously, it must be an exit so that some other exclusive-execution block in the task can cause it to be Ved. In ISPL, each semaphore and Port can be either synchronous or asynchronous. Thus, the control flow resulting from SEND and RECEIVE operations on Ports depends upon whether the remote Port is in the same process or same task, and what its priority is relative to the executing exclusive-execution block. This structure enables us to build control structures ranging from completely asynchronous execution to those that switch control every time a SEND or RECEIVE is executed.

USAGE

Ports can obviously be used to communicate between programs. But the capability to externally specify the connection, and the arbitrary nature of the program to which the connection is made, enable the Port mechanism to be utilized for a variety of other purposes.

Since batch and multiprogrammed monitors, job control has traditionally been handled through a special language. This job control language has two main functions, allocation of resources and fitting the job into an environment. Fitting the job into an environment consists of setting up the communication paths between the job and the files, terminals, physical devices, programs, and monitor with which it is to communicate. This function is precisely what Ports are designed for, and is specified via the CONNECT command. In ISPL, each job has a Port named MONITOR, and it is used for all communication with the job's monitor. Because any program can be connected to this Port, this design allows for a hierarchical system of monitors, each controlling the jobs running under it. Naturally, ISPL's hierarchical design relies on much more than the Port mechanism (see Reference 7 for a full description), but Ports solved the communications requirements of the system.

Communication with the monitor through a Port provides the mechanism for handling the other main function of job control, allocation of resources. The creation and deletion of files, the allocation of file space, the allocation of core space for the job, and the specification of the central processor requirements are all transmitted to the supervisor through the MONITOR Port. The format of these specifications is a convention established by the supervisor.

Ports can also be used for debugging and simulation purposes. Output from a program can be routed to a terminal, and input obtained from the terminal so that a user can dynamically supply test data based on the program's performance. The user can also simulate the behavior of part of the system while observing and debugging the rest. A TEST program can be written to implement data breakpoints. That is, whenever the data transmitted through the Port to which the TEST program is connected satisfy the test condition, a 'break' occurs and the user at a terminal is notified or a printout occurs. The output of the TEST program is the same as its input so that it does not affect the logical processing of the program being debugged. A SPLITTER program, whose two outputs are the same as its one input, can be used to monitor, copy, or provide an audit trail of the data transmitted through a Port.

The last two programs mentioned, TEST and SPLITTER, offer examples of what we hope will be the major impact of the Port concept—a mechanism for the construction of systems from small general-purpose "plugable" programs.

Perhaps the single most important problem facing the computer industry today is our inability to generate, cheaply and quickly, debugged software systems. Many people have proposed modularity as the solution, but such systems have been hard to construct because of the strict hierarchical nature of subroutine calls—the only common method of linking together such a set of programs.

The Port concept improves the construction of modular systems in three important ways. First, the entity to which the connection of a Port is made need not be specified within that program, and can be dynamically decided at execution time. Second, the linkage is co-routine rather than subroutine. As others have suggested, this simplifies the construction of many programs, enables retention of context, and removes the strict hierarchical organization dictated by subroutine linkage. Finally, connection of a Port can be made not only to Ports in other programs, but also to terminals, files, and physical devices. Thus, the same system can, with different connections, be used in a variety of ways: on-line, off-line, audit-trailed, data-breakpointed, or partial-user simulation.

The effectiveness of the Port concept results from the combination into a single mechanism of three powerful software techniques: co-routines, indirect specification, and communications commonality. We expect to extensively test the concept, especially its modularity potential, through its implementation in ISPL.

REFERENCES

1 *IBM System/360, Supervisor and data management services*
Form C28-6646 IBM Corporation Poughkeepsie N Y 1967

2 R M BALZER
Dataless programming
AFIPS Conference Proceedings FJCC 1967 Vol. 31
Thompson Book Co. Washington D C 1967 pp 535-545

3 *IBM System/360, System control blocks*
Form C28-6628 IBM Corporation Poughkeepsie N Y 1967 pp 21-78

4 M CONWAY
Design of a separable transition-diagram compiler
Communications of the ACM Vol 6 No 7 July 1963 pp 396–398

5 R M BALZER
Block programming in OS/360 assembly code
The RAND Corporation P-3810 May 1968

6 R M BALZER
The ISPL language specifications
The RAND Corporation R-563-ARPA (In process)

7 R M BALZER
ISPL Machine: Principles of operation
The RAND Corporation R-562-ARPA (In process)

8 E W DIJKSTRA
The structure of the 'THE'-multiprogramming system
Communications of the ACM Vol 11 No 5 May 1968 pp 341–346

9 E HARSLEM J HEAFNER
The ISPL basic file system and file subsystem for support of computing research
The RAND Corporation R-603-ARPA (In process)

10 *IBM System/360, PL/l reference manual*
Form C28-8201 IBM Corporation Poughkeepsie N Y 1968

Automatic program segmentation based on boolean connectivity*

by EDWARD W. VER HOEF

Ocean Data Systems, Inc.
Rockville, Maryland

INTRODUCTION

The past few years have seen a significant increase in the number of computers with segmented memories, i.e., computers in which executable memory is divided into a fixed number of fixed length segments. A computer so organized can offer significant advantages over the more conventionally organized machine, the foremost advantage being that such an organization facilitates running a job with only a portion of that job in executable memory. This can be done in other computers but the determination of what portions of the program must be in executable memory at any given point in the process is then the responsibility of the person writing the program, i.e., the programmer must schedule the overlaying himself and reflect this schedule in the program.

In most segmented memory computers such decisions are made by the executive or operating system. The programmer need not even be aware of these decisions. This is made possible by the fact that in such computers, any reference from within one segment to something outside that segment (whether a fetch, store or transfer) causes a trap to the executive. The executive then determines whether the segment containing the referenced item is in executable memory. If it is, the action takes place as desired; if not, the desired segment is moved from peripheral storage to executable storage and then the action takes place. Thus the advantages of such machine organization arise from this latter action but the price of these advantages lie in the executable action required even when the desired segment is already in executable memory. Techniques have been developed to minimize this cost but it cannot be eliminated completely.[1,2,3] The main thrust of these techniques has been devoted to making the executive action as efficient as possible.

However, Lowe[4] has shown that in a heavily loaded system of this type, even a small reduction in the inter-segment activity results in a significant increase in efficiency of the system. Ramamoorthy[5,6] made the observation that the paths of possible control flow may be represented by a directed graph. Furthermore he presented a method for reduction of inter-segment references which was based on the principle that the instructions represented by a maximal strongly connected subgraph (i.e., a strongly connected subgraph that is not a proper subset of any other strongly connected subgraph) should not be split into two or more segments. The difficulty in this technique is that an entire program could be a maximal strongly connected subgraph and be larger than a segment.

This article presents an algorithm for partitioning a program into some number of pieces (called pages) such that none of the pages exceeds segment size and the number of interpage (inter-segment) references is reduced. This algorithm operates solely on connectivity and size data describing the program, data readily available from a compiler or assembler with relatively modest modification. As mentioned above, inter-page references can be a fetch or store of data or a transfer of control. For simplicity of presentation, only the latter are considered in this article. However, in the research project on which the article is based, both constant and variable data were treated in like manner.[7]

OVERVIEW OF ALGORITHM

The basic premise of the algorithm is that one should first optimize loops and then optimize the linear portions of the program. This is simply a recognition of

* The work presented here was performed at Informatics, Inc. (Washington, D. C. Division). It was supported by the U.S. Air Force Systems Command, Research and Technology Division, Rome Air Development Center, Griffiss Air Force Base, New York, under contract F30602-68-C-0285.

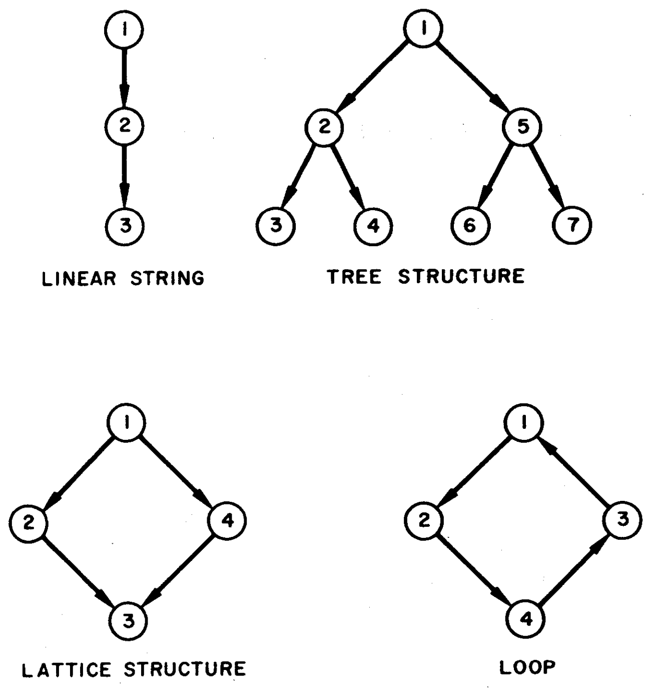


Figure 1—Types of structures

the fact that the code in a loop is executed many more times than the code in a linear portion of the program. Thus any inefficiency in a loop is paid for many times over. The algorithm operates in three phases. In phase 1, loops are detected and their component instructions are identified. The smallest loops are found first, followed by successively larger loops. The size of a loop is measured in terms of the number of program units (to be defined later) that comprise the loop. Size is a measure of the estimated time required for execution of the loop.* As soon as a loop has been detected the program units comprising the loop are merged and thereafter treated as a single unit. Phase 1 is finished after the largest loop that could fit in a single segment has been considered.

When phase 2 begins, the program consists of a collection of program units forming a linear string, a tree structure, a lattice structure, loops larger than a segment, or some combination(s) of these (see Figure 1). In this phase, a particular path is traversed and the program units along that path are identified and merged until segment size is exceeded or the end of path is encountered. In either event a new path is then selected and traversed in like manner. Phase 2 is complete when all such paths have been traversed and all merging based on connectivity accomplished. However, there may yet remain merged units of less than segment size. Such units are identified in phase 3 and merged to minimize waste space.

PROGRAM DECOMPOSITION AND REPRESENTATION

The program to be analyzed is considered to consist of "units." Units are defined by Lowe[8] in the following manner. Let the smallest executable part of a program be called an instruction element, frequently consisting of a single word. An instruction unit is an ordered collection of instruction elements $e_1, e_2, \ldots e_f$ $(f \geq 2)$ such that:

1. Each instruction element of the program appears in exactly one unit;
2. For $1 \leq i < f$, the set of successors of e_i consists of the single element e_{i+1}
3. For $1 < i \leq f$, e_i is the successor of exactly one element and that element is e_{i-1}
4. The total volume of the unit (i.e., number of words of storage required) is not greater than some fixed maximum. (This maximum, for our purposes, will be segment size.)
5. There exists no $f' > f$ for which the above four conditions are true.

A special case is made for any element which has multiple predecessors and multiple successors and whose volume is less than the fixed maximum. Such an element is considered to be a unit.

Lowe has suggested that if a program is known to consist of m such units, an $m \times m$ Boolean connectivity matrix, $A = [a_{ij}]$, can be constructed where $a_{ij} = 1$ if the ith unit (called α_i) can transfer control to the jth unit (called α_j). Otherwise $a_{ij} = 0$. If $a_{ij} = 1$ there is said to be a path of length 1 from α_i to α_j. In addition it is possible to construct a row vector $G = (g_1, g_2, \ldots, g_m)$ where g_i is the volume requirement of α_i.

LOOP DETECTION

Figure 2 shows the algorithm used for identifying and merging units forming loops. The basic tool used in the detection of loops is the connectivity matrix, A, raised to some power. Lowe has developed a fast, simple technique for multiplying Boolean matrices.[9] If $D = A^n$ and $d_{ij} = 1$, there is a path of length n from α_i to α_j. In particular, if $d_{ii} = 1$, there is a loop of path

* Lowe[8] defines the timing estimates available at this stage of analysis as event time. He then goes on to develop the concept of unit time and shows a transformation from event time to unit time. Unit time provides a more accurate means of estimating running time than does event time but event time suffices for our purposes.

length n involving α_i. Furthermore, for all other members, α_j, of this loop containing α_i, $d_{ij}=1$. However, there may be more than one loop of path length n and therefore there may be more than n values for i such that $d_{ii}=1$. Thus one cannot always uniquely identify the members of a loop by merely inspecting D. If A^n, A^{n-1}, and A are all simultaneously available, this problem is solved.

Let $C=A^{n-1}$ and let $D=A^n$. Choose some i such that $d_{ii}=1$. Then α_i is a member of a loop of path length n. If there exists some j such that $c_{ij}=1$, $d_{jj}=1$ and $a_{ji}=1$, then α_j is a member of the same loop. Furthermore, if one were to start at α_i and traverse the loop, the last unit encountered before returning to α_i would be α_j. The process could be repeated, looking for k such that $c_{jk}=1$, $d_{kk}=1$, and $a_{kj}=1$, etc., until all n members of the loop have been identified. It should be noted that in addition to identifying these units, this technique also yields information regarding their sequence of execution; i.e., they are detected in reverse order of that in which they would be executed although at this point one cannot tell which will be the initial unit of the loop.*

After a loop has been completely identified, the units comprising the loop are merged in execution order into the first unit detected for this loop. To merge α_j into α_i the following steps must be performed:

1. Replace the ith row of A by the logical sum of itself and the jth row of A.
2. Replace the ith column of A by the logical sum of itself and the jth column of A.
3. Replace $a_{1j}, a_{2j}, \ldots a_{mj}; a_{j1}, a_{j2}, \ldots a_{jm}$ and a_{ii} by zero.
4. Replace g_i by g_i+g_j.
5. Replace g_j by zero.

It is to be noted that mergers are reflected in A but not in the higher powers of A. If, when it is time to merge some member of the loop, say α_k, into α_i, g_i is such that g_i+g_k exceeds segment size no further mergers are made into α_i. Instead the remaining mergers for the loop are made into α_k. This reflects the situation where the volume requirement of the loop is such that it exceeds segment size and the loop must be split between two or more pages.

It is clearly not necessary to consider loops with path length of one as no merging of units is necessary in such cases. Therefore if one examines A^n and A^{n-1} for values of $n=2, 3, \ldots, m$ (where m is the dimension of A) one finds successively loops of path length 2, 3, $\ldots$, m. If the segment size is μ and the size of the smallest unit is η (both measured in some common unit such as words or instructions), then it is obvious that $\lfloor \mu/\eta \rfloor$* is the largest number of units that could be merged without exceeding segment size. Thus it suffices to limit the above examination to values of $n \leq \min(m, \lfloor \mu/\eta \rfloor)$.

* If it were desired to detect units of a loop in order of execution rather than reverse order one need only use c_{ji} and a_{ij} in place of c_{ij} and a_{ji} respectively.

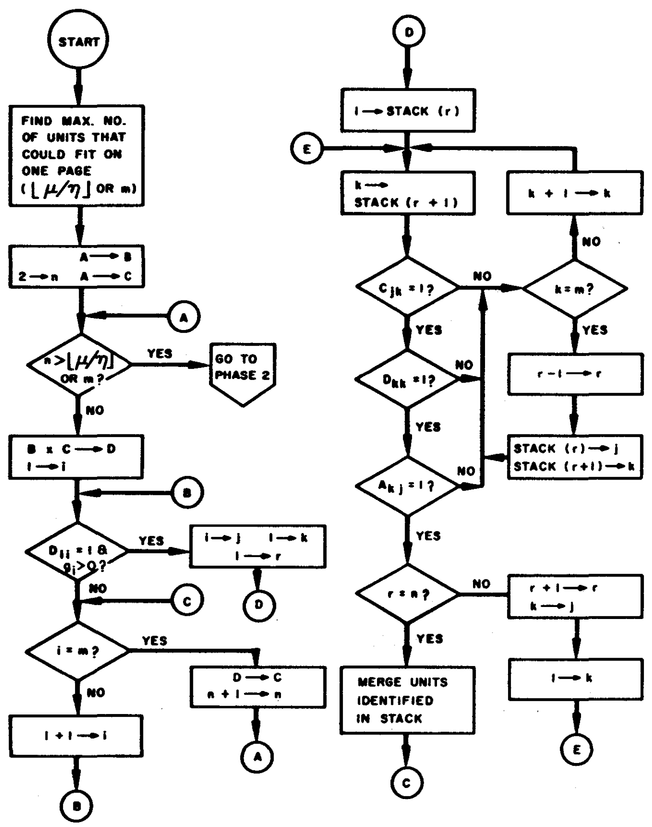


Figure 2—Loop merging

NON-CYCLIC CONNECTIVITY

As stated earlier, at the start of phase 2 the program consists of units forming a linear string, a tree structure, a lattice structure, loops larger than a segment, or some combination thereof. For illustrative purposes the example shown in Figure 3 suffices.

Figure 4 shows the approach used in phase 2 which is as follows: Find the smallest i such that $g_i>0$. Find the smallest j such that $a_{ij}=1$, $g_j>0$ and g_i+g_j is not greater than segment size. Similarly find the smallest k such that $a_{jk}=1$, $g_k>0$ and $g_i+g_j+g_k$ is not greater than segment size. In this manner one continues to move along some path from α_i to some unit α_p where

* $\lfloor X \rfloor$ is the largest integer less than or equal to X

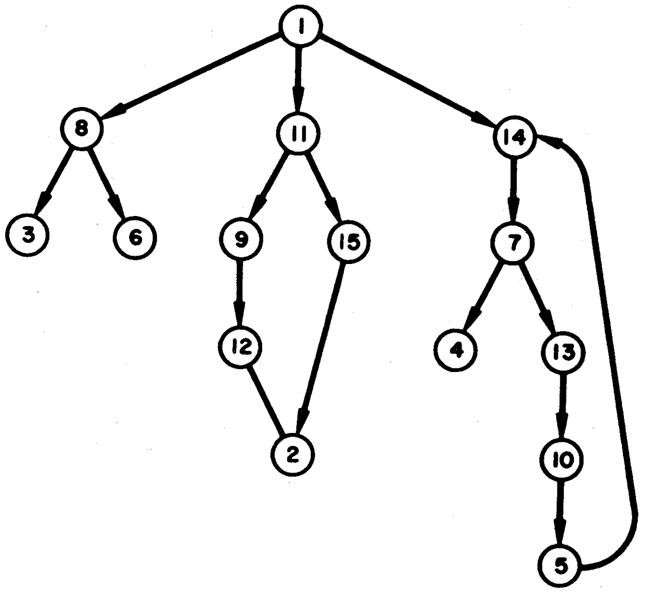


Figure 3—Program structure to be merged

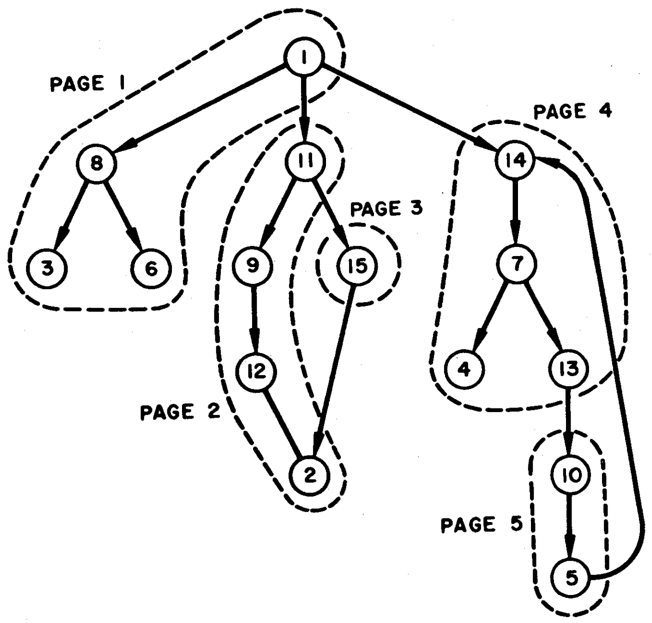


Figure 5—Merged program

either $\alpha_{pq}=0$ for $1 \leq q \leq m$ (i.e., α_p is terminal) or g_q, when added to $g_i+g_j+\cdots+g_p$ is greater than segment size. If α_p is not terminal, all units along the path from α_j to α_p are merged into α_i, α_i is put into a Last-In, First-Out list, and the path is explored further using α_q as the initial unit. Finally a terminal unit must be encountered. When that happens this unit is merged into the unit which preceded it in this path. If, after the merger, the resultant unit is not terminal, the next branch from that unit is selected as the path to be traversed. If, however, the resultant unit after merger is terminal, it is merged into the unit that preceded it in the path. The procedure will finally result in an initial unit which is terminal. This unit is then marked as having been processed and the last unit in the LIFO list is reselected. A branch from this unit is selected in the above described manner and the resultant path is traversed using the same rules as before. In all the above processes, if ever a branch is selected which leads to a unit that has already been processed (regardless of whether it was actually merged), this fact is recognized and a different branch is selected.

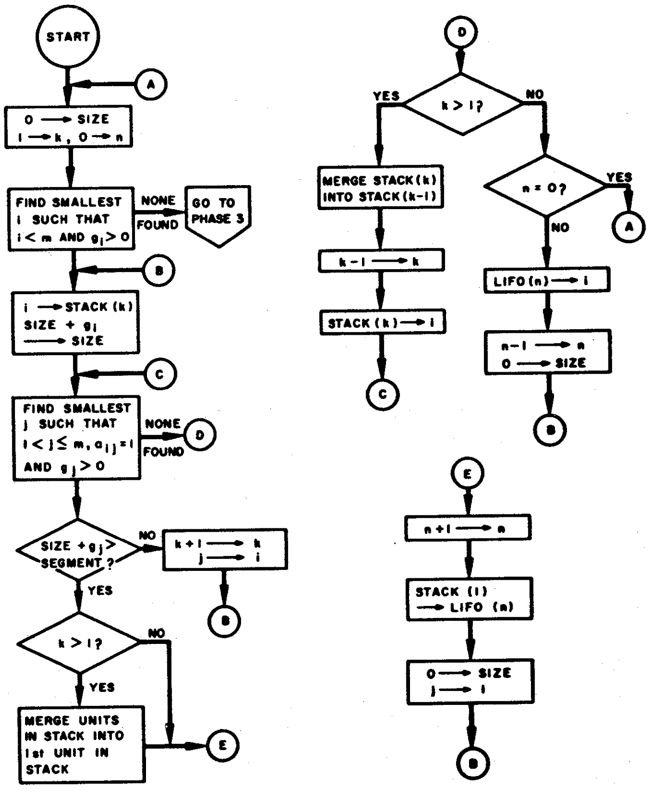


Figure 4—Non-cyclic merging

When the LIFO list is finally empty the entire structure emanating from the originally selected unit, α_i, will have been merged on a connectivity basis, subject to segment size constraints. However α_i might not be the unit which contains the entry point for the program and the program might even have multiple entry points. Therefore there might be units which were not encountered in the above process. For this reason the units are scanned to determine whether any were not processed. If any such unit is found, the process is repeated starting with that unit. Note however that there will be no repetition of analysis of units that were already processed. Only when all units have been processed will this phase terminate.

The result of applying phase 2 to the example shown earlier, assuming each unit was of equal size and segment capacity was four units, is shown in Figure 5.

CLEANUP

At this point all packing based on connectivity is complete. However, note that there might be units which are less than segment size, such as units 10 and 15. Phase 3 is a clean-up process which attempts to minimize total required storage. In this phase only the unit size vector, G, is examined. The algorithm is as follows:

1. Find smallest i such that $g_i>0$.
2. For $i+1\leq j\leq m$, if $g_j>0$ and g_i+g_j is not greater than segment size, merge α_j into α_i.
3. Find smallest i' such that $i+1\leq i'<m$ and $g_i'>0$.
4. If i' can be found go back to step 2; otherwise the process is complete.

At this point each unit of non-zero size is assigned to some page.

CONCLUSIONS

The algorithm presented above provides a simple means for partitioning programs into pages in such a manner as to reduce the number of inter-page references and therefore amount of inter-segment activity. It is based solely on information about the program that is obtainable by automatic inspection of the program such as could be preformed in a compiler or assembler. The inspection and packing algorithms are not very complex and thus should require very little time for their execution. The packing algorithm has been implemented as a post-compilation operation. It has been tested with a wide variety of program structure models and has been found to give the anticipated packings. The inspection algorithm is now being incorporated into a JOVIAL compiler specially built for experiments in segmentation.

REFERENCES

1 R C DALEY P G NEUMAN
A general-purpose file system for secondary storage
AFIPS Conference Proceedings Vol 27 Fall Joint Computer Conference Spartan Books Washington DC pp 213-229

2 E L GLASER J F COULEUR G A OLIVER
System design of a computer for time sharing applications
AFIPS Conference Proceedings Vol 27 Fall Joint Computer Conference Spartan Books Washington DC pp 197-202

3 V A VYSSOTSKY F J CORBATO R M GRAHAM
Structure of the MULTICS supervisor
AFIPS Conference Proceedings Vol 27 Fall Joint Computer Conference Spartan Books Washington DC pp 203-212

4 T C LOWE J G VAN DYKE R A COLILLA
Program paging and operating algorithms
RADC Final Report TR-68-444 November 1968
Rome Air Development Center AFSC Griffiss AFB New York

5 C V RAMAMOORTHY
Analysis of graphs by connectivity considerations
J ACM 13 2 April 1966 pp 211-222

6 C V RAMAMOORTHY
The analytic design of a dynamic look-ahead and program segmenting scheme for multi-programmed computers
Proc ACM 21st Nat Conf 1966 Thompson Book Co Washington DC pp 229-239

7 E W VER HOEF D L SHIRLEY
Block file and MULTICS systems interface investigation and programming
Final Report Vol II RADC Final Report TR-69-40 April 1970
Rome Air Development Center AFSC Griffiss AFB New York

8 T C LOWE
Analysis of boolean models for time-shared paged environments
Comm ACM 12 4 April 1969 pp 199-205

9 T C LOWE
An algorithm for rapid calculation of products of boolean matrices
Software Age 2 March 1968 pp 36-37

Partial recompilation*

by RONALD B. AYRES and RICHARD L. DERRENBACHER**

Technology for Information Management, Inc.
Albany, New York

INTRODUCTION

It is axiomatic that a great deal of the machine time required to debug and implement a computer program is used to compile and recompile source code. In many cases, recompilations require at least as much machine time as an initial compilation, although the resultant change to the program object code may be minimal. In a time-sharing environment the redundancy associated with these recompilations can be expected to contribute to deterioration of response time. Conversational debugging packages, such as DDT,[1] seem to present only a partial solution to the problem, since the debugging operation is usually divorced from the source language updating and production of a corrected object code program.

The purpose of this paper is to describe a compiler writing technique which eliminates the need for entire program recompilations. The technique is known as "partial recompilation" and has been implemented into the JOVIAL† compiler for the 1604B computer at the U.S. Air Force's Rome Air Development Center (RADC). While interpreters and incremental compilers[2,3,4] have been designed primarily for time-sharing systems, the technique described in this paper may be used in a batch environment as well, particularly when used to debug and implement very large programs which require large amounts of compilation time. The subject compiler was modified to incorporate conversational features and to operate in either a batch or in a simulated time-shared environment.

* This work was performed at TIM, Inc. and was supported entirely by the U.S. Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York under contract F30602-69-C-0316.

** Currently associated with Mohawk Data Sciences and Informatics Incorporated, respectively.

† JOVIAL is a general purpose, relatively machine independent, procedure oriented language developed by System Development Corporation. It is the official command and control language for the U.S. Air Force.

APPROACH

In the classic case, computer programs are written, debugged, and implemented as follows:

1. Write program source code statements.
2. Compile source program and produce object code.
3. If errors are detected, correct source code statements and return to step 2.
4. Execute program object code.
5. If errors are detected, prepare corrected source code statements and return to step 2.

Programmers who are fortunate enough to know machine code often eliminate the requirement to return to step 2 by modifying the program object code and continuing with step 4. Although this short cut often saves time, several pitfalls confront the programmer who elects to debug in this manner. First of all, errors are probably more apt to be made when program corrections are introduced at the machine code level. Secondly, the possibility always exists that the programmer will forget to implement a correction at the source level after it has been successfully implemented and tested at the object code level. Finally, as source language corrections are accumulated over long time spans, the possibility of the introduction of new errors is magnified.

Conversational debugging packages[1] which permit source level corrections without necessitating a program recompilation provide a similar debugging shortcut and eliminate the need to enter corrections at the machine code level. Even with such debugging packages, the other pitfalls still exist, however. Certain types of program modifications are not possible at the object code level at all, e.g., expanding or reducing the size of certain tables or buffer areas.

The technique of partial recompilation reduces program debugging time by permitting any source language modification to be made while nearly eliminating the requirement to completely recompile an entire program, i.e., by eliminating the requirement to ever return to step 2 of the program implementation process. Instead, programs are implemented as follows:

1. Write program source code statements.
2. Compile source program and produce object code.
3. If errors are detected, correct source code statements and compile only those statements necessary to effect the change in the previously generated object code.
4. Execute program object code.
5. If errors are detected, correct source code statements and compile only those statements necessary to effect the change in the previously generated object code and return to step 4.

Thus, a means of performing a partial recompilation is provided. Since the partial recompilation results in the processing of only a small number of source language statements (depending upon the number and type of source language modifications made), a significant amount of time is saved over performing an entire program recompilation. For most types of source statement modifications, the partial recompilation takes only a fraction of the time required to perform an entire program recompilation, since most of the redundancy associated with program recompilations is eliminated. The programmer need not learn machine or object code, since all communication with the compiler takes place at the source language level.

COMPILER OPERATION

Source language input records contain an alphanumeric source record identification, or "sequence number." This record identification is unique and is used to refer to a unique program record, or line. In addition to the usual editing and modification functions, the sequence number is used by the compiler as a link between the source language input statements and the intermediate information generated and saved for those compiled statements. This relationship is necessary in order that the compiler may determine exactly which source language statements must be compiled (or recompiled) based upon modifications made to a source language program.

The partial compiler operates in one of two modes, as specified by control information provided by the user. The first mode is designated the INITIAL mode and operates very much like a classic compiler. An INITIAL compilation is always performed when a program is compiled for the very first time. During this compilation, all source language statements comprising the program are compiled. The second mode is the PARTIAL mode and is used whenever source language modifications are to be made to a previously compiled program.

The INITIAL compilation is performed as follows:

1. *Generator Phase*

Source language statements are obtained from either a prestored source language file or are transmitted directly to the compiler via an on-line CRT unit. If the statements are transmitted via the CRT unit, a source language file is constructed. As the statements are obtained, the statement sequence numbers are used to construct entries in a Sequence Number Dictionary Table (SNODCT table). Source language is processed on a statement by statement basis. If a syntactic error is detected and the compilation is being performed in the time-sharing, rather than batch environment, the user is notified via the CRT unit and is given the option of correcting the error immediately, proceeding, aborting the compilation, or obtaining "tutorial" assistance in identifying the detected error. Statements are translated into binary values which contribute to the construction of various compiler lists and tables, such as the dictionary and intermediate analysis tables. The SNODCT table is constructed so that the intermediate data generated for a given source language statement may be identified.

2. *Translator Phase*

Declarative data from the dictionary is translated into machine language via entries in an intermediate object code table. This consists of reserving core storage, presetting the environment of the object program, processing long octal and literal constants, and modifying the dictionary. Secondly, imperative data from the intermediate analysis tables is translated into machine language via entries in the intermediate object code table. Temporary registers are allocated and tag locations entered into the dictionary.

3. *Assembly Phase*

The assembly phase generates an object code program from entries in the intermediate object code table. The SNODCT table is completed so that the object code program registers generated for a given source language statement may be identified.

The most noticeable difference between the classic compiler and the INITIAL mode of the partial compiler is that the partial compiler saves the generated data, such as the dictionary and intermediate analysis tables, and generates the SNODCT table as a means of relating source language statements to the compiler generated data and to the object code program registers generated for those statements. All such output from the INITIAL compilation is written on the secondary storage device and is required as input for a PARTIAL recompilation. The overhead associated with the generation of this data during an INITIAL compilation is usually more than compensated for by the time saved during the first PARTIAL recompilation. This overhead was found to never exceed 15 percent for the 5 JOVIAL programs for which timings were made. The larger mass storage requirements are imposed upon the system by the partial recompilation technique only so long as the source program is in a debug status. A purge function is used to return space to the operating system whenever a program is debugged to the programmer's satisfaction. Another INITIAL compilation will return the program to a debug status by regenerating all data necessary for another PARTIAL recompilation.

A PARTIAL recompilation is requested whenever modifications are to be made to the program source language. Rather than perform an entire program recompilation, source language modification requests are input to a PARTIAL recompilation, which operates as follows:

1. *Set-Up Phase*

The set-up phase is designed to prepare the way for a PARTIAL recompilation. Its functions include:

a. Read source language modification requests from the on-line CRT unit or from an "alter card" deck. These requests specify source language statements to be added, deleted, or replaced from the source language program.
b. Build an update file which contains all information needed to update the source language file.
c. Set indicators within the SNODCT table to indicate the presence of insertion or deletion requests and to identify source statements which must be recompiled during the PARTIAL recompilation. The number of such statements depends upon the type of source modification requests made.
d. Construct object code modification tables which indicate the points at which machine executable instructions are to be inserted or deleted from the object code program, based upon the source code modification requests and the appropriate SNODCT data.

2. *Generator Phase*

Unlike the generator phase in the INITIAL mode, the generator during a PARTIAL recompilation processes only a limited number of the statements comprising the source program. All new statements are processed, as well as any statements flagged for recompilation by the set-up phase. Dictionary entries are established for all new or recompiled data declarations and statement labels. New (temporary) intermediate data generated as a result of compiling new statements or of recompiling old statements is saved. A new (temporary) version of the SNODCT table is generated for new statements compiled as well as for old statements recompiled.

3. *Translator Phase*

The translator phase operates exactly the same during a PARTIAL recompilation as during an INITIAL compilation, with the following exceptions:

a. Intermediate object code table entries are generated for declarative data based upon those dictionary entries which are new or which resulted from a recompilation of old statements.
b. Intermediate object code table entries are built for imperative data based upon entries of the new (temporary) intermediate analysis tables rather than the permanent version.

4. *Assembly Phase*

The assembly phase during a PARTIAL recompilation operates exactly the same as during an INITIAL compilation except that in the former case a new (temporary) object code program is generated based upon the compiled and recompiled statements.

5. *Reconciliation Phase*

The reconciliation phase is divided into two parts; object code reconciliation and source code reconciliation, as follows:

a. The object code program generated during the INITIAL compilation or during the previous PARTIAL recompilation is modified based upon the object code modification tables generated during the set-up phase and the object code formed by the assembly phase of the current PARTIAL recompilation. Appropriate object code program registers are deleted, new object code inserted, and address relocation performed. The resultant object code program now embodies all requested source language modifications.

b. New source language records from the update file created by the set-up phase are combined with the original source language statements to produce an updated version of source language. The SNODCT table and intermediate analysis tables are restructured based upon the new (temporary) versions and upon the source language modification requests in the update file. All program files and intermediate data are now in such a state that another partial recompilation could be performed.

One of the major considerations in performing a PARTIAL recompilation is to determine, based upon the type of source modifications made, exactly which source statements must be recompiled. For certain types of modifications, a PARTIAL recompilation may not be possible. For example, in the subject compiler an INITIAL compilation must be performed if the organization of the environment (data declaration area) is modified via the JOVIAL OVERLAY statement.

Ideally, only new source language records would be compiled during a PARTIAL recompilation, along with all statements which contained undefined data references or statement label references during the INITIAL compilation (or previous PARTIAL recompilation). Although many PARTIAL recompilations may result in the processing of such a limited number of statements, many types of source language modifications necessitate that unchanged but related statements also be recompiled. If a very large number of source language modifications of this type are requested, an INITIAL compilation might be better, i.e., faster than a PARTIAL recompilation. These determinations are made by the set-up phase of the compiler based upon source modifications requested and upon the content of the SNODCT table.

When the SNODCT table is constructed by the generator phase of the compiler, certain source language statements are "linked" together by setting switches within appropriate SNODCT entries. If any change is made to a statement within a linked group, or if new statements are inserted into a linked group, all statements within that group must be recompiled during a PARTIAL recompilation. Statements which are linked together for recompilation purposes in the subject compiler include the following:

1. Logically related data declarations, such as table entries or entries within a certain array.
2. Certain program loops (such as the JOVIAL "for loop" or FORTRAN "do loop") and conditional statements.
3. Certain program subroutines.

A change to declarative data necessitates that references to it be recompiled during a PARTIAL recompilation since it is possible that the modifications to the data declaration should result in different object code for processing the data, such as in changing a data definition from floating point to integer or vice versa.

Also, whenever, a statement which contains a statement label and is located within a linked group is recompiled, statements which reference the label must be recompiled since the relative object code address of the statement label is apt to change. Fortunately, this does not carry on forever, that is, such recompiled statements do not cause other statements which reference tags to *them* to be recompiled since only the address reference of the secondarily recompiled statement changes.

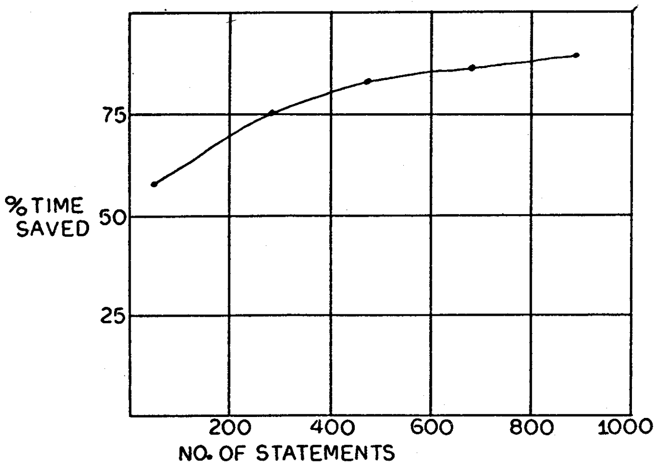
Admittedly, some redundancy still exists within the current version of the compiler. For instance, it should not always be necessary to recompile all logically related data declarations and their associated references whenever a small portion of the declaration (such as a table or array entry) is modified. Nor should it always be necessary to recompile an entire program subroutine whenever a portion of the subroutine is modified. This redundancy is the result of an attempt to keep modifications to the current version of the compiler relatively straightforward. By sophisticating the set-up and generator phases even less redundancy would exist during a PARTIAL recompilation.

TIMINGS

The partial recompilation technique was tested and timings made on five JOVIAL programs of various length. The timing figures set forth for these programs were obtained by using a high speed drum unit as the mass storage device. Compiler program loading times were subtracted from the actual elapsed times, since all JOVIAL compiler programs in the subject system are loaded from magnetic tape. Thus, timing comparisons tend to reflect time spent during the actual compilation process.

Although the secondary storage requirements given for the five test programs are representative of the requirements of the subject compiler, it must be pointed out that these requirements could be reduced by employing more efficient data storage techniques than those currently used by the compiler. For example, 80 characters of storage is currently required for each line of source language, even though the line may contain considerably fewer than 80 characters of information. Optimization of secondary storage usage should result in storage requirements at least 20 percent less than those given in this report.

For comparison purposes, an INITIAL compilation was compared to a PARTIAL recompilation during which a single insertion into the source language was made. Partial recompilations which cause many additional source statements to be recompiled could be expected to take longer than the cases presented here. Usual debug outputs, such as source and assembly listings, were suppressed during both the INITIAL and PARTIAL runs. The accompanying graph represents "percentage of time saved" by performing a PARTIAL recompilation rather than an INITIAL program compilation.



TEST PROGRAM 1

No. of statements	44
Secondary storage requirements	24,266 characters
INITIAL compilation	21 seconds
PARTIAL recompilation	9 seconds
Percent time saved, PARTIAL vs. INITIAL	57 percent

TEST PROGRAM 2

No. of statements	273
Secondary storage requirements	73,350 characters
INITIAL compilation	81 seconds
PARTIAL recompilation	21 seconds
Percent time saved, PARTIAL vs. INITIAL	75 percent

TEST PROGRAM 3

No. of statements	468
Secondary storage requirements	115,780 characters
INITIAL compilation	146 seconds
PARTIAL recompilation	26 seconds
Percent time saved, PARTIAL vs. INITIAL	82 percent

TEST PROGRAM 4

No. of statements	685
Secondary storage requirements	160,417 characters
INITIAL compilation	216 seconds
PARTIAL recompilation	31 seconds
Percent time saved, PARTIAL vs. INITIAL	85 percent

TEST PROGRAM 5

No. of statements	884
Secondary storage requirements	203,631 characters
INITIAL compilation	296 seconds
PARTIAL recompilation	36 seconds
Percent time saved, PARTIAL vs. INITIAL	88 percent

CONCLUSIONS

The technique of partial recompilation seems to present a means of reducing most of the redundancy associated with program recompilations. If implemented into a compiler which operates in a time-sharing environment, this time saving should result in improved terminal response time, particularly to the user attempting to debug a source language program conversationally. The primary debugging advantages afforded by the incremental compiler and the interpreter are available, i.e., local syntactic errors may be corrected from the terminal as they are encountered.

The major advantages of a compiler with the partial recompilation capability are:

1. A programmer need not be concerned with program object code, since a partial recompilation can be performed in less time than a programmer could determine and make object code patches to an erroneous program. Programmers should be encouraged to perform source language corrections via a partial recompilation as soon as errors are detected.

2. The need for sophisticated debugging packages is lessened since debugging commands and console typeouts may be entered at the source language level via a partial recompilation.
3. The partial recompilation technique reduces the burden (except for mass storage requirements) of the system upon which the compiler operates, since a minimum amount of time is spent on complete program recompilations.

The major disadvantages of a compiler with the partial recompilation capability are:

1. Due to the additional general housekeeping functions necessary, the construction of the compiler itself is necessarily more complex than for a compiler which does not offer the capability.
2. Additional mass storage requirements are imposed upon the system to accommodate the additional data which must be generated and saved in order to allow for a partial recompilation.

ACKNOWLEDGMENTS

The authors are grateful to their contract monitor, Captain Andrew M. Kobziar of the U.S. Air Force's Rome Air Development Center (RADC), for his very valuable contributions to the design and implementation of the partial recompilation technique. Acknowledgment is also made to Mr. Albert J. Kreutzer for programming assistance and to Messrs. Jack Rusman and Robert T. Strain for their initial guidance and inspiration.

REFERENCES

1 P V WATT
Generating and debugging programs at remote consoles of the PDP-6 time-sharing system
Proc Third Australian Computer Conference

2 H KATZAN JR
Batch, conversational, and incremental compilers
Proc SJCC 1969

3 K LOCK
Structuring programs for multiprogram time-sharing on-line applications
Proc FJCC 1965

4 J L RYAN R L CRANDALL
M C MEDWEDEFF
A conversational system for incremental compilation and execution in a time-sharing environment
Proc FJCC 1966

PL/C:—The design of a high-performance compiler for PL/I

by HOWARD L. MORGAN and ROBERT A. WAGNER

Cornell University
Ithaca, New York

INTRODUCTION

A general purpose production compiler faces many diverse and demanding tasks. It must obviously accept the full source language—including all rarely used and difficult to compile features; it must produce efficient object code; it must be prepared to accept very large and long programs—and accept them in pieces to be integrated later. It is obvious that by yielding on some of these requirements, and by sacrificing generality for efficiency for a particular class of program or user, improved compiler performance should be obtainable. One large and important class of users are those just learning the language. Their programs are typically short, submitted all at once, and require only a modest subset of the language. They need to be compiled repeatedly to eliminate errors, and are often discarded as soon as they execute properly. In general, their execution time is very small relative to the total compilation time. It should also be noted that such programs occur in tremendous numbers since classes of hundreds of students submitting programs daily are not at all uncommon. This class of program is further enlarged by the observation that even experienced programmers have rather similar requirements (except for requiring a larger subset of the language) during the development and checkout of a new system.

PL/C is a special purpose processor for PL/I.[5] It has been designed and implemented by a group of faculty and students in the Department of Computer Science at Cornell University, both to serve the needs of instruction and program checkout, and to serve as a test vehicle for some novel concepts in compiler construction and diagnostic strategy. It is an open ended project, but the work to date has yielded a very high-performance compiler for a usefully large and strictly compatible subset of the PL/I language.

This paper describes the strategies and structures employed in the design of PL/C. Surprisingly many of the techniques used appear to be generally applicable in compiler design. The overall organization of the compiler project is discussed in terms of both the technical details and the personnel assignment strategy.

DESIGN OBJECTIVES

Before commencing the design of any large system, the overall objectives must be well thought out. Four major goals, along with several smaller objectives, shaped the design of PL/C. These were:

1. High-performance—PL/C had to compile programs of the size generally found in instructional situations an order of magnitude faster than the speeds obtainable with the "production" compilers. Efficiency of the generated object code was not nearly as important, although one would not want to completely overlook this question.
2. Diagnostic—Diagnostic assistance was at least as important as processing efficiency. The goal here was one of error correction rather than simply error detection. The PL/C reaction to the discovery of an error should be to supply an explanatory message, effect a repair, display the nature of the repair to the user, and continue processing. No source error should terminate the scanning of the program, and as few as possible should inhibit execution. In addition, as many execution errors as feasible should also be handled in this manner. This approach had been developed on a series of previous compilers at Cornell, beginning in 1962, and had been found to be remarkably effective.[1,2,3]
3. Upward compatibility—In order to be useful as a debugging and checkout compiler, and to serve as a widespread aid, the subset had to be strictly upward compatible with the IBM F-level implementation of PL/I, both in language specification and in the semantics actually im-

plemented by IBM. Thus, any program which will run under PL/C without incurring any diagnostic messages will run under the IBM implementation and provide the same results.

4. Space—There are many installations which now dedicate partitions to the high performance compilers[10] such as WATFOR. In addition, there are a large number of IBM 360's of 128K and larger. Thus, PL/C had to run in a 128K (byte) machine under either the OS or DOS systems. This meant that PL/C, along with any associated symbol tables and generated object code had to run in at most 100K bytes, with the rest reserved for the operating system.

Clearly, all of these goals interact heavily, and decisions made in attaining any one will affect some or all of the others. For example, adding more diagnostic checking affects both space and speed, while increasing the subset causes the need to add more diagnostic checking.

Several subgoals were also adopted by the design group. These were:

5. Portability—Previous experience had shown that isolating operating system interfaces permitted easy conversions of compilers to new operating systems.
6. Open-endedness—As a Computer Science group, it was hoped that the structure which would emerge would be one which could be modified and added to with relative ease. For example, object code optimization, compile time facilities, and language extensions have all been proposed, and should be able to fit within the PL/C structure.
7. Breadth of subset—Goal 3. above merely stated the compatibility requirement. It was hoped that PL/C would provide far more than the FORTRAN subset of PL/I. Here the interaction with the open-endedness goal was high. If that one was met, then additions to the subset should be easy.

RESOURCES

When planning the design and implementation, the resource allocation problem becomes quite important, The PL/C design group consisted of an average of five people, all highly motivated, and all very experienced. Among them they had designed major parts of at least eight compilers and four operating systems. The personnel, all either Ph.D. or advanced graduate student level, performed not only design, but also detailed coding and checkout of the compiler. In particular, this meant that design modifications could be and were in fact carried out throughout the entire programming process. Such modifications could easily lead to disaster (or at least greatly increased implementation schedules) in a larger or more formally structured group.

The second major resource needed for a programming project is computer time. Although real dollars were used to pay for this, it was decided at the outset to treat the machine as a freely available commodity. Thus, the personnel were encouraged to use the computer in order to save their time, or the overall project time.

OVERALL DESIGN AND INTERFACES

The overall design of the PL/C implementation leaned heavily on the design of the CUPL compiler.[2] Several members of the group had participated in CUPL's design and implementation a few years earlier, and CUPL actually meets all of the PL/C design goals except for that of source language compatibility.

Experience with several error correcting compilers and operating systems has proven the utility of an explicit, syntactically correct representation of the source program as a major interface between the "scan and correct" phase and the object code generation phase of the compilers. The presence of this intermediate language code (I-code) has several advantages.

First, it permits the compiler to diagnose and correct errors in near source language form. This in turn allows error corrections to be displayed to the user in the form of a reconstructed source statement. For example, the following is a typical set of diagnostics for the statement in error:

```
          PUT LIST (A  B)
          ERROR SY06   MISSING COMMA
          ERROR SY07   MISSING SEMI-
            COLON
PL/C USES PUT LIST (A, B);
```

The pedagogical advantages of displaying correct statements to the user should not be underestimated. It will often save him the trouble of looking in a manual and trying to figure out what he should have done.

A second major advantage of I-code is that it constitutes a very clean interface between the major compiler sections. The scan phase guarantees syntactically perfect I-code to the code generation phase. As a result, the design and coding of these sections could proceed in parallel. Communication between designers took place only when changes were made in I-code or in the major interface, the symbol table. In addition, the code generation phase could be streamlined to take advantage of the fact that there would be no errors in its input.

The syntax of I-code was carefully worked out early in the design process, and was precisely documented using BNF-like notation. The independence of designers thus afforded the opportunity for each section's designer to choose and develop algorithms according to personal taste and programming style. The machine representation of I-code is a string of 16 bit items (halfwords), each representing one token of the source program. In addition, certain pointers are inserted to speed the code generation process, and to retain close links to the source language form for reconstruction purposes.

In order to further speed the code generation task, and to provide more diagnostics for the users, an intermediate pass over the I-code is made after scan, and before code generation. This pass, called "semantics" resolves problems created by the fact that PL/I is a block structured language which permits explicit declarations of variable names to appear anywhere in a block. Thus, although a statement may be syntactically correct, it may be semantically incorrect, i.e., the declared attributes of a name may conflict with that name in usage. For example, if L has the attribute "label," it is semantically incorrect to write:

$$L=L+1;$$

One of the important design decisions was to accept and solve this semantics problem without resorting to what probably would have been a simpler solution—rule out declarations after use in the source language subset. Since the above restriction would still not remove all semantics errors, it was deemed better to accept the challenge.

In a language with as rich a variety of data types as PL/I, it is only natural to expect that the design of the symbol table will have an important effect on the overall compiler design. In PL/C, this is doubly true since the symbol table is required not only during compilation but also during execution of the object program. Its use during execution is in continuing the source level error messages. For example, a typical execution error message is:

```
IN STMT NNNN ERROR EXB7
   ABC HAS NOT BEEN INITIALIZED.
                        IT IS SET TO 0.
```

The fact that this table had to be present in all phases also meant that space was at a premium.

Memory organization for PL/C was dictated by the space needs of the two major interfaces, I-code and the symbol table. The goals of the memory allocation were simple—make as much use of all available space as possible in all phases. Thus, it was desired to maximize the amount of core available for the runtime stack during execution. Since the symbol table and object code must also be present, this stack must clearly start at the end of these two and grow to the maximum core address allowed. A look at the code generation phase reveals that I-code, symbol table, and growing object code must all coexist. The organization presented in Figure 1 is now used.

During the syntactic phase, both the symbol table and the I-code must grow in the same direction. Thus, the fraction of available space allocated to the symbol table must be chosen carefully, so that the probability of the symbol table catching up to the start of I-code is small. In fact, in the first implementation, enough space had to remain between the completed symbol table and the beginning of I-code to hold most of the object code. This came about because our initial strategy was to generate code for an entire source block, thus skipping around in I-code if necessary. It was felt that this might save execution time, by reducing the number of base register loads necessary. A later pass at the design indicated that large programs could be more easily compiled if the code generation were done linearly, with branches inserted where necessary

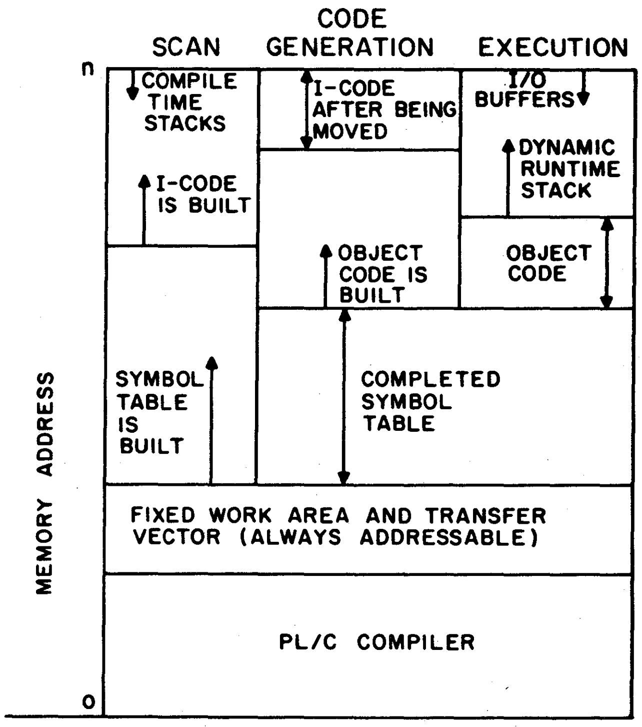


Figure 1—Memory organization of PL/C

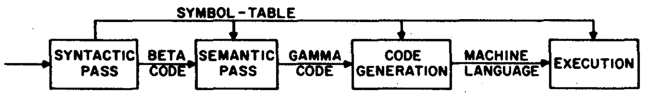


Figure 2—Overall organization of PL/C

due to the static structure of the program. In addition, the I-code is now moved so that it abuts the end of core just before code generation, thereby making the maximum possible room available. The cost of this move is negligible for small programs, but makes possible the compilation of some large programs which would otherwise overwrite their I-code with object code, before the I-code has been compiled.

Figure 2 shows the overall organization of PL/C into the three passes over the source code or its I-code equivalent. The first pass, syntactic analysis (SYNA), processes declare statements and performs syntactic analysis, error detection, and correction on the entire source program. SYNA produces the initial version of I-code (called betacode) and constructs the symbol table.

The second pass, semantic analysis (SEMA), performs the semantic error detection and correction on all expressions in betacode. In addition, it modifies these expressions into postfix form and rewrites them leaving a new I-code, called gamma code. This gamma code is the promised "perfect" intermediate language representation of the program which is then processed by the third pass, code generation (CGEN).

DETAILED ORGANIZATION

The PL/C design proceeded in two phases. First, the design of the major interfaces was performed. This portion of the design process alternated with the design of those algorithms in each pass which directly interacted with the interfaces. Design and redesign of interfaces and algorithms proceeded through roughly three iterations before coding began in earnest.

During this period of interface design communications among the design group were quite free, and all of the group were encouraged to participate in all of the designs under consideration. Once coding began, each designer became primarily responsible for some section of the compiler. Thus they then became rather concerned with the development of efficient local algorithms.

During the coding phase of the project, the value of a simple linear data flow (Figure 2) became apparent. Such a structure reduced the burden of communication among people dramatically. The person responsible for code generation knew (and needed to know) little about the algorithms and capabilities of the scan or semantics sections of the compiler. Had a co-routine organization been chosen, which would have involved calls between generate and semantics on an expression by expression basis, many more interfaces would have had to be designed and documented.

Figure 3 shows a more detailed breakdown of the compiler into actual modules, together with an indication of the personnel assigned to each task (letters are used instead of names). In addition to meeting the subgoal of an isolated operating system interface and control module, the organization was consciously designed to minimize the number of person to person interfaces which had to be created. This was done by assigning the same person to routines on both sides of what would require a natural interface. Not only is documentation reduced, but the ability to change to a more efficient interface on short notice is vastly increased.

INNOVATIONS

The combined result of allowing a high degree of independence to the individual implementers, and the amount of experience present was predictable. Each of the major sections of the compiler contains some innovation or new method of doing things. In addition, those sections which were more cut and dried employ algorithms which appear to be as efficient as any now known. All parts of the compiler were implemented in IBM 360 Assembler Language,[4] most with extensive use of the macro facilities available. This assembly language implementation allowed highly efficient code to be produced for each algorithm, and probably saved a factor of two or three in speed and space over a possible implementation in higher level language. (Another compiler being developed at the same time was forced to abandon its high level implementation language for using too much space.)

The following sections are intended to describe the most important or innovative algorithms used in the various modules of the compiler. It is our suspicion that the relatively high speed attained by PL/C is due more to the combination of good algorithms than to any specific one.

Lexical analysis

The lexical analyzer (LEXI) produces a stream of tokens from the stream of input characters forming the PL/I source program. Thus it must recognize constants, identifiers, operators, and reserved words and condense these into single tokens. The recognition of these is straightforward. The AED[9] approach of building a finite state machine is not followed closely, although the TRT instruction of the IBM 360, which in effect simulates the FSM, is heavily used. The algorithm worth mentioning is that used to hash identifiers into the symbol table. A double random algorithm, which constructs two hash functions, h and g, is used. The ith look into the hash table for an item with key K is made at location $L(i, K)$ where

$$L(i, K) = h(K) + (i-1)g(K) \bmod p,$$

p being the size of the hash table, and $g(K)$ is made relatively prime to p. The hash functions are chosen to minimize the possibility that $h(K_1) = h(K_2)$ and $g(K_1) = g(K_2)$ for $K_1 \neq K_2$.

Syntactic analysis

SYNA accepts the stream of tokens produced by LEXI, recognizes statement boundaries (this task was simplified through the introduction of 18 reserved words, mostly statement keywords), and produces I-code for each statement after checking for syntactic correctness. In addition, SYNA must process DECLARE statements, which accounts for about one half of SYNA's instructions. Errors are corrected, generally by either ignoring or supplying tokens to fit the required syntax. The arithmetic expression analyzer is interesting for its conciseness, and flexibility. It is basically a pushdown automaton, with a finite state machine as a component. The FSM is implemented as a branch table. LEXI supplies a class for each token, CL(t). SYNA simply issues a GOTO TAB (S, CL(t)), where S is the current state. The indicated routine performs some action, which may be an error correction, sets the new state of the machine, and goes back to the main loop. Parentheses and certain other delimiters simply push down the current state and start the FSM again. Thus, an 8 class by 9 state branch table summarizes the processing of expressions, by far the most exercised mechanism in SYNA.

Semantic analysis

Semantic analysis (SEMA) scans the betacode produced by SYNA, searching for expressions. A special betacode token for expression start has already been inserted by SYNA to make the search simple. This token also includes a "semantic requirement code" which specifies the restrictions to be imposed on the expression. A typical restriction might be (SCALAR, STRING) imposed on the Boolean expression in an IF

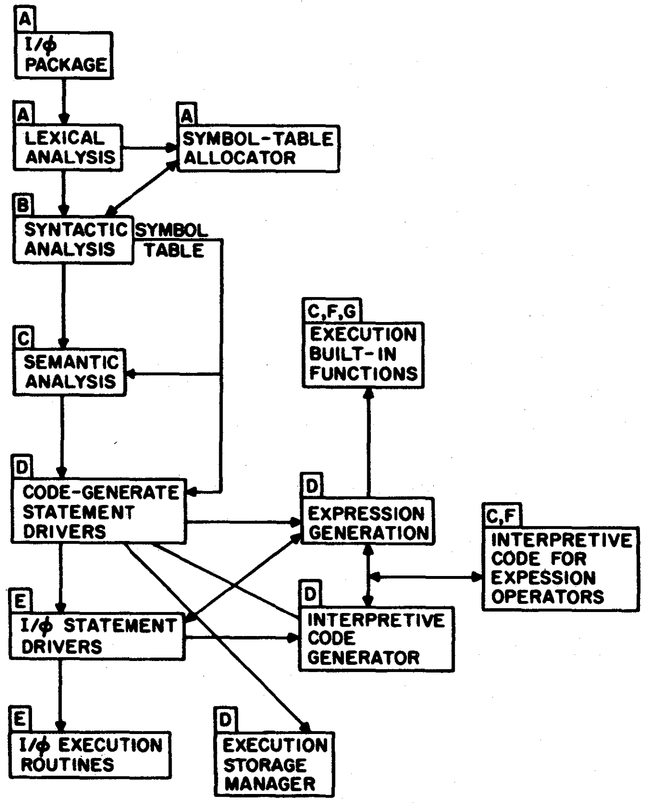


Figure 3—Detailed organization and personnel assignment in PL/C

statement. SEMA proceeds to construct a parse-tree for the expression, in postfix order. Simultaneously, it produces a postfixed, resolved form of the expression, and determines the conversion class (type) and structuring (dimensionality) of each subexpression. Where a conflict between the conversion class (arithmetic, string, label, pointer) of a subexpression and its governing operator arises, SEMA reports an error, and corrects it by modifying the parse tree. If no errors are detected, the postfixed string replaces the original infixed string in betacode, creating gammacode. Note that the postfixed form must be shorter than the infixed string, and thus only expressions change from betacode to gammacode. If errors were detected, a recursive, top-down pass is made over the corrected parse tree to get the new postfixed string. This strategy permits some highly sophisticated error correction to be used; for example, a scheme which will minimize the number of tokens changed has been proposed. Such a scheme requires that information about the entire expression be conveniently available, and the parse tree fills this need. In addition spelling correction for identifiers could be conveniently introduced at this stage.[7]

Code generation

The code generation pass consists of a series of statement drivers, each called into action by a particular statement body token placed in I-code. Each statement is viewed by CGEN as a string of expressions held together by a network of pointer tokens which appear in I-code. The statement drivers follow these pointer chains to generate machine language code to evaluate the imbedded expressions in an appropriate order.

Interpretive coder

An important feature of CGEN is the mechanism by which individual machine language instructions are generated. A macro language to facilitate this task has been developed. This macro language is best viewed as the machine language for a special purpose automaton for generating instructions. The language is reasonably powerful, permitting conditional branching, the setting of flags, and both machine language and interpretive language subroutine calls. In addition, the interpreter keeps near automatic track of the location of each PL/C program variable. The generation of accessing instructions is wholly a function of the interpreter.

To produce the code needed to compute FLOOR(X), where X is REAL FIXED BINARY (p, q), one writes:

GLOAD	G, REAL1	generate load of X into register
GRS	SRA, REAL1, RO, A1	generate SRA X, q to delete the fraction bits of X
GFIN		finish code sequence

Here, the number q has been placed in cell A1 by previous instructions. The GRS macro causes generation of any RS format instruction as follows: GRS opcode, register1, base2, displacement2. The opcode appears literally. The other three items are pointers to various locations during compile time. Thus, REAL1 is a pointer to REAL component of the 1st stack entry. The interpretive coder records in location REAL1 information allowing it to generate code to access the quantity involved. As X is moved (by the GLOAD macro) from storage to a register, the interpretive coder records its new conceptual location in the compile time stack entry REAL1.

Execution

The PL/C runtime environment is mostly quite standard. Thus there is a runtime stack management routine, an I/O package which interfaces to the PL/C operating system interface package, and a collection of mathematical and string manipulation routines. Two execution time interrupts are handled in interesting manners. These are the timeout and uninitialized variable interrupts.

Traditionally, the expiration of a step time limit has been somewhat destructive of the executing program environment. The timeout can occur deep within the evaluation of an expression, and often results in unintelligible diagnostic output. PL/C takes the approach that timeouts should be permitted only between statements, and accomplishes it as follows:

Upon completion of each statement, control is passed by subroutine call to a small (one instruction) subroutine, which is located at the beginning of the common data area always covered by general register 12. The routine usually contains a BCR 15, R5 instruction (unconditional return), and is generally called by a BALR R5, R12. When the time out occurs, the return is replaced by a two instruction sequence which passes control to the appropriate diagnostic routine. The state of the environment is fairly clean, permitting the user to recover more information about the progress of his program than would have been possible if intra-statement timeouts occurred.

The uninitialized variable interrupt is not a hardware interrupt at all. PL/C creates such an interrupt by the following device: Each variable is initialized to the constant x'80000000', which is the smallest possible negative integer on the 360, and whose complement cannot be represented. PL/C generates an LPR instruction before the accessing instructions, which will cause a hardware fixed overflow interrupt to occur if the indicated value is encountered. This constant represents the integer -2^{31}, which requires 32 bits of precision for its representation. Since PL/I imposes a precision limit of 31, the constant -2^{31} cannot arise through fixed-point binary computations without overflow. In fact, in the 360, implementation of PL/C, this constant can occur legitimately only as the representation of a character-string. The possibility of this occurrence is felt to be so rare as not to require further checking.

Service routines

Every compiler includes a gamut of programs which are not central to the operation, but which may be called by many other parts of the compiler. The ensemble of these programs provides an internal environment for the compilation task. In PL/C, these programs include an overall control module, an error message writer (phrase expander) similar to that described for the

PUFFT system,[8] and the "reverse translator," which produces source images from I-code, and is used to indicate corrections which have been made by SYNA and SEMA.

Special assemblers

Several special programs have been developed to create the initial environment for compiling PL/C programs. There are over 250 keywords which must be hashed into the symbol table, for example, and several hundred error messages which must be broken down into phrases.[11] For both of these tasks, special routines have been written which create these tables in the PL/C compile time environment, and then punch out object deck copies of core. These are then linkedited with the rest of the compiler to produce a great savings in initialization time.

DESIGN ACHIEVEMENTS

The actual performance of PL/C has been as good or better than hoped for with respect to most of the seven goals mentioned earlier. The compilation speed seems to be an order of magnitude faster than PL/I-F. On a 360/65 running under OS/MVT, compilation speeds of between 6,000 and 20,000 statements per minute have been observed, depending upon the programs used. A moderate sized sample of random student jobs, with a substantial number of errors yielded the formula compile time $= .04 + .007s$ seconds, where s is the number of statements. This gives an impressive 8500 statements/minute compile speed for programs in which the error correction mechanisms are heavily exercised.

The compiler correctly repairs a useful fraction of the punctuation errors that are endemic with neophytes and which even afflict those programmers with more experience. While its repair of significant syntactic and semantic errors is less often successful in the sense of being able to recreate what the author intended, it is often very successful in prolonging the life of a program sufficiently to yield additional useful diagnostic information. Eight years of experience with this approach have clearly demonstrated a significant reduction in the number of job submissions required to obtain a successful execution of a program. In its current form PL/C probably offers more ambitious and sophisticated diagnostic assistance than any other compiler for any language in general use, and this aspect of the system is quite open ended, with further developments anticipated.

The breadth of subset which has been achieved has surprised even the authors. The only significant omissions from the current version are the compile time facilities, multitasking, I/O other than stream sequential, and list processing. The achievement of open-endedness makes the addition of these possible, and work is proceeding on the list processing area now.

The two factors which were most underestimated by the group were space and implementation time. The goal of running in $100K$ was met through the use of overlays. The memory organization proved robust enough to handle the overlays without requiring any redesign of modules. PL/C will compile and execute small programs in as little as $90K$, and can handle about 250 statement programs in $100K$. A completely core-resident version (no overlays) is obtained by changing one linkedit control card, and runs in $100K$.

The original implementation schedule called for a rough version ready for testing in 6 months, and a production version at the end of 9 months. Outside sources who had implemented PL/I compilers estimated 18 months as a more realistic target for a production version. In fact, the major testing with live student audiences began after about 13 months, and production versions were shipped in 17 months. Thus, the old rule of thumb about multiplying any estimates of programming time by two is again validated.

CONCLUSIONS

A high-performance compiler has been designed and implemented for a subset of PL/I using the strategies outlined in this paper. In retrospect, one of the few major changes in this strategy would have been the early implementation of debugging aids for the compiler writers. While we still feel strongly that assembler language was a wise choice, formatted dumps of I-code, symbol table, and object code might have significantly reduced the debugging time, and helped to meet implementation deadlines.

In our judgment the key factors in the success of the PL/C project were:

1. Overall design which provided clean, clearly specified interfaces between the major compiler phases.
2. Elimination of many internal communication problems through the assignment of personnel. In particular, placing the same person on both sides of an interface is seen as a major factor.
3. Use of new techniques in analysis and code generation.
4. Refinement of "standard" techniques for high-performance compilers. These include limiting the size of programs so that auxiliary storage is

not used during the compilation process, and use of a single pass over the actual source to minimize I/O time.

Confirmation of the fact that high-performance compilers are here to stay has come recently in the form of the announcement of the IBM Checkout PL/I,[6] which is actually an interpreter. It appears from the announcement that PL/C still enjoys a three or four to one performance advantage, but the days of having all users pay the high penalties inherent in using production compilers for debugging are numbered.

ACKNOWLEDGMENTS

The PL/C design is directed by Professor Richard Conway and includes the authors, T. Wilcox, and M. Zelkowitz. Additional design and programming assistance has been provided by M. Bodenstein, H. Cabassa, P. Dormont, R. Fisher, T. Kahne, R. Holt, S. Lisberger, N. Weiderman, K. Wong and W. Worley.

REFERENCES

1 R W CONWAY W L MAXWELL
CORC—The Cornell computing language
Comm ACM 6 Sept 1966 pp 317-321

2 R W CONWAY W L MAXWELL
CUPL—An approach to introductory computing instruction
Technical Report 68-2 Dept of Computer Science
Cornell University

3 R W CONWAY DELFAUSSE MAXWELL WALKER
CLP—The Cornell list processor
Comm ACM 8 April 1965 pp 215-216

4 *IBM 360 assembler language*
IBM manual no C28-6514

5 *IBM 360 PL/I (F) reference manual*
IBM manual no GC28-8201-2

6 *IBM scientific computing report*
Fall 1970

7 H L MORGAN
Spelling correction in systems programs
Comm ACM 13 Feb 1970 pp 90-94

8 S ROSEN R SPURGEON J DONNELLY
PUFFT—The Purdue University fast FORTRAN translator
Comm ACM 8 Nov 1965 pp 661-666

9 D T ROSS JOHNSON PORTER ACKLEY
Automatic generation of efficient lexical processors using finite-state techniques
Comm ACM 11 Dec 1968 pp805-814

10 P SHANTZ GERMAN MITCHELL SHIRLEY ZARNKE
WATFOR—The University of Waterloo FORTRAN IV compiler
Comm ACM 10 Jan 1967 pp 41-44

11 R A WAGNER
Common phrases and minimum space text storage
Technical Report 70-74 Dept of Computer Science
Cornell University

GPL/I—A PL/I extension for computer graphics

by DAVID N. SMITH

Boeing Computer Services Company Inc.
Wichita, Kansas

INTRODUCTION

The tools available to the professional graphics applications programmer are without a doubt in the model T stage. They are often implemented as subroutine packages[1,2] that either provide limited power or place a large burden on the user of the package. In addition, in an industrial environment the cost of training an experienced programmer to become proficient with the package can be very high.

The users of computers have long since become accustomed to dealing with normal input-output in a logical rather than a physical manner, often not knowing or caring what mechanism is employed to read a sequential file or where, at compile time, the file is stored. High level languages have provided the ability to have logically identical files on such diverse units as tapes, disks, card readers and data cell drives.

A graphics terminal is just another input-output device. Granted, it is a relatively new and still glamorous device, but this should not prevent creation of software to allow a programmer to *use* the device rather than to fight cumbersome packages and produce unreadable source code. Several applications oriented, specialized or device dependent languages have been described in the literature,[3,4,5] but none are general.

In an attempt to find a better solution to the problem, a design study was initiated by the author in 1968. This paper will present the status of this study as of September, 1970. Discussed first will be general considerations in the design of a high level graphics language,[6] followed by a short description of the PL/I based language. PL/I was chosen as the base language, rather than ALGOL or FORTRAN because of its built-in interrupt handling structure, its richness of data types and structuring, its built-in input-output (as compared to ALGOL) and its "extendability" (as compared to FORTRAN). A basic knowledge of PL/I is assumed throughout.

GENERAL CONCEPTS

Since almost any graphics application of any significant size will require non-graphical computing support, the language should at least allow access to a general procedural language or be an extension of such a language. Some of the advantages to the "base" language approach are:

A. Provide a syntactic base to guide in the design of the graphics statements.
B. Cut implementation time because less needs to be implemented and because some or most of the implementation can be done in the host language.
C. Take advantage of extensions to and improvements of the base language.
D. Avoid "re-inventing the wheel". A GO TO statement is a GO TO statement whether in ALGOL, FORTRAN, JOVIAL or "GRAPHICSTRAN." (It might differ in flavor and texture but not in meaning.)
E. Cut training time when the base language is one known to the users.

The language should provide general digital graphics facilities that are not limited to any machine or device type. The user should not need to know the specific characteristics of the hardware to write a program. A display should be able to be described in terms that are general to most (if not all) devices, with user knowledge of a specific device limited to knowing which language features are supported. Defaults should be supplied when possible. Thus, if an image in storage contains the information that it is to be displayed as a red image, the image should still be displayable on a device that does not support red images by simply ignoring the option.

The above brings up an important point: That images (or pictures) should be defined in terms of concepts rather than specific hardware commands. A statement like:

LOWER RED PEN;

could be defined for a plotter and a statement like:

MAKE BEAM RED;

could be defined for a CRT device. However, each of these statements is conceptually applying the attribute "color" to an image that is to be displayed. A more general statement might be:

ATTRIBUTE OF (image) IS RED;

where "image" is a picture name. This does not limit the application of the attribute "color" to existing or supported devices but has applicability even to such exotic output media as oil paintings and holograms.

It is a temptation to provide other than basic facilities in a language. This should be avoided unless some overall gain results. As an example, it is far better to provide the ability to create labeled axes in an applications subroutine library written in the language than to build it into the language.

Implied in much of the previous discussion is the idea that the language should support any and all devices that are considered to be graphics devices. This might well include, for example, line printers if print plots are desired as output.

Image data structuring should be built in the language in such a way that the details of the structure are hidden from the user. Many examples of powerful but complex image structuring schemes are described in the literature. They have a common purpose, to provide dynamic data structuring, but suffer from being so complex to use that the average user will never apply them. In addition, direct use of a specific data structure prevents easy structure conversion as a program or its application grows. Thus, image data structures should be included in the language in such a way that the details of the structures are hidden from the user and so that the structural mechanism may be readily changed. More than one structure should be available to allow the user to select which type best fits his needs at the moment, and a "non-structure," which eliminates the need for expensive structural manipulations, should be available.

The "non-structure" or sequential image provides the ability to store pictures as a sequential list or stream of graphics data. The characteristics of devices such as plotters require that the programmer have control over the order of the output.

Two approaches to causing the display of images can be conceived. One is the "describe-it-then-output-it" method and the other is the "it-gets-output-while-I-describe-it" method, or automatic method. The first requires the user to output any and all images explicitly after they are created. The second provides automatic output as the image structure is being manipulated. Both should be available with the automatic method being the default.

With present graphics programming techniques, it is necessary to have the graphics device allocated to the program for all debug runs from the very start of testing. This can place a hardship on the programmer who needs to gain access to an overcrowded graphics terminal, not to mention the increase in costs due to the use of the device and to the occupation of other computer facilities. In most cases, initial debugging of a program or of a change to a program can be planned ahead of time and the session at the terminal is necessary only because it is the only testing method available. An alternate solution is to provide a method of making debug runs without an interactive terminal. This would require a software simulated device with the following characteristics:

A. A trace of input-output operations including the names of all pictures displayed, whether or not lightpen detects are allowed, which functions keys are enabled, etc.
B. An input stream containing simulated user responses to displays and actions to be taken to allow further processing when errors occur.
C. The debug package should react to all operations in the same manner as the device it replaces.
D. No program changes are required. The use of the simultated device should be set by control cards.
E. Simulation of a high cost device might be provided on a low cost terminal.

LANGUAGE DESCRIPTION

Vectors

Vector data and vector operations have been added to GPL/I to allow for ease in geometric calculations and to provide a method of describing language elements that contain coordinate information. Points are described by the components of a vector with its tail at the origin. Scaling information is given by a vector which describes the "distortion" along each of the coordinate axes. Rotation information is given by a vector which describes the "torque" perpendicular to the plane in which rotation occurs.

Vectors may be two-dimensional (VECTOR (2)), which is the default when the number of dimensions is omitted, or they may be three-dimensional (VECTOR

(3)). A vector is described in terms of its components along the coordinate axes. A component constant is a constant of any base, scale or precision followed by an X, Y or Z. All of the following are vector components:

14.7X

2Y

10010BX

0.214159E1Z

A vector may be formed by an expression of these components. Several examples are:

VEC1 = 43.5X + 2.2Z + 1.4X + 2.7Y;

VEC2 = 8X + 2Z;

VEC3 = 8 + 2Z;

The Y component of VEC2 is zero. VEC3 has the same value as VEC2 since a scalar constant is converted to an X component. Vectors may be defined by the VECTOR function and examined by the MAG, XMAG, YMAG and ZMAG functions.

Operators are provided for the vector and scalar products. The dot product of two vectors is:

VEC1*.*VEC2

and the cross product is:

VEC1***VEC2

Addition and subtraction are performed by the standard operators. In the hierarchy of operations, *** is highest and *.* is next highest.

Images and image expressions

The "image" is the foundation of GPL/I. The language would not exist without it, and the bulk of the power of the language is dependent upon the manipulations that may be applied to it. Thus, a careful understanding of the concept and properties of the image is essential.

TABLE I—Image Data

POINTS	7.2X + 3.3Y - 1.5Z
TEXT	"'DEPRESS ANY FUNCTION KEY', 0.14"

FUNCTIONS	ARC(POINT1,POINT2,3.14159)
	COPY(IM)
	INSTANCE(PIC)

TABLE II—Operators

NAME	FORM	
	Char 60	Char 48
Inclusion	+>	INC
Connection	|||	CON
Positioning	@	AT
Scaling	<>	SCL
Rotation	*>	ROT

Defining an image as "an identifier to which the IMAGE attribute has been applied", is correct but provides little information. An image may be better defined as a variable which may take as values some combination of pictorial data, pictorial function values and other images. This is still incomplete but is a good starting definition. The best definition may be obtained by listing the properties of an image as described in this paper.

An image is a new data type which may have any storage class and may be structured. Its value is in two parts, pictorial values and attribute values. Both may be changed at execution. Dynamic attributes are discussed later. The storage for the value of an image is acquired dynamically and is structured as necessary to record relationships between images, their contents and attributes. Although the structuring mechanism may be very complex, it is totally hidden from the user.

Image data consists of points (represented as vectors), text, or values returned from image functions. See Table I for an illustration of each.

Image data and other images, whether defined (given values) or not, may be combined to form a new image by the five image operators in a statement patterned after the arithmetic assignment statement. The operations are: inclusion, connection, positioning, scaling and rotation. Table II summarizes the operators. The priority of graphics operators is lower than all other PL/I operators, as shown in Table III.

The inclusion operator causes the arguments to be included in a resultant image. Thus,

A +> B

results in an image which contains A and B. The connection operator is similar to the inclusion operator except that a line is drawn from the last point of the first argument image to the first point of the second argument image. Assume that P4 and P5 are points,

TABLE III—Operator Priority

OPERATOR	PRIORITY
***	Highest
.	
** ¬ prefix + prefix -	
* /	
infix + infix -	
< ¬< > ¬> <= ¬= >= =	
&	
|	
<> *> @	
||| +>	Lowest

then

P4 ||| P5

are the two points connected by a line.

The positioning operator specifies how an image is to be positioned. All images have a default coordinate system. The positioning operator specifies how the origin of this coordinate system is to be positioned in the coordinate system of the resultant image.

DEAD_MAN

= GUN @ P1 +> SLUG @ P2 +> MAN @ P3;

The origin of the image, GUN, is positioned at point P1 of a new image, SLUG is positioned and included and MAN is positioned and included and the result is assigned to the image DEAD_MAN.

The scaling operator specifies how the image is to be scaled. It is like the positioning operator in that the scaling applies to the resultant image.

LITTLE_BOX = BOX < > (1000.0X + 1000.0Y);

BIG_BOX = BOX < > (.01X + .01Y);

Scaling may be used to create concentric circles, assuming that CIRCLE contains a circle and that CONCIRC has no previous value.

```
DO I = 1 TO 5;
CONCIRC = (CONCIRC + CIRCLE)
                < > (.9X + .9Y);
END;
```

The rotation operator is used like the scaling operator. It provides an amount in radians to rotate an image about the Z axis.*

DEL = DELTA *> 3.14159Z;

Images may be defined in two or three dimensions.

Any legal non-complex arithmetic expression or vector expression may occur in an image expression in place of an image. The expression is evaluated and then converted to vector (if necessary). The vector is considered to be based at the origin. Its components then describe a point which is at the tip of the vector. The vector is thus converted to an image of this point.

Images, when used to define other images, are assigned by name rather than by value. This is one of the more powerful features of the language. Consider three images, PA, PB and PC, that contain the coordinates of the vertices of a triangle. The triangle is formed by the statement:

TRIANGLE = PA ||| PB ||| PC ||| PA;

The last side is drawn by connecting PC with PA. Since PA is an image and is assigned by NAME, this is exactly the same point as the first PA. Thus, the image TRIANGLE not only looks like a closed figure, it is a closed figure. Applications in several fields** either require or are simplified by an image which has the same structure as the thing it represents.

Since images are assigned by name, any change in the value of an image causes a change in any image which

TABLE IV—Examples of Image Assignment Statements

```
1)      A = B +>   C +>   D;
2)      IM = (BB +>   CD)   @ POS(J)  +>   LMN @ XYZ;
3)      EZ = (B @ X +>  C @ Y) <>  SC    *>    ANG;
4)      XX = A ||| B ||| 2.7X + 3.3Y  |||   (0 + 0Y);
5)      A = B ||| C ||| D;
        A = A ||| E;
        /* IS EQUIVALENT TO */
        A = B ||| C ||| D ||| E;
6)      DCL MSG CHAR(13), ERMSG IMAGE, ERPOS VECTOR;
        MSG = 'ERROR IN DATA'
        ERMSG = "MSG" @ ERPOS;
        /* IS EQUIVALENT TO */
        ERMSG = "'ERROR IN DATA'"  @ ERPOS;
```

* Three dimensional images may be rotated about any or all axes.
** Such as electronic circuit design.

contains it. An image may be defined in terms of other images which have no value at the time of assignment, thus allowing definition to be deferred. As an example, consider the concentric circle example given earlier. The nest of circles becomes a nest of squares by:

```
CIRCLE = (0X + 0Y) ||| (1X + 0Y) |||
              (1X + 1Y) ||| (0X + 1Y) ||| (0X + 0Y);
```

The image, CONCIRC, would now be changed as would a display of CONCIRC* on a terminal.

The COPY function may be used to assign images by

```
A = XX;
B = COPY (A + R);
A = YY;
```

The value of B is not altered by the redefinition of A in the third statement.

The effect of the scale, rotation and position operators as well as most image attributes** applies to all images in their scope. For example,

```
DCL A IMAGE REGION(0X + 0Y, 1X + 1Y),
    B IMAGE REGION(-1X-1Y, 0X+0Y);
```

The regions of A and B do not overlap. Thus,

```
B = A;
```

would cause B to be empty. (The reason is that everything outside the region of B is scissored or deleted and all of the region of A is outside the region of B.) Consider also:

```
DCL (AA, B, C) IMAGE SCALE (2X + 2Y);
AA = (0X + 0Y) ||| (1X + 0Y);
B  = AA;
C  = B;
```

AA is a one-inch line scaled by 2X + 2Y (which would be a ½ inch line when displayed). B is AA scaled by 2X + 2Y (which would be a ¼ inch line when displayed). C is B scaled by 2X + 2Y which is AA scaled by 2X + 2Y (which would be a ⅛ inch line when displayed).

Image grouping is obtained by the INSTANCE function. Consider an image named IMX which is made up of a number of other images:

```
IMX = A + >B + >C + >D;
```

A lightpen detect on IMX will be correlated with the most basic image. Thus a detect on the part of IMX corresponding to A would cause the detect to be correlated with A (or a component of A) rather than IMX. However, if IMY is defined as:

```
IMY = INSTANCE (IMX);
```

then a lightpen detect on any part of IMY causes IMY to be correlated.

A number of useful image attributes are available. They are described in detail in an appendix but they include:

BLANKED	INTENSITY	THICKNESS
DASHED	PERSPECTIVE	VIEWPLANE
FLICKER	STYLE	

The images described up to this point have been structured images. A second type of image, the stream image, may be operated upon like a structured image. It is always assigned by value (and thus must be defined before assignment) and lacks the structure which is so useful in lightpen correlations, graphics design, etc. A stream image does have several advantages; among them are savings in storage, savings in time to manipulate images and a direct correlation with sequential devices (plotters, for example). Unless noted otherwise, all images described are structured.

Interrupt management

Interrupts are manipulated by an extension of the standard PL/I interrupt management statements. An interrupt queue is provided for interrupts that are a result of a user's action at a terminal. Such interrupts raise the FUNCTIONKEY, KEYBOARD, LIGHTPEN, and DESIGN conditions. Interrupts that raise the IMAGEINTERRUPT, CHARACTERDISPLAY, and REGIONBOUNDARY conditions do so when the interrupt occurs.

Interrupts are queued as they occur in the queue of the task owning the file on which the interrupt occurs. However, if the condition is disabled, it is ignored, and if enabled for standard system action, the action is performed immediately. The interrupt queue is examined for processing when any of the following occur:

1. When the TAKE or WAIT statements are executed.
2. When the file is closed.
3. When a task is terminated.

The TAKE statement allows interrupts to be processed without entering a wait state. Interrupt conditions may be selected by name or by file, or all may be taken. The

* See the I/O section and the ACTIVE attribute.
** COLOR is a partial exception, for example.

TABLE V—Interrupt Processing Statement Examples

```
1)     TAKE FILE(X) LIGHTPEN FUNCTIONKEY;
2)     TAKE LIGHTPEN;  /*  ALL FILES  */
3)     TAKE FILE(X) NONE;  /*  CLEAR QUEUE FOR FILE X */
4)     TAKE FILE(X1) ALL, FILE(X2)  DESIGN;
5)     ON LIGHTPEN(ADAGE) GO TO LPEN;
6)     ON ENDKEY(TUBE) BEGIN; END;
7)     ON LIGHTPEN(X) IDENT(A,B,C)
         BEGIN;
           GO TO LAB(IDENT);
           LAB(1):
             ---
           LAB(2);
             ---
           LAB(3);
             ---
         END;
```

queue may be selectively or completely cleared. The statement:

TAKE FILE(TUBE) LIGHTPEN;

causes all LIGHTPEN interrupts on file TUBE to be processed. Additional examples are shown in Table V.

Eight graphics conditions are defined. The DESIGN condition is raised when a graphics device signals that a change to the displayed image is complete. The FUNCTIONKEY condition is raised in response to a function or selector key being depressed while the KEYBOARD condition is raised in response to the end of manual data entry. REGIONBOUNDARY is raised upon an attempt to output an image that is bigger than the area (REGION) in which it is contained. IMAGEINTERRUPT is raised by the display of an image created by the built-in function, IMINTR. The LIGHTPEN condition is raised as the result of the detect of an enabled image by a lightpen, and the CHARACTERDISPLAY and IMAGEDISPLAY conditions are raised in response to an attempt to display images on a file that does not support the given attributes.

Most graphics conditions allow the use of the IDENT option* to assist in the correlation of interrupts and the item causing the interrupt. Imagine a device (HAL 3361, for example) with four function selector buttons. The FUNCTIONKEY on statement may be:

ON FUNCTIONKEY(HAL3361)

IDENT('1101'B) GO TO LAB(IDENT);

* KEYBOARD is the exception.

The bit string bits correspond to buttons 1, 2, 3 and 4 and specify that buttons 1, 2 and 4 are to be enabled for interrupts. (The bit string may be a bit string variable and will be referenced by name.) When an interrupt occurs and the on-unit is invoked, the function IDENT returns the button number. The IDENT option may be used to correlate lightpen detects:

ON LIGHTPEN(HAL3361)

IDENT (A,B,C,D,E) ---;

The IDENT list contains image names that are to be enabled for interrupts. The IDENT function returns the sequence number of the image in the list.

File management and input/output

GPL/I defines a fourth file type, GRAPHIC,* and a set of attributes that describe the characteristics of the file. GRAPHIC files may be of three types: DESIGN, DISPLAY, and STORAGE. DESIGN files are defined as files on which an image is displayed, and from which it is possible to change some or all images. (Note that the images need not be changed but that it must be possible to do so.) Most C.R.T. devices fall in this category.

DISPLAY files are defined as files on which an image is displayed but on which no action at the device may directly cause a change in the image. Most plotters fall in this category. A device that can be associated with a DESIGN file may also be associated with a DISPLAY file to eliminate any possible overhead associated with DESIGN files.

STORAGE files are defined as files on which no image is displayed but on which images may be stored for later retrieval.

DISPLAY files may be declared as being HARDCOPY or SOFTCOPY. HARDCOPY files are files on which the output image is permanent; that is, the image cannot be modified once output. Images on SOFTCOPY files may be modified.

Files are opened and closed as in PL/I. Implicit opening is supported. Additional open options are also supported:

STORAGE, DISPLAY, DESIGN

ASSOCIATE (image-list)

PAGESIZE (vector-expression)

HARDCOPY, SOFTCOPY

UPDATE (option)

PAGESIZE specifies the size of the area (or volume) on

* STREAM, RECORD and TRANSIENT are the other three.

the device on which images are to be displayed. UPDATE specifies that the file may accept input/output from ACTIVE images. The other options are described elsewhere.

Output may occur either explicitly by execution of a PUT or ANIMATE statement, or implicitly by a change in an image that has the ACTIVE attribute. The ACTIVE attribute may be applied directly:

```
DECLARE BRACKET ACTIVE(X) IMAGE;
```

(BRACKET is active on file X), indirectly by the ASSOCIATE file option, and may be changed dynamically.

Any change made to an active image is output and any change made at a terminal is input with no further action on the part of the program being necessary.

Explicit output is controlled by the PUT statement:

```
PUT FILE(ABC) LIST(CAR, HOUSE,
  (CLOUDS(I) DO I=1 TO N) );
```

EDIT directed output allows use of a format which specifies overriding attributes.

```
PUT FILE(XXX) EDIT (IM1,IM2)
  (2 G(SCALE(V)) );
```

The G (graphics) format item specifies the scale for the two images. Options are available to sound alarms at the terminal, lock and unlock keyboards, change the page on hardcopy files or clear softcopy files, and position the new page on hardcopy files.

The GET statement is used to retrieve images from storage and design files.

```
GET FILE(ABC) LIST(CAR,HOUSE);
```

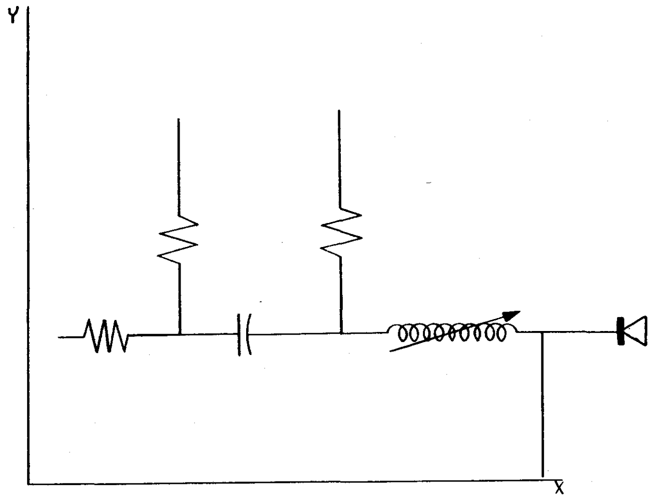


Figure 1—Circuit generated by example in text

The ERASE statement can cause selective or total erasure of the display on a softcopy file.

```
ERASE FILE (XYZ) LIST (CAR, CLOUDS(4));
```

Example of image construction

The example below creates an image of part of an electronic circuit. The components are generated by image functions. The function that generates capacitors is given as an illustration. The generated circuit is given in Figure 1. Each component is assumed to be one inch long. The X and Y axes are given for reference only.

```
DECLARE (NODE1,NODE2,NODE3) VECTOR,
          CIRCUIT IMAGE;
DECLARE
 (RESISTOR,CAPACITOR,VARINDUCT,
  DIODE) RETURNS(IMAGE);
NODE1 = 2X + 2Y;
NODE2 = 3X + 2Y;
NODE3 = 4X + 2Y;
CIRCUIT = (RESISTOR *> 180Z) @ NODE1 +
          (RESISTOR *> 90Z) @ NODE1 +
          CAPACITOR @ NODE1 +
          (RESISTOR *> 90Z) @ NODE2 +
          VARINDUCT @ NODE2 +
          DIODE @ NODE3 +
          NODE3 ||| (4X + 0Y);
CAPACITOR: PROCEDURE RETURNS(IMAGE);
 DECLARE CAP IMAGE;
 CAP = (0.0X+0.0Y) ||| (0.45X+0.0Y) +
       (.045X+0.2Y) ||| (.045X−0.2Y) +
       ARC((1.0X+0.0Y), (0.597X+0.1Y),60.0Z)
       + (0.55X+0.0Y) ||| (1.0X+0.0Y);
 RETURN(INSTANCE(CAP));
END CAPACITOR;
```

Animation

Images may be animated in two basic ways. The first is accomplished by outputting an image that contains a reference to the IMINTR function. IMINTR returns an image that causes the IMAGEINTERRUPT condition to be raised when it is displayed. The on-unit can then change the image and re-display it. The amount of control available using IMINTR is greater than with the second method but can require extensive use of the C.P.U.

The second type of animation is controlled by the ANIMATE statement which may be implemented in part or all by hardware features or software in a peripheral unit or display controller. As an example, consider the following:

```
ANIMATE FILE (X) LIST (CAR)
  (OFFSET (.2X+1.7Y)) EVENT (EV);
```

The image CAR (on file X) is to be moved a distance of .2X+1.7Y inches every second. The process continues until the event variable, EV, is set to completion, the file is closed or the image is erased.
The list of animation options may contain any of the following:

ROTATE	THICKNESS
OFFSET	PERSPECTIVE
SCALE	VIEWPLANE

The argument gives the change per second. It is evaluated only when the animate statment is executed. However, the dynamic option allows the value of the argument to be evaluated each tenth of a second.

```
ANIMATE LIST(A,B,C) (ROTATE(ANG))
  FILE (X) DYNAMIC;
```

Dynamic attributes

Certain image attributes may be changed at execution time. A dynamic change in an attribute is properly a dynamic change in the current VALUE of the attribute. An image always has some color; for example, the current value of the color attribute may be RED at certain times and BLUE at others.

Attributes are applied to images in one of two ways. Functions may be used which return an image which is the argument image with the attribute applied. Thus,

```
COLOR(IMAGEX,'YELLOW')
```

returns an image which is identical to "IMAGEX" but with the color attribute applied with a value of 'YELLOW'.

The second method of altering attribute values is with the attribute assignment statement. Attributes may be assigned to variables declared with the ATTRIBUTE attribute or to the ATTR pseudo variable. For example,

```
ATR = COLOR('BLUE') + FLICKER(X);
```

causes the color and flicker attributes to be assigned to the attribute variable ATR. No other attribute values are set.

```
ATTR(IM) = THICKNESS(I)
  + REGION(X1,X2);
```

The thickness and region attributes of the image IM are changed by the above example.

Other attributes that can be applied to an image in a manner like the above include:

SHIFT

ROTATE

INTENSITY

DASHED

BLANK

ACTIVE

PERSPECTIVE

STYLE

Functions

A full set of graphic, vector and attribute functions are defined in GPL/I and are described in an appendix. Several are of sufficient power to warrant further mention.

The CANONICAL attribute may be applied to any ENTRY identifier with the RETURNS(IMAGE) attribute. It specifies that the function is NOT to be invoked when encountered but that the entry name and parameters are to be retained. The function is invoked when the image is displayed. This prevents functions which return large amounts of data from wasting storage unnecessarily.

The ATTACHER function returns a special image to which lines may be later connected with the CONCAT function. It is useful in network or circuit design.

REFERENCES

1 *Graphic Subroutine Package (GSP)*
IBM Form C27-6932

2 *GPAK—Version II*
IBM Program 360D-3.4.005 September 1966

3 G BRACCHI M SOMALVICO
An interactive software system for computer-aided design—An application to circuit project
CACM September 1970 pp 537-545

4 P G COMBA
A language for 3-dimensional geometric processing
Written Form IBM NY Scientific Center Report
#320-2923 November 1967

5 A HORWITZ J P CITRON J B YEATON
GRAF, Graphic additions to fortran
SJCC 1967 pp 553-557

6 R HORNBY D SMITH J GENTRY C BRYANT
Unpublished results of the graphics language committee
The Boeing Company Wichita Division 1969

APPENDIX A—FUTURE LANGUAGE ADDITIONS

Major Additions

1. New coordinate systems—polar, spherical, cylindrical
2. User defined REGION boundaries
3. Debug feature
4. Define graphics design in more detail
5. HALFTONE, SOLID and LIGHTSOURCE attributes for solid and graytone images.

Research Items

1. Constraints
2. Structure analysis
3. Debug feature
4. New hardware features
5. Hidden line elimination
6. Processing continuous pictures

APPENDIX B—CHARACTER SET

The GPL/I character set is identical to that of PL/I except for the changes listed below.

1. The commercial 'AT' sign, @, is an operator rather than an alphabetic extender character.
2. The following operators are added to the language.

Char 60	Char 48
+>	INC
|||	CON
@	AT
< >	SCL
* >	ROT

3. The double quote (") character is added to the character set.

APPENDIX C—GRAPHIC OPERATORS

Operator	Description
Inclusion	The first and second operands are included in a new image.
+ >	A + > B
Connection	Include the first and second operands in a new image and connect the last point of the first image with the first point of the second.
|||	A ||| B
Positioning	Position the image (first operand) at the point given by the second operand.
@	A @ (2.7X + 3.3Y)
Scaling	Scale the image (first operand) by the factor given by the second operand.
< >	A < > (4.0X + 8.0Y)
Rotation * >	Rotate the image (first operand) by the amount given by the second operand. Rotation is counterclockwise.
	A * > (.172Z)

APPENDIX D—ATTRIBUTES

The attributes that may be altered by functions at execution are flagged with an asterisk.

Attribute	Applies To	Description
*ACTIVE(---) QUIESCENT(---)	Image	ACTIVE specifies that the image is to be automatically output when changed and that the display is to be input when a change is made to it. A list of files on which the image is active may be appended. QUIESCENT is the opposite and is the default. I/O must be explicit.
ASSOCIATE(---)	File	ASSOCIATE specifies images that are to be active on a file. A parenthesized list of images is appended to ASSOCIATE.
ATTRIBUTE	Data	ATTRIBUTE specifies that the variable is to have attributes as its value.
*BLANKED UNBLANKED	Image	BLANKED specifies that the image is invisible. UNBLANKED is the default.
CANONICAL	Entry	CANONICAL specifies that the entry is not to be invoked when encountered but that the entry and its parameters are to be retained as an intermediate result. The entry is invoked when the image containing it is to be displayed. CANONICAL implies REDUCIBLE and RETURNS(IMAGE).
*COLOR(---)	Entry	Color specifies the color of the image. The parenthesized item is an implementation defined color name in quotes. The default is COLOR(''). If a device does not support the given color, the default is used.
*DASHED(---)	Image	DASHED specifies that the image is to be displayed as a dashed line or curve. The parenthesized list is of the form: (up-length, . . .), (down-length, . . .) The curve is drawn one "up-length". Then one "down-length". The next item in each list is selected and the sequence is repeated. A list is restarted if exhausted.
DATAFORM(---)	Data	DATAFORM specifies that the data is to have the same internal form as a reference to the CANONICAL entry name in the parentheses. Assignment may be made to a DATAFORM variable or it may be referenced in an image assignment statement.
DEFINED(---)	Image	DEFINED specifies that the image is to be defined on the value of the base image. Its value is that part (or all) of the value of the base image that is contained in the region of the defined image.
DETECTABLE	Image	DETECTABLE specifies that the image can be detected upon by a lightpen. It is default for all images.
DESIGN DISPLAY STORAGE	File	DESIGN specifies that the file is to be associated with a device that can display and store images in such a way that they can be both accessed by the program and changed by the user at the terminal. DISPLAY specifies that the device can display images but cannot store them and that the image cannot be changed by the user. STORAGE specifies that the device may store images but cannot display them.
*FLICKER(---)	Image	FLICKER specifies that the image is to be made alternately

Attribute	Applies To	Description
		visible and invisible. On and off times are specified in seconds in a list similar to that provided with DASHED.
GRAPHIC	File	GRAPHIC specifies that the file supports graphic input and/or output.
HARDCOPY SOFTCOPY	File	A HARDCOPY graphic file produces permanent copies and thus cannot be changed by program action once output. SOFTCOPY files may be changed by program action. SOFTCOPY is default.
IMAGE(-)	Data	IMAGE specifies that the data is to have a picture (or pictures) as its value. The parenthesized list may be (2) for two dimensional images or (3) for three dimensional images. See text of paper.
*INTENSITY(n)	Image	INTENSITY specifies the relative intensity of the image. The option "n" is a number between between 0 and 9.
*LOCATION(-)	Image	LOCATION specifies the postion of the image in a containing image or display.
*PAGESIZE(V)	File	PAGESIZE specifies the size of the page on a graphic file. The vector "V" describes the page coordinates. On a CRT, the page is the area of the screen that will contain all displays, on a plotter the area to be plotted in until the page is next changed and as needed on other devices to describe the working area.
*PERSPECTIVE(__)	Image	PERSPECTIVE specifies that the 3-D image is to be displayed with a perspective view.
PROTECTED UNPROTECTED	Image	PROTECTED specifiies that the image will be protected from change by a user at a terminal when it is displayed. It is the default. UNPROTECTED specifies that the image may be changed by the user at a terminal.
*REGION(__)	Image	REGION specifies the area or volume in which the image is to have a displayable value. Any parts of the image outside the region boundary are scissored when the image is displayed. The form is: REGION(vector [, vector])
*ROTATE(__)	Image	ROTATE specifies an amount that the image is to be rotated with respect to the containing image or device.
*SCALE(__)	Image	SCALE specifies an amount that the image is to be scaled when displayed.
STREAM STRUCTURE(---)	Image	STRUCTURE specifies that the image is to be structured. See text of paper. STREAM specifies that the image is to be stored as sequential data items. Stream images are always PROTECTED and UNDETECTABLE. See text of paper.
*STYLE(__)	Image	STYLE specifies the form of display of character data in images.
*THICKNESS(n)	Image	THICKNESS specifies the relative thickness of the display of lines in the image. The argument "n" ranges from 0 to 9.
*UPDATE( )	File	UPDATE specifies the status of automatic I/O on a GRAPHIC DESIGN file. The options are (the default) IMPLICIT which allows automatic I/O and EXPLICIT which does not.
VECTOR(n)	Data	VECTOR specifies that the data is to be a 2-D (n=2) or 3-D (n=3) vector.
*VIEWPLANE(---)	Image	VIEWPLANE specifies the size and position of a plane from which views of a 3-D image are to be taken for 2-D display.

APPENDIX E—DESIGN

Design

The DESIGN condition is raised when the device signals that a change in the displayed image (except for keyboard entry) is complete. The FILE and IDENT options may be used. Standard system action is to update the structure in core if the image is active or else to ignore the condition. DESIGN is enabled by default and may be disabled by the NODESIGN label prefix.

Functionkey

The FUNCTIONKEY condition is raised by depressing a function key or selector button. The IDENT option gives the name of a bit string with a length equal to the number of function keys. A one bit indicates that the on-unit is to be executed for that key. If omitted, a bit string of all one bits is assumed. The file option is also allowed. FUNCTIONKEY is enabled by default and may be disabled by the NOFUNCTIONKEY label prefix. Standard system action is to ignore the condition. The bit string is referenced by name.

Example:

```
DECLARE ENABLED-KEYS BIT (32)
    INITIAL ('1111111111111111111111111'B);
ON FUNCTIONKEY (BW2250) IDENT (ENABLED_KEYS)
    BEGIN; ---- END;
```

Keyboard

The KEYBOARD condition is raised whenever the end of data entry is signaled. The IDENT option may not be used. Standard system action is to update the image if the image is active or else to ignore the condition. KEYBOARD is enabled by default and may be disabled by the NOKEYBOARD label prefix.

Regionboundary

The REGIONBOUNDARY condition is raised when an image is being output to a file and a portion of the image exceeds the region boundaries. System action is to delete all parts of the image that exceed the region. The IDENT option is allowed. REGIONBOUNDARY is enabled by default and may not be disabled.

Imageinterrupt

The IMAGEINTERRUPT condition is raised by the display of an image that was created by the IMINTR function. The standard system action is to ignore the condition. The IDENT option may be applied. IMAGEINTERRUPT is enabled by default and may not be disabled.

Lightpen

The LIGHTPEN condition is raised by the detection of a DETECTABLE image by a lightpen. The IDENT option may be applied. LIGHTPEN is enabled by default and may be disabled by the NOLIGHTPEN label prefix. The standard system action is to ignore the condition.

Characterdisplay

The CHARACTERDISPLAY condition is raised when an unprotected image containing text data is to be output to a file that cannot support unprotected text data with the given display characteristics. A standard return from the on-unit causes defaults to be applied in an attempt to display the data. If no on-unit is active, the ERROR condition is raised. The IDENT option may be applied. CHARACTERDISPLAY is enabled by default and may not be disabled.

Imagedisplay

The IMAGEDISPLAY condition is raised when an image is to be output to a device which cannot support certain attributes of the image. A standard return causes an attempt to display the image by changing certain attributes. If this fails, the image is ignored. Standard system action is to raise the error condition. The CHARACTERDISPLAY condition is raised for text items. The IDENT option may be applied. IMAGEDISPLAY is enabled by default and may not be disabled.

APPENDIX F—FUNCTIONS

Vector manipulation built-in functions

XPROD(V_1,V_2)	Vector cross product
DPROD(V_1,V_2)	Vector dot product
ANGLE(V_1,V_2)	Angle between vectors
MAG(V)	Vector magnitude
VECTOR(S_1,S_2[,S_3])	Convert scalars to vector
XMAG(V)	Component magnitudes
YMAG(V)	
ZMAG(V)	

Attribute built-in functions (See attributes in Appendix D)

SCALE	PERSPECTIVE
REGION	ACTIVE
COLOR	FLICKER
BLANKED	INTENSITY
STYLE	DASHED
SHIFT	THICKNESS
ROTATE	PAGESIZE
VIEWPLANE	UPDATE

Condition built-in functions

IDENT	Return number of item in on-unit IDENT option responsible for interrupt.
LPDETCT	Returns the image on which the lightpen detect occurred.

Image built-in functions

INTERSECTION	Returns image that is the intersection of the two argument (3-D) images.
VIEW	Return image that is the view of the argument (3-D) image from its viewplane.
TEXT	Build character images.
ARC	Build arcs and circles.

ATTACHER	Returns special image. See text.
NULLI	Returns a "NULL" image.
CONCAT	Connects images at attachers.
SWEEP	Returns an image (3-D) that is the result of sweeping another image (2-D) along a given path.
IMINTR	Returns special image which causes an interrupt when it is displayed.
INSTANCE	Grouping function. See text.
LPTRACK	Returns the tracking symbol for graphics design.

Special graphic built-in functions

INTERSECT	Returns '1' B if the two argument images intersect.
CHARLEN	Returns the total displayable length of a character string given its style and height.

Image structure built-in functions

COPY (IM)	Copies the structure of IM (remove references by name).
IMAGENØ(A,B)	Returns the sequence number of image B in image A.
INIMAGE(IM,N)	Returns the Nth image that contains the image IM.
SUBELEM(IM,N)	Returns the Nth image that is contained in the image IM.
EXPAND(IM)	Expands all canonical function references in IM.

APPENDIX G—STATEMENTS

Attribute assignment statement

Syntax:

$$\left\{\begin{matrix}\text{attribute-data-element}\\ \text{ATTR(image)}\end{matrix}\right\}\ [,\ldots] = \text{attribute-expression};$$

Where "attribute-expression" is:

$$\left\{\begin{matrix}\text{attribute}\\ \text{attribute-data-element}\\ \text{attribute-expression}\\ \text{(attribute-expression)}\end{matrix}\right\}\left[\left\{\begin{matrix}+\\ -\end{matrix}\right\}\text{attribute-expression}\right]$$

The attribute expression is evaluated left to right. For addition (+ operator), attributes are collected to form a complete set. As additional attributes of the same type are encountered, the new attribute value replaces the existing value. For subtraction, attributes are removed from the set. On assignment to an attribute-data-element all unspecified attributes remain unspecified. On assignment to an image by the ATTR pseudo variable, all specified attributes replace those of the image and the others are unchanged.

Example:

```
    DCL (ALPHA,BETA) IMAGE;

        . . .

    ATTR(ALPHA) = ATTR(BETA) − REGION(BETA) + COLOR('RED');
```

Logical expressions

Images and attributes may be compared to images and attributes, respectively, with the = and ¬ = operators. The value and type of attributes is compared but image comparison is of identity only.

Image assigment statement

Syntax:

image [, . . .] = image-expression;

where "image-expression" is:

$$\left\{\begin{array}{l} \qquad\qquad \text{Vector-Expression} \\ \qquad \text{Image} \\ \text{image-expression} \left\{\begin{array}{l} +> \\ ||| \end{array}\right\} \text{image-expression} \\ \text{image-expression} \left\{\begin{array}{l} *> \\ @ \\ <> \end{array}\right\} \text{vector-expression} \\ \text{(image-expression)} \\ \text{text-item} \end{array}\right\}$$

"vector-expression" is any legal vector or scalar arithmetic term or expression.

$$\text{"text-item" is} \left\{\begin{array}{l} \text{"'character-string' [,size]"} \\ \\ \text{"character-string-identifier [,size]"} \end{array}\right\}$$

See text of paper for details of use.

On statement

Only the modifications to the ON statement are specified in detail:
Syntax:
ON graphics-on-condition(filename) [IDENT(option)] on-unit;
where:

$$\text{"option" is} \left\{\begin{array}{l} \text{bit-string} \\ \\ \text{image-name [,image-name, . . .]} \end{array}\right\}$$

The use of the IDENT option is described in the text of the paper and with the on-condition descriptions.

Take statement

Syntax:

$$\text{TAKE [FILE (filename[, . . .])]} \left\{\begin{array}{l} \text{ALL} \\ \text{NONE} \\ \text{on-condition-name[. . .]} \end{array}\right\} \text{[. . .];}$$

Interrupt(s) in the queue are processed when the TAKE statement is executed.

ALL	Process all interrupts.
NONE	Clear the queue.

"on-condition-name" Process interrupts for the given condition.
The FILE option limits the selection to interrupts from the given file.

Erase statement

Syntax:

ERASE FILE (filename [, . . .]) [(list)];

where

'list' is an image list that has the same syntax as a list in a PUT LIST statement.

ERASE will selectively clear images from a display. If the list is missing, the entire file is erased.

Animate statement

Syntax:

```
ANIMATE FILE (filename) LIST (image-list) (parameters)
    EVENT (event-variable) DYNAMIC;
```

The images in "image-list" are displayed on FILE (filename) and animated as specified by the "parameters". The animation of an image continues until the image is nullified, the image display is erased, the "event-variable" is set to completion or the file is closed. The allowable "parameters" are:

```
ROTATE (change-in-angle)
POSITION (change-in-position)
SCALE (change-in-scale)
THICKNESS (change-in-thickness)
PERSPECTIVE (change-in-perspective)
VIEWPLANE (change-in-viewplane)
```

The arguments to the parameters may be variables or expressions whose value changes each $\frac{1}{10}$ of a second when the DYNAMIC option is specified and the implementation allows dynamic animation parameters for that device. Otherwise the initial value is always used.

ANIMATE may be used on GRAPHIC DESIGN files only.

Signal statement

The SIGNAL statement is extended for GPL/I. Only the extension is described in detail.

Syntax:

```
SIGNAL on-condition [SET (set-values)];
```

where:

"set-values" are references to built-in condition functions.

The SET option defines values to be returned by condition functions that are invoked to determine the reason that the condition was raised. The "set-values" are function names with the values that they are to return in parentheses. Example:

```
SIGNAL LIGHTPEN(FILEX) SET (IDENT(1));
```

Open statement

Additional options are allowed for the open statement:

DESIGN	ASSOCIATE	HARDCOPY
STORAGE	PAGESIZE	SOFTCOPY
DISPLAY		

Put statement

Only the extensions for graphic files are shown.
Syntax:

PUT FILE (filename) [data] [options];

where:

$$\text{'data' is } \left\{\begin{array}{l}\text{LIST (image-list)}\\ \\ \text{EDIT (image-list) (edit-format)}\end{array}\right\}$$

"options" are SIGNAL	sound an alarm at the terminal at the start of output.
$\left\{\begin{array}{l}\text{LOCK}\\ \text{UNLOCK}\end{array}\right\}$	lock or unlock the terminal keyboard before output starts.
PAGE	change the page on the device. This clears the CRT screen prior to output or changes the physical page on HARDCOPY devices.
POSITION(--)	Position the pen, scribe, etc., to the given point before changing the page. POSITION applies to HARDCOPY files only and may be used only with the PAGE option. It overrides automatic page repositioning implied by the files PAGESIZE attribute.

"edit-format" is a format made up of the existing R format item and/or a G format item with the form: "n G (attributes)".
The attributes override those of the output image.

APPENDIX H—EXAMPLE PROGRAM

The program displays a circle and its center point. When the circle is detected upon it gets larger while detection upon the center point causes the circle to shrink. The character string, 'END-PROGRAM' is displayed and initiates program termination when detected upon.

```
/* SAMPLE GRAPHICS LANGUAGE PROGRAM */
SAMPLE: PROCEDURE OPTIONS(MAIN);
S1: DECLARE

        IBM2250 GRAPHIC FILE,

        (CENTER,CIRCLE,MENU) ACTIVE(IBM2250) IMAGE,

        RADIUS FLOAT BIN(21) INIT(2.0),

        LP(3) LABEL;

S2: CENTER = (5.0X+5.0Y);

S3: CIRCLE = ARC((5.0X+5.0Y), VECTOR(RADIUS+5.0, 5.0),360.0);

S4: MENU = 'END PROGRAM',.2" @ (.1X+.1Y);

S5: PUT FILE(IBM2250) SIGNAL;

S6: ON LIGHTPEN IDENT(CIRCLE,CENTER,MENU) BEGIN;

    S7: GO TO LP(IDENT);

        LP(1): /* DETECT ON CIRCLE */
            RADIUS = MAX(RADIUS+0.25, 6.0);
            CIRCLE = ARC((5X+5Y), VECTOR(RADIUS+5 5.0), 360Z);
            GO TO END;
        LP(2): /* DETECT ON CENTER */
             RADIUS = MIN(RADIUS-0.25, 0.5);
             CIRCLE = ARC((5.0X,5.0Y), VECTOR(RADIUS+5.0, 5.0), 360Z);
             GO TO END;

        LP(3): /* END PROGRAM */
             ERASE FILE(BW2250);
             COMPLETION(ENDPROGRAM) = '1'B;
    END: END;

S8: WAIT(ENDPROGRAM);

END SAMPLE;
```

Description of example program

Label	*Discussion*
S1	IBM2250 is declared as a file. The device is an IBM 2250 or its functional equivalent. The images: CENTER, CIRCLE and MENU are declared as ACTIVE on the file IBM2250.
S2	The center of the circle is created as a point at 5.0X and 5.0Y and it is output.
S3	The circle is defined as a 360 degree arc with a 2 inch radius and it is output to the file IBM2250.
S4	The image MENU is given the value of the character string 'END PROGRAM'. It is .2″ high and its lower left corner is at .1X+.1Y.
S5	The console alarm is sounded to indicate the start of the program to the user.
S6	The actions that are to be taken when a lightpen detect occurs are defined. The IDENT option limits the invocation of the ON unit to the three images named.
S7	The IDENT function returns a 1, 2 or 3 corresponding to the image in the IDENT option (S6 above) that was detected upon. A branch is than made to one of the labels LP(1), LP(2) or LP(3).
LP(1)	The radius of the circle is increased by 0.25 inches. Then the image, CIRCLE, is redefined and output to replace the previous displayed circle.
LP(2)	The same action as above is taken except that the radius is decreased by 0.25.
LP(3)	The display is cleared and the event variable, ENDPROGRAM, is set to completion to end the program.
S8	Wait on the end of the program.

ETC—An extendible macro-based compiler

by B. N. DICKMAN

Bell Telephone Laboratories
Whippany, New Jersey

INTRODUCTION

ETC (ExTendible Compiler) is a high level language compiler that allows the programmer to produce very efficient code when necessary, getting as close to the machine as he desires, and yet to write in machine independent statements when the production of optimized code is not necessary. The programmer may also easily extend ETC, creating new data types and operations either from previous extensions or from the machine operations (or both).

Extendible languages come in several forms. One type allows extensions only in terms of a relatively high level and complex base language.[1] Another extendible language may try for a 'minimal' base language[2] which is still machine independent, with the idea that if the base language is minimal, the definitions of the extensions become more comprehensible.

Many extendible languages depend on some macro facility for text substitution, but usually the facility is primitive,[1] rigid,[3] or not implemented. One language even has two separate macro languages.[4] An extendible language could be extendible only in terms of machine independent statements, or it could be extendible 'downward', allowing the programmer to define new operations in terms of machine operations as well.

One test of the downward extendibility of a language is the degree to which the base language was implemented or could easily have been implemented (back to the machine language) using the primitives intended for extension of the language. This is not the same as bootstrapping, since it implies accessibility to the innards of the compiler by the user at any level of extension of the language. Furthermore, if one makes extensions carefully, the programmer who wishes to make changes to the language need not know anything about the implementation of the syntax 'below him' or how the syntax is described.

If one proceeds to extend the machine language using only the assembler, macro facilities, and a judiciously chosen set of primitives, then downward extendibility is ensured.

ETC is such a language. It is an extendible infix language compiler designed to produce very efficient code. ETC gives the programmer the advantages of using a high level language (e.g., relative ease of programming, documentation, and debugging) without the usual drawbacks (e.g., the production of inefficient code). One of the major aims of the language is to do as much 'bookkeeping' for the programmer as possible commensurate with the production of efficient code.

A second goal is to provide polymorphic operators over data types, that is, have the same operator operate on variables of different data types. Thirdly, it is desired to provide a language that can be incrementally implemented with relative ease.

An important bookkeeping feature of ETC is the limited automatic register allocation feature. A way is provided to divide the registers between explicit use by the programmer and implicit use by the compiler, without reserving certain registers for exclusive use by the compiler. Except for subroutine interfaces, there is virtually no need for the programmer to know the number of the hardware register being used. All that is important is the type of register. Thus ETC keeps a record of which registers are free and allocates them as appropriate. For most translators for high level languages implemented for machines with more than one register in which to do computations there is a problem of efficient allocation of the registers. Usually, memory-to-register and register-to-memory machine operations consume much more time than do register-to-register operations. Thus it is advantageous to retain the intermediate results of computations in registers as much as possible. If the number of intermediate results exceeds the number of registers, those results to be used most often should be given priority for remaining in registers.

The key phrase here is 'most often.' Even if a translator can tell which results occur the greatest number of times in a program, the translator cannot tell which results will be used most often in the execution of that program. Even the programmer does not always know which paths will be executed most often. Of course, certain heuristics can be used, but efficient register allocation, and for that matter, generation of efficient code, is not a trivial matter. It is difficult to tell a language translator what space-time trade-off one wants to make. Of course, this does not mean that optimizing translators are impossible.

The solution here has been to give the programmer the ability to produce less-than-optimal code when programming constraints allow programming in a high level language, and yet give the programmer the means to produce code as optimized as he wishes without programming in machine language. The programmer may name and use the thirty-two machine registers just as memory locations, except for subscripting.

If the programmer runs out of registers, the translator outputs an error message and lets the programmer decide which results are most important. A programmer rarely runs out of registers. In practice, the programmer who knows nothing about the registers and does not explicitly use them never runs out of registers.

THE ENVIRONMENT

The SWAP assembler

ETC, the base language and the extended language, exists as a macro library written in the SWAP[5] macro language. While the language extension primitives could be abstracted from the context of the SWAP assembler, many of the problems of other extendible languages do not exist in ETC, because ETC is based on SWAP. SWAP has assembly-time arrays and push-down lists and allows the programmer easily to extend the assembler symbol table. The programmer can store and retrieve macro, symbol, and attribute libraries and can change the input stream while compiling. The macro facilities constitute a powerful set of text substitution functions.

Since ETC is implemented using the SWAP macro facilities, the label, commenting, and continuation conventions of ETC are identical to those of SWAP. The conventions are as follows:

Labels must start in column one. Machine operations, pseudo-operations, macro calls, or ETC statements start anywhere after column one, with one or more blanks or exactly one comma used to delimit arguments. Everything occurring after a "sharp sign", #, is considered commentary. The "commercial at" sign, @, is used to denote continuation.

If @ is the last non-blank character on a card, the next card(s) starting with column one, is concatenated with the character preceding the @.

The machine

The machine for which the language was implemented is similar to the IBM 360. The differences are that there are no memory-to-memory machine operations; the memory-to-register and register-to-memory instructions only move data; the machine has a splendid set of true (immediate) instructions.

The term "labeled common" is used here to refer to read-write memory. Programs reside in read-only memory. Read-write memory is based, word addressed. A labeled common region may not overlap a physical unit of read-write memory, but several labeled common regions may be defined within one physical unit. Read-only memory is not based.

There are sixteen each of A-registers and B-registers. A-registers are used primarily for arithmetic operations; B-registers for indexing, basing, and logical operations.

THE EXTENDED LANGUAGE

General discussion

The application-oriented features will be described only insofar as necessary to give a basis for understanding the base language.

The extended language looks somewhat like FORTRAN except for the following: ETC statements may be freely mixed with pseudo-operations, machine operations, and macro calls; arrays may be only singly subscripted; and there is a facility for symbolically referring to parts of words.

Features include a DECLARE statement; DO and IF statements; limited automatic register allocation; and predefined functions such as FLOAT and SQRT. Other operations include plus, minus, divide, multiply, exponentiation, logical 'or', logical 'and'.

The metalanguage used to describe the syntax has the following conventions. Square brackets denote optional syntactic entities. In a definition the words in lower case or with embedded underscores are to be treated as metasymbols. That is, they represent the form of what is to replace them in an actual statement. All other words and special characters are to be coded as specified.

ETC uses the lexical token break-out of SWAP. This allows one to distinguish between assembly time and run time operations where the same character is used for both, but it means that blanks have syntactic significance. In general, no blanks are allowed on both sides of assembly time operations, but at least one blank must be on either side of an operator generating machine code (operators defined by the operation defining attributes, OPERATION, BINOP and UNOP).

In ETC all registers which are explicitly referenced by the programmer must be declared symbolically. All labeled common regions must be based. This means that the base address of the labeled common region which contains the symbol must be in a register, and the labeled common region name and the register name must be tied together via a USING statement. The ETC BASE statement does all of this for the programmer.

The "type" and "value" of a symbol locate the contents of the symbol in the machine or denote it as a constant. For example, B-register one has type B and value one. Attributes, as specified in the DECLARE statement, may be either locative, as type and value, or descriptive, as signed and floating.

Example of the extended language

Problem: To search the ten word array LIST and put the maximum value in MAX.

Solution 1 (Register independent):

```
MAX = 0
DO I = 0 TO 9
  IF LIST(I) > MAX THEN MAX = LIST(I)
DOEND
```

CONCEPTS

The primitives, constituting the base language, are those features of ETC which are intended expressly for extending the language. They were used to implement ETC and are available to any programmer.

Extensions to the language are always made by the compiler to the "beginning" of the language, so that old definitions of operations, functions, or attributes may be modified, extended, or replaced completely.

Extensions to the language are generally local to the program in which they occur. To make extensions permanent, it is necessary to make them in the program defining ETC.

MAIN CONTROL

One might wonder how the assembler and ETC interact in terms of gaining control from each other. Essentially, if SWAP does not recognize a statement as a prefix operator with arguments, it is passed on to ETC via a redefinition of NONOP (the pseudo-operation called by SWAP for undefined operations) as a macro. This macro is the main driving routine. It does print-formatting and calls the scanner. The scanner is a macro whose definition is extended (always at the beginning) by OPERATION, BINOP (binary operation), UNOP (unary operation), and FUNCTION attribute declarations (and, of course, by any attribute defined by the programmer in terms of these attributes). Actually, BINOP, UNOP, and FUNCTION are all defined in terms of the OPERATION attribute. BINOP, UNOP, and FUNCTION also output information for use by the general expression parser, which OPERATION does not. Syntax determination of whether to call the parser was itself done as an OPERATION declaration.

The default on the data type of the result of an operation or function evaluation is the data type of the first operand or argument. Explicit specification to the parser of the resultant data type may be made in the implementation of the operation or function by an appropriate declaration of the ETC system symbol SYS_R_TYPE.

Defining operations

Several attributes may be used in a DECLARE statement to define operations.

OPERATION is the general operation defining attribute. Its form is

DCL operationname OPERATION(booleanexpression)

where

operationname is the name of the macro to be called if *booleanexpression* is true. The boolean expression may combine as much syntax and semantic analysis as the programmer desires. *booleanexpression* can be any SWAP macro language boolean expression but will usually be in terms of SYSLST(1) to SYSLST(m), where SYSLST(i) is the i*th* token in the source language statement.

OPERATION is generally used to define operations which are not binary or unary operations, to extend the definition of an existing operation to handle new data types, to improve the code generated from an existing definition, and to correct bugs in the code generation for an existing operation.

For instance, one might have the '+' operation implemented for fixed and floating point quantities and want to extend '+' to handle matrices. One would use the ATTRIBUTE attribute to introduce a MATRIX attribute and then specify an OPERATION declaration which detects not only the syntax involved in the addition operation, but also the fact that *matrices* are being added. Then a macro is defined to do code generation for this specialized form of '+'.

Also, a programmer might discover a 'trick' by means of which a specific combination of operands and operations may be made to produce more efficient code than for the general case. He might specify an OPERATION declaration to detect this special case and write a macro to generate the code. No knowledge about the implementation of the operation, other than that of the specific case detected, would be required.

Finally, an OPERATION declaration may be used to detect syntax and semantics of the special case of an operation for which the code generation is incorrect and to extend the compiler to generate the correct code.

The UNOP and BINOP attributes are used specifically to define unary and binary operations.

Used in conjunction with UNOP and BINOP is the PRECEDENCE attribute. Its argument is the relative or absolute precedence to be given the operator. If not explicitly specified, the precedence of an operator defaults to the currently highest precedence plus one.

The RIGHT__TO__LEFT__SCAN attribute is used in conjunction with the BINOP attribute to indicate that the direction of scan of operators of equal precedence is to be from right to left. Default scan for BINOP is from left to right.

Defining functions

The form for the function attribute is

```
DCL name FUNCTION[(macroname)]
```

where *name* is the name of the function and also the name of the macro to be called for code generation, unless *macroname* is specified. The code generation macro is passed an argument list of form

```
variable = functionname(arguments)
```

and must perform the assignment to *variable* of the result of evaluating the function.

Example: A SIN function which calls a SIN subroutine expecting its argument and returning its result in A-register 0 might be coded as follows:

```
 DCL SIN FUNCTION
 MACRO
 SIN
# SYSLST(3,1) IS THE ARGUMENT OF THE THIRD TOKEN, I.E.,
# THE ARGUMENT OF SIN.
 A0 = SYSLST(3,1)
 CALL SIN
# SYSLST(1) IS THE FIRST TOKEN, I.E., THE VARIABLE ON
# THE LEFT HAND SIDE OF THE EQUAL SIGN.
 SYSLST(1) = A0
 MEND
```

In general SYSLST(i) is a way of referencing parameters to a macro call. SYSLST(i) is local to the macro in which it appears. (Nested macro calls "push" SYSLST.)

The function may now be used in any expression.

For example:

```
X = Y + SIN(Z(l)) * 2.0
```

Compile Time Bookkeeping
Attributes of Symbols
Symbol Table Equivalence: EQV

Whereas the SET pseudo-operation will set only the value and type of one symbol to that of another, EQV may be used to copy the whole symbol table entry. This is especially useful when implementing an operation recursively, or when bootstrapping. Often one wants to give a register variable the same attributes as a memory variable.

Or suppose one has implemented an IF statement in a straightforward manner. Consider the comparison to less-than-zero of a signed variable fewer than 32 bits long whose high order bit occupies the high order bit of a word. (In ETC variables are allowed to be parts of words.) Our present implementation would probably unpack this variable from the word, propagate the sign, compare it to zero, and branch accordingly. In most twos complement machines it would be simpler to test the whole word as being negative and branch accordingly without unpacking the variable from the word.

We might extend IF to detect this particular comparison; use EQV to define a new variable the same as the old, except make the new variable a full word variable; and recurse, expanding the IF macro with a 'less than zero' comparison.

The syntax of EQV is as follows:

sym1 EQV sym2

where *sym1* and *sym2* are symbols, will set the value, type, and all attributes of *sym1* to those of *sym2*. If *sym2* is a constant, defaults for the attributes are used. Particular attributes of *sym2* may be overridden by the use of a number of optional keyword parameters:

VALUE=n	sets the value of *sym1* to *n*.
STYP=char	sets the type of *sym1* to *char*.
SIZE=n	sets the size of *sym1* to n
DISP=n	set the displacement of *sym1* to *n*.
SUF=n	sets *sym1* to be signed (SUF=1) or unsigned (SUF=0).
FIXED=n	indicates that *sym1* is fixed if *n* is 1, not fixed if *n* is 0.
FLOAT=n	indicates that *sym1* is floating if *n* is 1 not floating if *n* is 0.

Defining attributes

New attributes may be defined by use of the ATTRIBUTE attribute. The form of ATTRIBUTE is

DCL name ATTRIBUTE[(macroname)]

where *name* is the name of the attribute and also the name of the macro to be called by DECLARE for processing the attribute, unless *macroname* is specified. The attribute processing macro is passed its arguments with the variable name starting in column one, and with the contents of the attribute subscript, if any, passed as argument list.

Example:

The following example, the implementation of the FUNCTION attribute, illustrates the use of the ATTRIBUTE attribute:

```
  DCL FUNCTION ATTRIBUTE(SYSDFNFN)
  MACRO
FN__NAME SYSDFNFN MACRO__NAME
  DCL IS('MACRO__NAME'¬=,MACRO__NAME)IFNOT(FN__NAME) OPERATION@
(I.N.I.SYSLST=3&'I.SYSLST(2)'='='&'I.O.I.SYSLST(3)'='FN__NAME')
  MEND
```

Explanation: If the optional macro name (MACRO__NAME) is supplied, it is to be used as the name of the code generation macro, otherwise, the function name is assumed to be the name of the code generation macro. In any case, FUNCTION is defined as an operation with exactly three tokens, where the second token is '=', and the 'operand' part of the third token is the name of the function. (The 'operand' part consists of everything in the token up to a left parenthesis.)

The argument of the DEFAULTS attribute is the name of a macro to be called to handle defaults. When used in conjunction with the ATTRIBUTE attribute, DEFAULTS associates the attribute with the name of a macro to be called by DECLARE after the last explicit attribute has been processed. The macro to be called is the one associated with the last explicitly specified attribute. Usually, the same macro will be specified for each attribute in a set of consistent attributes.

Register allocation and deallocation

ACTIVATE and DEACTIVATE are the problem programmer oriented register allocation pseudo-operations. ALLOC and DEALLOC are primitives essentially for allocating and deallocating "rock bottom" scratch registers. There is limited automatic register allocation. No "flow analysis" is done, so that the automatic register allocation mechanism generally is valid only for straight line segments of code i.e., where the only entry points or exit points are at the beginning or end of the segment.

Three attributes are necessary for data declaration purposes: REG where the specific numerical register designation is left to the automatic register allocation mechanism, and AQU and BQU for static allocation of registers.

The syntax for the register allocation attributes is as follows:

```
DCL name AQU(number)
DCL name BQU(number)
DCL name REG(type)
```

The third form would declare *name* to be a particular type (A or B) of register. It would *not* allocate a register. The first two forms specify both type and number of register.

The form for the allocation pseudo-operations is:

```
ACTIVATE list
DEACTIVEATE list
```

The elements of *list* are register names, that is, they have been declared to have the AQU, BQU, or REG attribute. If the register to which the symbol refers previously has been statically allocated (i.e., through an AQU or BQU attribute), no allocation is done; the automatic allocation mechanism is merely notified that this register is free (DEACTIVATE) or that it is in use (ACTIVATE). The abbreviations ACT and DEACT are allowed.

If the register to which the symbol refers has been statically allocated and is being deactivated, the value of the symbol is "taken away". If the symbol has no value and is being activated, it is given a value by the automatic register allocation mechanism.

If the symbol has been declared using REG(A), ACTIVATE finds a free register, gives its value and type to the symbol, and marks the appropriate register busy. In this context a register is considered free if it has not been designated as an argument to AQU and is not marked busy.

If the symbol has been declared using AQU(num) or BQU(num), ACTIVATE gives the symbol the value *num*, the appropriate type, and marks the appropriate register busy.

DEACTIVATE undefines its arguments and erases the busy mark for the appropriate register.

Uses of ACTIVATE and DEACTIVATE include automatic allocation of registers to scratch variables and the means to tell when a register no longer contains a particular variable (just as the IBM 360 Assembler DROP avoids the problem of using a register that on longer contains a base address).

AQU and BQU do not define the symbol in the DCL statement. The symbol will be undefined until an ACTIVATE statement is processed at which time it will be defined with the value designated with the AQU or BQU attribute.

Here is the previous example of the extended language, coded using registers to avoid generating redundent fetches and stores:

```
DCL RMAX REG(A)
ACTIVATE RMAX #ALLOCATE A REGISTER
RMAX = 0
DO I = 0 TO 9
  IF LIST(I) > RMAX THEN RMAX = LIST(I)
DOEND
MAX = RMAX
```

There is an inverse, IQU, for AQU and BQU. This pseudo-operation will change the designated symbol from static allocation to automatic allocation. That is,

```
DCL  VAR  AQU(3)
     IQU  VAR
```

is equivalent to

```
DCL  VAR  REG(A)
```

That is, registers in use by the programmer are explicitly marked via ACTIVATE and DEACTIVATE, and all others are available for temporary use (within a statement) by the compiler via ALLOC and DEALLOC.

ALLOC finds a free register, gives its value to the symbol, and marks the register busy. In this context a register is free if it is not marked busy (whether statically allocated or not). ALLOC assumes that its argument has a register type defined before ALLOC is called. ALLOC may be used in preference to ACTIVATE when defining an operation, since one is assured that no ACTIVATE of statically allocated registers will occur during the compilation of one statement.

DEALLOC is the inverse of ALLOC. It is similar to DEACT except that it does not "undefine" (i.e., push) its argument. It only erases the busy mark for the appropriate register.

The philosophy of register allocation

We make it necessary for the programmer to activate each register explicitly for several reasons: (1) At any point in the program one can easily see what registers are in use; (2) a statement is automatically flagged if a register not containing an expected value is used; (3) while the register is inactive, ETC can use it internally as a scratch register for one ETC statement; (4) subroutine interfacing, via passing arguments in registers, is made easier.

Notice that the use of many statically allocated registers nullifies the usefulness of the automatic allocation feature. No register that has been statically allocated (i.e., through use of AQU or BQU) can also be dynamically allocated (i.e., through use of REG and ACTIVATE). This is true for the same reason that *memory* that has been statically allocated cannot also be dynamically allocated. It is advantageous to leave the specific register undesignated by using REG wherever possible.

The capability of automatic register allocation buys several advantages, but it also has one penalty: if it is to work effectively, it must have registers to allocate. Thus sparing use must be made of the static allocation attribute.

Generating diagnostics: DIAG

DIAG is a ETC primitive which is used to generate the diagnostics printed by ETC. DIAG will check the diagnostic level then in effect and, if the level of the diagnostic to be printed is not less than this diagnostic level, the diagnostic will be printed. DIAG also allows the user to prefix and suffix his message with standard phrases. Output of error flags is accomplished by specifying the flag in column one of the DIAG call. A typical DIAG call is

```
DIAG LEVNO=4,PREFIX=8,SUFFIX=1,SUBTRACTION
```

which will cause the following diagnostics to be printed if the diagnostic level is less than or equal to 4.

```
****MIXED MODE SUBTRACTION NOT IMPLEMENTED
```

The problem programmer may use the pseudo-operation DIAGNOSTIC__LEVEL to specify that no diagnostic less serious than that of a specified level be printed.

USING THE PRIMITIVES

An inner product operator

Suppose one wanted to define the inner product of two three-dimensional vectors:

```
V3 = V1 . V2
```

where V1 and V2 are each the first word of blocks of three words. One could write

```
  DCL . BINOP(DOT)
    MACRO
    DOT   V3 EQ V1 D V2
  V3 = V1:(0) * V2:(0) + V1:(1) * V2:(1) + V1:(2) * V2:(2)
#THE COLONS ABOVE INDICATE CONCATENATION. WITHOUT THE COLONS,
#THE SUBSCRIPTED SYMBOLS WOULD BE TAKEN TO BE SUBPARAMETER
#REFERENCING.
    MEND
```

A square root function

```
#THIS IS THE DEFINITION OF A FUNCTION TO TAKE THE SQUARE
# ROOT OF A VARIABLE

  DCL SQRTF FUNCTION
  MACRO
  SQRTF X EQ Y

#CALL A SUBROUTINE TO EVALUATE THE SQUARE ROOT
#ASSUME IT TAKES TWO ARGUMENTS: LOCATION TO STORE RESULT
# AND INPUT VALUE.
```

```
  SQRT__SUB X,Y(1)
  MEND
#NOW SUPPOSE WE HAVE A MACHINE
#OPERATION, FSQRT, WHICH TAKES TWO A-REGISTERS AS ARGUMENTS
# BUT WORKS ONLY FOR FLOATING VARIABLES.
  MACRO
  SQRTF X EQ Y
#IF FIXED, CALL THE OLD SUBROUTINE
IS(FIXED(Y(1)), SQRT__SUB X,Y(1) ; JUMP .OUT)
#IF X AND Y(1) ARE BOTH A-REGISTERS, GENERATE THE MACHINE OPERATION
IS('MTYP(X)'='A'='MTYP(Y(1))', FSQRT X,Y(1) ; JUMP .OUT)
#IF X IS AN A-REGISTER, USE IT AS A SCRATCH REGISTER AND RECURSE
IS('MTYP(X)'='A', X = Y(1) ; X = SQRTF(X) ; JUMP .OUT)
#OTHERWISE, ALLOCATE A SCRATCH REGISTER AND RECURSE
TMP EQV 0,STYP=A ; ALLOC TMP
  TMP = SQRTF(Y(1))
  X = TMP
  DEALLOC TMP
  MEND
```

Complex arithmetic

We will define the attribute COMPLEX and extend the operations "+" and "−" to handle unsubscripted floating variables of type COMPLEX. The functions REALPART and IMAGPART will also be defined.

```
#THIS STATEMENT DEFINES THE ATTRIBUTE "COMPLEX"
#SYNTATICALLY. IT SAYS "CALL THE MACRO "CMPLX"
#TO EVALUATE THE ATTRIBUTE "COMPLEX"".
  DCL COMPLEX ATTRIBUTE(CMPLX)
#THIS MACRO PROVIDES THE SEMANTICS FOR THE "COMPLEX" ATTRIBUTE.
  MACRO
VAR COMPLX
  DCL VAR BSS(2) FLOAT
#THE FOLLOWING STATEMENT MAKES AN ENTRY INTO THE SYMBOL TABLE
#DEFINING THE "COMPLEX" ATTRIBUTE OF VAR TO HAVE VALUE ONE.
#THIS KIND OF STATEMENT ALSO DEFINES THE ATTRIBUTE
#FOR ALL SYMBOLS IN THE SYMBOL TABLE. UP TO 255
#ATTRIBUTES ARE POSSIBLE (WITHOUT PACKING).
COMPLEX(VAR) SET 1
  MEND
#THIS STATEMENT EXTENDS THE COMPILER TO RECOGNIZE THE
#ADDITION OR SUBTRACTION OF TWO COMPLEX VARIABLES.
  DCL CMPLXOP OPERATION(N.SYSLST=5&ANY("SYSLST(4)","+","−")@
&COMPLEX(SYSLST(1))=COMPLEX(SYSLST(3))=COMPLEX(SYSLST(5))=1)
#THIS MACRO DEFINES THE SEMANTICS OF ADDITION AND SUBTRACTION
#OF COMPLEX VARIABLES.
  MACRO
  CMPLXOP X EQ Y OP Z
```

```
#THESE THREE COMPILE TIME STATEMENTS DEFINE THREE REAL
#VARIABLES OCCUPYING THE FIRST WORDS OF X, Y AND Z
TMPX EQV X,COMPLEX=0;TMPY EQV Y,COMPLEX=0;TMPZ EQV Z,COMPLEX=0
#
  TMPX = TMPY OP TMPZ
#THESE THREE STATEMENTS DEFINE THREE REAL VARIABLES (REDEFINE
#TMPX, TMPY AND TMPZ) OCCUPYING THE SECOND WORDS OF X, Y AND Z.
TMPX EQV TMPX,VALUE=TMPX+1
TMPY EQV TMPY,VALUE=TMPY+1
TMPZ EQV TMPZ,VALUE=TMPX+1
  TMPX = TMPY OP TMPZ
  MEND
  DCL REALPART FUNCTION
  DCL IMAGPART FUNCTION
  MACRO
  REALPART A EQ FNCALL
#FNCALL(1) IS THE FIRST SUBARGUMENT OF FNCALL. HERE FNCALL(1)
#IS THE ARGUMENT OF THE FUNCTION REALPART, SINCE THE REALPART MACRO
#IS FED THE FUNCTION CALL AS PART OF AN ASSIGNMENT STATEMENT.
TMPX EQV FNCALL(1),COMPLEX=0
  A = TMPX
  MEND
  MACRO
  IMAGPART A EQ FNCALL
TMPX EQV FNCALL(1),COMPLEX=0,VALUE=FNCALL(1)+1
  A = TMPX
  MEND
  DCL CMPLXASSIGN OPERATION(N.SYSLST=3&COMPLEX(SYSLST(1))=1&@
COMPLEX(SYSLST(3))=1)
  MACRO
  CMPLXASSIGN X EQ Y
TMPX EQV A,COMPLEX=0
TMPY EQV Y,COMPLEX=0
  TMPX = TMPY
TMPX EQV TMPX,VALUE=TMPX+1
TMPY EQV TMPY,VALUE=TMPY+1
  TMPX = TMPY
  MEND
```

DISCUSSION

Downward extendibility probably limits the resultant language to the same structure as that of the assembly language (if the assembler does not have block structure, it will be very difficult to graft this onto the extended language). But perhaps this is a relationship shared by any base language and its extensions.

Code optimization on a global (i.e., more than one statement) scale is very difficult, if not impossible. However, on a statement-by-statement level one can produce very efficient code relatively easily. Allowing the programmer access to machine registers and machine operations facilitates further optimization.

In test cases coded both in ETC and machine language, ETC generated less than ten percent more code than the machine language programs with substantially less coding time for the ETC programs.

The extendibility of the language has proved to be especially useful on a large project with stringent programming schedules where it was not clear in advance what features the particular language should have. Programming began in assembly language and parts of the extended language were used as implemented.

Finally, extendibility makes maintenance of the compiler simpler. It is not necessary to know anything about the implementation of an operation in order to correct a bug in code generation. The language may easily be extended to detect the particular case for which wrong code is generated and to generate the correct code.

ACKNOWLEDGMENTS

Mr. C. B. Hedrick, Miss V. L. Bishop, Mr. D. J. Desmond, Mrs. I. T. Pelak, and Mr. D. J. Sidor have contributed to the implementation of the language. The author would like to thank Messrs. D. Bzowy, R. R. Conners, P. D. Jensen, and J. S. Walden for many helpful discussions.

REFERENCES

1 J NICHOLLS
PL/1 compile time extensibility
SIGPLAN Notices Boston Mass Vol 4 No 8 pp 40–44 1969
2 B J MAILLOUX J E L PECK
Algol 68 as an extendible language
SIGPLAN Notices Boston Mass Vol 4 No 8 pp 9–13 1969
3 W M WAITE
A Language independent macro processor
Comm ACM Vol 10 No 9 pp 433–440 1967
4 A L SPRINGER
Extensible languages
Notes for Advanced Topics in Systems Programming University of Michigan Ann Arbor Michigan 1970
5 M E BARTON
The macro assembler, SWAP
Proceedings of the Fall Joint Computer Conference AFIPS Press Montvalle New Jersey Volume 37 pp 1–8 1970

A file organization method using multiple keys

by MICHAEL L. O'CONNELL

Sanders Associates, Inc.
Nashua, New Hampshire

BACKGROUND

The nature of mass storage hardware is such that each data record stored in a given file is given a unique identifier. That identifier, or key, may be the physical address of the record, either absolute or relative to some fixed point, or it may be a key which is made up of some combination of the record's characteristics, such as the contents of a specific field within the record. Thus, in a personnel file, the record of "John Smith" may be given the key of "19" (if it is the nineteenth record in the file), or "01230402" (if it is on device 01, cylinder 23, track 04, the second record), or "15216" (if John Smith's employee number is 15216). Regardless of the method used to assign the key, however, all keys for records in any one file must be unique, i.e., no two records may possess the same key value. The reason for this restriction is obvious: two records cannot share the same physical address because two records cannot share the same physical space, and two records sharing the same key would lead to ambiguity, i.e., a command containing the non-unique key would not be sufficient to enable the hardware to determine which of the several records containing that key was to be accessed.

Current hardware, then, requires that each data record be identified by one and only one key, whose value is unique to that file, even though that key may be comprised of more than one logical data field.

USER REQUIREMENTS

The user of data records in a mass storage file is frequently hampered by such key restrictions. This is especially true with on-line applications in which many users access the same records for different information.

For example, an accounts payable record may be needed by one user who knows only the vendor number, by another who knows only the vendor's name, and by still another who knows only the product purchased from the vendor. Hardware requires that a specific piece of data (the key) be used in accessing the record. In this example, vendor number would probably be the key, and the two users who don't know the vendor number would be required to look it up manually before they can request the record. And, of course, if the information needed is a list of vendors who had supplied a specific product, it would be impossible to get without looking at all records in the file, because, even if product were the key, a given product might appear in more than a single record, leading to duplicate keys.

The user needs to be able to access a record if he knows any pertinent information about that record, and he needs to be able to access all records which contain a common value.[1] A software system which permits both these functions has been implemented. Named SADI* (Sanders Associates **D**irect **I**ndexing), the system is used in both on-line and batch environments.

GENERAL APPROACH

SADI forms a software interface between the user, with his multiple and non-unique keys, and the hardware/operating system, with its single and unique keys (Figure 1). It accepts a key from the user and selects one or more data records containing that key, which are then passed to the user serially.

Because of the high activity and number of keys involved, the data records are stored in a file separate from the keys. This approach permits key searching to proceed independent of data accessing. In addition, since the keys are usually significantly shorter than the data records, the directory file consisting of all the keys can be searched much more efficiently than could a combined file. A data file and its associated directory file are known collectively as a "user file".

* T.M., Sanders Associates, Inc.

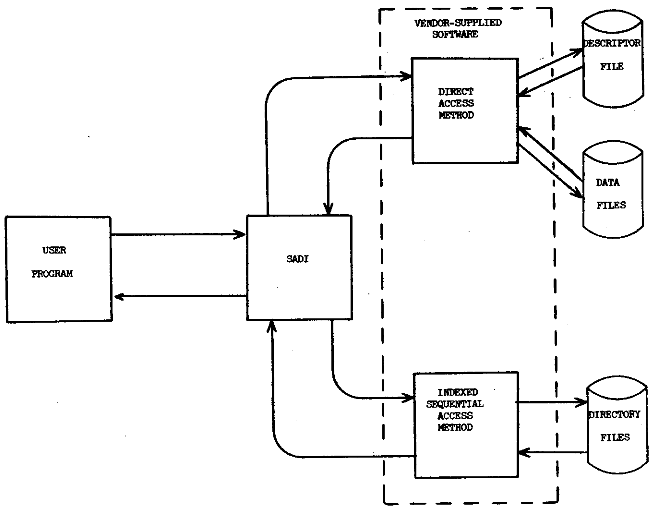


Figure 1—System configuration

A third file containing a description of the data and keys is also created and used by SADI. The descriptor file is used primarily during the creation of data records to identify and extract keys from the data for inclusion in the directory file.

THE DESCRIPTOR FILE

There is only one descriptor file in the entire system, regardless of the number of user files. Each record in the descriptor file describes one user file (Figure 2).

The descriptor file has a direct organization, and each descriptor record's key as known to the hardware/operating system is the directory file name which appears in that record.

In order to create a new user file, the application programmer supplies the information required for the descriptor record (file names, key locations, and characteristics), and any data records that are to exist in the original file. SADI then creates the descriptor record and uses it and the user data to build the directory file and the data file.

The "mode" of each key field is specified in order to convert the user's request key, which is alphanumeric, into the proper format (packed, binary, floating point, etc.) for the directory searching operation.

THE DIRECTORY FILE

The directory file has an indexed sequential organization, i.e., records can be accessed at random or sequentially. There is one directory file per data file. There is one directory record (Figure 3) for each key which appears in each data record. Thus, a data file consisting of 1,000 records, each with five keys, requires a direc-quence by data 5,000 records. The directory file is in se-tory file of key.

The length of the data key field is fixed in any given directory file, but different directory files may have different length data key fields. The length of the data key field is equal to the length of the largest key in the corresponding data file. Data keys shorter than the maximum are left-justified in the directory file and filled to the right with null values. This insures the proper sequence in the directory file.

The key number indicates which key field in the data record contains the value in the associated data key field. The programmer must assign a key number to each key field in the data record. Keys numbered from one through ten are called "secondary" keys, and any key numbered zero is called the "primary" key. It is not necessary to assign a primary key, but there are significant advantages to be gained from its use, as discussed later.

The third field in each directory record is the relative physical location of the data record (in the data file) which contains this particular data key.

The key by which a given directory record is known to the hardware/operating system is the entire directory record. Even if two data records contain the same key value and key number, their different physical addresses assure a unique directory record. Similiarly, if one data record contains the same value in two different key fields, uniqueness in the directory file is assured by different key numbers assigned to the two key fields.

THE DATA FILE

The data file is the simplest of the three SADI files (Figure 4). It has a direct organization, and contains all the user's data. The key of each data record is the record's physical location relative to the beginning of the file, assuring a unique key for each record. The keys shown in Figure 4 are the keys known to the user and which appear in the directory file; they are not the keys by which the data record is known to the hardware/operating system.

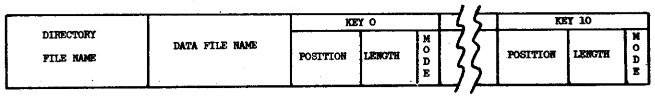


Figure 2—A descriptor record

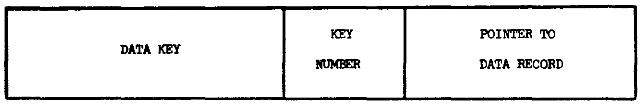


Figure 3—A directory record

The data file can be accessed only by first reading a directory record, and obtaining from it the pointer to the appropriate data record.

CREATING A USER FILE

Creating a user file is a three step process:

1. The descriptor record is created and written into the system's descriptor file.
2. The user's data is read sequentially and written into the data file. As each data record is processed, information in the descriptor record is used to extract key information from the data record, build a directory record, and write it in a work file.
3. The work file is sorted into sequence by data key and written into the directory file. The work file is then scratched.

The relationship between the files and records is shown in the example of a personnel file without a primary key in Figure 5. There are three data records with three keys each; there are nine directory records in the directory file.

ACCESSING A DATA RECORD

The entire system of files is known to the user only by the name of the directory file. When the user names the file he requires with the command FILE EMPLOYEE, SADI brings into memory the corresponding record from the descriptor file. The directory file and data file are then located in preparation for a user request.

When the user requests access to a data record with the command SHOW JONES, the directory file EMPLOYEE is accessed and the record with the key of JONES is brought into memory. (Partial, or generic,

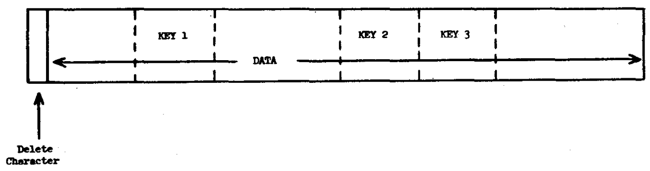
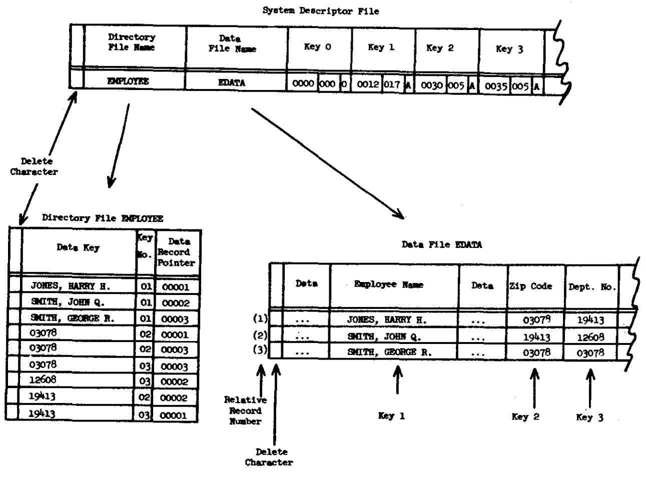


Figure 4—A data record



Directory File Name	Data File Name	Key 0			Key 1			Key 2			Key 3		
EMPLOYEE	EDATA	0000	000	0	0012	017	A	0030	005	A	0035	005	A

Data Key	Key No.	Data Record Pointer
JONES, HARRY H.	01	00001
SMITH, JOHN Q.	01	00002
SMITH, GEORGE R.	01	00003
03078	02	00001
03078	02	00003
03078	03	00003
12608	03	00002
19413	02	00002
19413	03	00001

	Data	Employee Name	Data	Zip Code	Dept. No.
(1)	...	JONES, HARRY H.	...	03078	19413
(2)	...	SMITH, JOHN Q.	...	19413	12608
(3)	...	SMITH, GEORGE R.	...	03078	03078

Figure 5—Record and file relationships

keys such as this are discussed later.) The data record pointer (00001) is used then to read the first record in the data file and turn it over to the user.

If the user specifies SHOW 19413, SADI accesses the first occurrence of the key 19413 (Zip Code), which points to the second data record. If the user's intent, however, is to access department number 19413, he can specify SHOW 19413(3), which indicates that he wants to access a data record in which 19413 appears as the third key field, department number. If the possibility of more than one employee in department 19413 exists, the user specifies the command BROWSE 19413(3), and SADI passes to the user, one at a time, all data records in which 19413 appears as the third key.

A DELETE 03078(2) command, however, leads to amibiguity, because the request refers to two data records. Primary keys are used to resolve this problem.

PRIMARY KEYS

As stated earlier, the applications programmer can specify one key field in a data record to be the primary key (key number zero). All other keys in the data records are secondary keys.

SADI does not permit more than one data record to contain the same primary key value. Thus, the primary key is used to identify, without ambiguity, a particular data record. This ability is especially useful in deleting and revising data records.

If the files in Figure 5 had been defined with a primary key, the user could access a given record with no ambiguity by specifying the unique key value, such as Social Security number or employee number.

ADDING A DATA RECORD

When the user passes a data record to SADI with an ADD request, SADI performs the same process that it does when the data file is initially created: it writes the data record into the data file, extracts and builds a directory record for each key in the data record, and inserts the directory records, in key sequence, into the directory file. If the data file has a primary key, SADI insures that the data record being added to the data file has a unique primary key.

DELETING A DATA RECORD

If a data file has no primary key, a DELETE request has questionable meaning and value. Any key value specified by the user could be contained in many data records, and the user's intent is not clear: should all data records containing the cited key value be deleted, or should just one be deleted? If one, which one? Because of the inherent ambiguity of such a condition, SADI does not permit data record deletions if no primary key exists.

If a data file has a primary key, that key must be cited in any DELETE request for that file. Otherwise, the ambiguities mentioned above would occur.

A DELETE request implies that all directory records which point to the deleted data record must also be deleted. Because the secondary key records are dispersed throughout the key file, the following procedure is used:

1. The primary directory record is brought into memory, giving access to the data record pointer.
2. The data record is brought into memory.
3. The secondary key values found in the data record are used to locate and delete each secondary directory record.
4. The primary directory record is deleted.
5. The data record is deleted.

The sequence of record deletions (secondary directory records, primary directory record, data record) insures that system crashes which occur during the process do not leave pointers to deleted records. The worst that can happen is to be left with a data record and no directory records. The data record is then inaccessible (the DELETE intended it that way!) and merely occupies "dead space" in the file.

REVISING A DATA RECORD

Data record revisions fall into three categories:

1. Data only is revised (no change to key values).
2. Secondary key values are changed (with or without other data changes).
3. The primary key value is changed (with or without other changes).

The first case (no key value changes) is simplest: The data record is read, revised, and rewritten. The directory file remains unaltered.

In the second case (secondary keys revised), when the user passes the revised data record to SADI, SADI compares all the new secondary key values with all the old secondary key values. For each secondary key which has been revised, SADI deletes the old secondary directory record and writes, in sequence, a new secondary directory record. The revised data record, of course, is rewritten into the data file.

Revision of the primary key is equivalent to requesting a new copy of the data record. When SADI discovers that the user has revised the primary key, it writes the revised record as a new record into the data file, then creates and writes the necessary primary and secondary directory records into the directory file. If the user's intent is not to copy, but truly to revise, he can then issue a DELETE request, citing the old primary key.

Revisions of data files without primary keys fall, by definition, into the first two categories, and are handled as described above.

PAGING THROUGH THE DATA FILE

The directory file is ideally suited for use in accessing data records sequentially by any or all keys. Its indexed sequential organization permits the first directory record for a given key value to be found immediately; the directory file can then be read sequentially from that point forward, thereby permitting sequential access of the directly-organized data file.

Page through the data file is accomplished through a BROWSE request. If no key is specified in the request, the first (sequential) record in the directory file is accessed. Upon a CONTINUE request, each successive directory record is read and its corresponding data record is passed to the user.

Generic key values may be specified in user requests. If the key in the user request contains fewer characters than exist in the key field, SADI accesses the first directory record which matches the request value, matching on only as many high-order characters as exist in the user request. Thus, if the user requests BROWSE 17 in a Zip Code file, the first record accessed would be for Zip Code 17014, or 17000, or whatever Zip Code is lowest in the 17000 series. This facility is useful if the user is not aware of any specific key value, but knows only the general area of the file he wishes to search. Generic keys may vary in length from one character up to the total key length. The user may thus be as general or specific in his request as he chooses.

SYSTEM MAINTENANCE

The "deletion" of a mass storage record is not done by erasing the record, but by writing a special character in a reserved location in the record. Each record read is checked for the presence of the special character; if present, the record is ignored.

SADI never reads a "deleted" data record, because its directory records are also "deleted". However, as activity in the files increases, the number of deleted directory records increases, and SADI spends more and more time reading and then ignoring deleted records. At some point in time, it becomes more economical to recreate the directory file by copying it and dropping the deleted records. Available file space may also become depleted.

If conventional indexed sequential file organization were used, instead of SADI, the process of copying the file would, of course, entail copying all the data as well as the keys. SADI requires that only the keys (contained in the directory file), which are usually significantly shorter than the data, be copied, resulting in a much more efficient maintenance system.

The descriptor file and the data files do not contribute to system inefficiencies due to deleted records, because of their direct organization. Lack of available file space is the only reason for copying these files.

Since the descriptor file has low activity, it rarely, if ever, requires maintenance. The data files, however, do periodically require recreation, although not nearly so often as the directory files. The copying process would modify the relative record addresses of most data records, however, rendering the pointers in the directory records useless. Therefore, at a convenient time when both a data file and its directory file need "cleaning up", the original file creation process is repeated, using the existing data file as input to create an entirely new data file and directory file. The corresponding old files are then scratched.

SYSTEM INTERFACE

Although the primary use of SADI today is in an on-line environment, its interfaces with the rest of the system use standard conventions, so that COBOL, FORTRAN, and PL/I programs can call SADI to perform its functions on their behalf. It has proven useful in batch programs which perform random file updates and matching functions.

The user does not describe his files or reference them through the standard operating system methods. Instead, he issues a call to SADI and passes the appropriate parameters, such as file name, command, and key value. In COBOL, for example, the user might code CALL SADI using "EMPLOYEE", "ADD", "19413".

SYSTEM IMPLEMENTATION

SADI is implemented on the IBM System/360 under Operating System/360. It uses the IBM-supplied Indexed Sequential and Direct Access Methods. Because there is no modification whatsoever to any vendor-supplied software, SADI is independent of new software releases. It requires 6,800 bytes of storage.

FUTURE IMPROVEMENTS

There are several functional and operational improvements currently under consideration.

The implementation of mnemonic key numbers would appear to be of value to the user. With such a feature, the user could specify SHOW 03078(ZIP) when he wants to access a record with that specific Zip Code; he would not have to know that Zip Code is the second key in the data record as he does now. Implementation is simple: expand the descriptor record to contain the mnemonic name of each key field, and, upon a request, translate the user's mnemonic name into the corresponding key number. Search of the directory file then proceeds as it does today.

Another study is under way to examine the tradeoffs involved in maintaining a directory of deleted data records for use in assigning addresses to new data records. Although updating and searching this directory would introduce new overheads, maintenance of the (possibly) large data file would be totally eliminated. Our results so far indicate that such a system can be implemented economically, both from a programming and operating standpoint.

If ultimately installed in our system, it will probably become an option rather than a requirement. The application programmer can then make the choice of whether or not to include the option, based on his knowledge of the data file size. The advantages of using the deleted-record directory increase as the number of records in the data file and therefore the maintenance cost increases.

CONCLUSION

SADI provides two valuable functions normally available only in expensive data management systems: access of data records by naming any one of several key values in the record, and accessing all data records which contain the same key value. Yet the initial implementation of SADI was accomplished at an estimated cost of less than $10,000. With the use of information contained in this paper, it is hoped that the user who needs these functions can implement his own system for significantly less.

Besides the enhanced functions, the system improves throughput over conventional methods by searching short directory records instead of longer data records. When file cleanup is required, it is usually more economical and faster to process SADI files than conventional ones.

The approach used in this system is similar to approaches used in several currently available proprietary data management systems. SADI employs the concept of the inverted list, in which the data file is inverted on each of its keys.[2] The approach is not new, but its usefulness in return for its implementation cost is significant. Because the concept discussed here is easily implemented on many systems, in a single module, it can be installed ~~easily~~ without perturbations in an installation's data base. All resulting files are in the operating system's standard format, so that, if necessary, additional processing programs can be written to access the files.

SADI is not a total answer to today's data management problems; I think that work currently under way in the CODASYL Programming Language Committee[3] will contribute significantly to those solutions. But SADI is here now, it works fast and efficiently, and, with a little effort, is available to virtually all users of today's operating systems.

REFERENCES

1 D S WARD
Data base technology
Honeywell Computer Journal Fall 1969 pp 14-25

2 G G DODD
Elements of data management systems
Computing Surveys June 1969 pp 117-133

3 Data Base Task Group Report to the CODASYL Programming Language Committee October 1969

Arranging frequency dependent data on sequential memories*

by C. V. RAMAMOORTHY and P. R. BLEVINS

University of Texas
Austin, Texas

INTRODUCTION

Often the arrangement of frequency dependent data such as pages on a sequential memory such as disks or tapes critically affects the turnaround time of real-time or dedicated mode processes. Since the size of typical problems renders exact solution techniques impractical, a fast, efficient heuristic procedure becomes very useful. This paper describes such a procedure which is applicable to a general class of objective functions corresponding to seek time functions constrained to be only monotonically piecewise linear.

The procedure employs a directed graph formulation analogous to the classical one-dimensional module placement problem. By combining the connectivity matrices representing the interpage frequencies and the seek distances, the objective function is related to a novel tableau called the Superimposed Tableau which allows profitable permutations to be identified as Local Gain Triangles.

Results include development of bounds for achieving a locally optimum arrangement, proofs for finiteness and feasibility, comparison of the procedure with a modified version, and simulation experiences.

Problem definition

Memory hierarchies employed by modern computing systems range from ultra fast "cache" buffer, to fast core, to bulk core, to drum, to disk, to tape. Blocks of data (pages) are transferred between different levels of the hierarchy as a function of the page use frequency. When collections of those pages are stored in a member of the hierarchy which can be characterized as possessing only one degree of access freedom, performance of the memory hierarchy depends strongly upon the arrangement of the page string.[2] When the frequency of transition between data blocks (pages, tracks) is known or predictable, favorable ordering of the blocks of information on the sequential memory can reduce the average seek time between these blocks. Ordering such frequency dependent data for storage defines an optimization problem which can be formulated in graph theoretic terminology as follows:

Given a weighted, directed graph of N nodes, determine an open string of N nodes such as to minimize

$$\sum_{i=1}^{N}\sum_{j=1}^{N} f_{ij}d_{ij}c_{d_{ij}}$$

where

f_{ij}	is a constant that denotes the transition frequency from node i to node j
d_{ij}	is a variable that denotes the normalized distance between nodes i and j expressed as an integer number
$c_{d_{ij}}$	is a constant that denotes the unit cost per d_{ij} unit

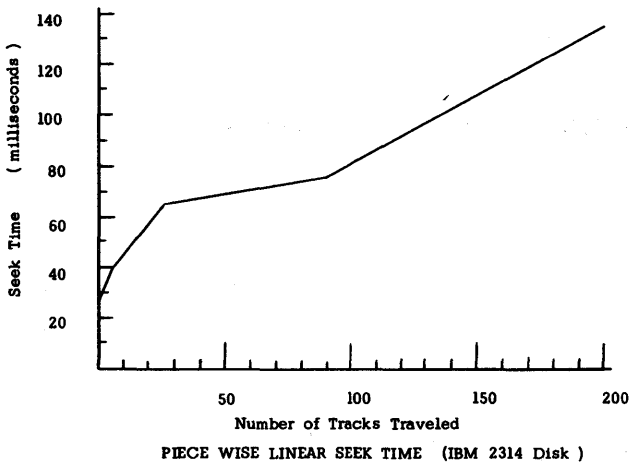
Application classes

Consider the applicable classes of disk units and tape units. Two general cases of the optimization problem exist. The product term $d_{ij}c_{d_{ij}}$ (seek time) can either be strictly linear with respect to d_{ij} or it can be only piecewise linear with respect to d_{ij}. In both cases the seek time monotonically increases with increasing d_{ij}.

Strictly linear seek time

The first case is exemplified by the typical magnetic tape unit possessing bi-directional read/write capa-

* This work was supported by National Science Foundation Grant GJ-492.



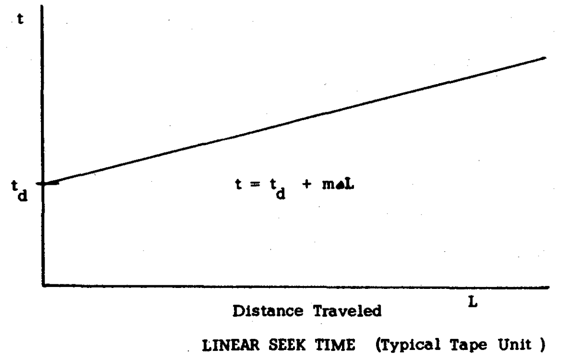


Figure 1—Application classes

bilities. With constant tape velocity and fixed page size the seek time corresponds directly to the distance (d_{ij}) between page i and page j. The initial tape acceleration delay (t_d) is common to all transitions and can be ignored. See Figure 1. The unit cost ($c_{d_{ij}}$) per unit d_{ij} is a constant for all d_{ij} allowing $c_{d_{ij}}$ to be set equal to unity by normalization. Therefore, when the seek time is strictly linear with respect to distance the objective function becomes:

$$\text{OF (LINEAR SEEK TIME)} = \sum_{i=1}^{N}\sum_{j=1}^{N} (f_{ij}d_{ij})$$

Piecewise linear seek time

The second case is exemplified by certain large disk units. First, a disk unit can be characterized as possessing only one degree of access freedom if it possesses only one head per surface and the time required for one disk revolution is much less than an intertrack head movement. In the corresponding graph model, all pages on a given track are considered to exist at a single node. Secondly, such modern disk units (IBM 2413) employ several different head velocities depending upon the number of tracks to be traveled. See Figure 1. As a result the seek time is only piecewise linear with respect to the distance traveled. Also, it is assumed that the separation distance between consecutive tracks is constant and that the seek time monotonically increases with increasing distance. Therefore, when the seek time is only piecewise linear with respect to the distance traveled, the objective function becomes:

$$\text{OF (PIECEWISE LINEAR SEEK TIME)} = \sum_{i=1}^{N}\sum_{j=1}^{N} (f_{ij}d_{ij}c_{d_{ij}})$$

Solution approaches

Such an important optimization problem recognizes two types of solutions, exact and approximate. As often discovered in combinatorics,[1] exact solutions are very costly since the number of computations is often proportional to N factorial or 2^N where N is the number of elements involved. Thus there exists a need for heuristic methods[5] possessing reasonable run time bounds.

Exact solutions

A feasible solution to the problem is any permutation of $(1,2,3, \ldots, N)$ where a permutation $(a_{i_1}, \ldots, a_{i_N})$ corresponds to placing page a_{i_1} in the first track, page a_{i_2} in the second track, and continuing until page a_{i_N} has been placed in the Nth track. An exact solution may be obtained by considering all $N!$ permutations. Run time estimates for the $(N!)/2$ undirected arrangements of a 12 node problem would require approximately 133 hours of CPU time on a CDC 6600!

Alternately, consider a dynamic programming algorithm after Karp and Held.[4] Consider the sequence in which a page a_{i_1} is placed in the first track after which a page a_{i_2} (other than a_{i_1}) is placed in the second track and so forth. When the jth page is placed in the jth track the 'state' of the sequence $(a_{i_1}, a_{i_2}, \ldots, a_{i_j})$ can be described by the unordered set $\{a_{i_1}, a_{i_2}, \ldots, a_{i_j}\}$. Hence, sequences (1, 2) and (2, 1) correspond to the same state $\{1, 2\}$. Clearly there are 2^N states.

The cost of a state $\{i_1, \ldots, i_j\}$ is defined as the minimum of the costs of all permutations of $i_1, \ldots, i_j$. As a result, the cost of the "final" state $\{i_1, \ldots, i_n\}$ is the cost of the optimal solution.

The following recursive function can be used to determine the state S cost:

$$C(\Phi)=0$$
$$C(S)=\min_{i\in s}\{C(s-i)+\sum_{k\notin s}\sum_{j\in s}(f_{jk}+f_{kj})\}$$

Since all 2^N states must be examined, the algorithm possesses a run time growth rate at least proportional to 2^N. Also, the formulation implicitly assumes that the seek time function is strictly linear. This is a severe limitation since most real problems concern monotonically piecewise linear functions.

Approximate solutions

Since both exact solution algorithms possessed unfavorable properties, a search for a heuristic procedure with favorable properties was initiated. The presented procedure possesses favorable run time bounds and is applicable to both linear and monotonically piecewise linear cost functions.

The remainder of the paper develops the procedure, describes the procedure steps, and presents an evaluation.

PROCEDURE DEVELOPMENT

This section presents the developments and definitions used by the optimization procedure. Matrix representations of both the jump (transition) frequency data and the constraints are developed. By superimposing these matrices and performing a termwise multiplication, the objective function is represented by a matrix called the Superimposed Tableau. The inherent structural properties of the formulation allow a simple, but fast heuristic optimization procedure to be developed. The fundamental development is the efficient identification of profitable node interchanges from so-called Local Gain Triangles within the Superimposed Tableau.

The simplicity of the procedure and the structural properties of the Superimposed Tableau facilitate programming the procedure. Approximately 200 Fortran statements are required and only one memory copy of the tableau is required. Obviously the procedure run time is sensitive to the number of nodes, their initial ordering, and the jump frequency data; however, the results based upon both random frequencies and those in the laboratory applications encourage its use in real-time situations.

Bounds for the number of node interchanges are developed in APPENDIX A and theorems for finiteness and feasibility are presented in APPENDIX B.

Objective function

The developed procedure employs the same objective function formulation as stated earlier in the problem definition. Since it represents the total access (seek) time required to achieve the given number of internode jumps, it is proportional to the average seek time. As a result, the procedure minimizes the average seek time for the set of data blocks arranged on the sequential memory.

Constraints

First, consider the fixed set of jump (transition) frequencies (f_{ij}). This set completely specifies the expected frequency of jumping between any two nodes in the string. The frequencies may be expressed either as integers corresponding to a number of jumps or as fractions corresponding to normalized percentages.

A convenient means of representing the set of jump frequencies employs a connectivity matrix and is called the Jump Frequency Matrix (JFM). Since it completely specifies the internode jump frequencies, it provides all the required input data for the developed optimization procedure. Note that when $i=j$, then $d_{ij}=0$. This condition corresponds to a self-loop or a jump from a node to itself. The cost of such a jump is considered to be zero since the distance traveled is zero.

Now consider the most stringent constraint which must be imposed upon the jump distance variables, d_{ij}. From the linear graph representation it can be observed that for the 5-node string, the jump distance variables must take on only the following list of integers:

Jump Distance Integer List = [1, 1, 1, 1, 2, 2, 2, 3, 3, 4]

In general, for an N node string the Jump Distance Integer List consists of the integers 1, 2, ..., $N-1$ where the integer K is included $N-K$ times. This list of integers can conveniently be entered into a matrix. To preserve the clarity of the developed tableau structure, the constant term $c_{d_{ij}}$ which modifies d_{ij} will be shown in a corner box. Since the seek time $(d_{ij}c_{d_{ij}})$ is monotonically increasing with respect to distance, $c_{d_{ij}}$ affects only the objective function evaluation. Such a matrix is called the Jump Distance Matrix (JDM).

Superimposed tableau

Superimposing the Jump Frequency Matrix upon the Jump Distance Matrix and performing a termwise multiplication generates the objective function represented on a matrix labeled as the JFM and the JDM,

			PAGE j					
			1	2	3	4	5	TABLEAU LABELS
			WL_1	WL_2	WL_3	WL_4	WL_5	WORKING LABELS
	1	WL_1	0	$1f_{12}^{\,c_1}$	$2f_{13}^{\,c_2}$	$3f_{14}^{\,c_3}$	$4f_{15}^{\,c_4}$	
	2	WL_2	$1f_{21}^{\,c_1}$	0	$1f_{23}^{\,c_1}$	$2f_{24}^{\,c_2}$	$3f_{25}^{\,c_3}$	
PAGE i	3	WL_3	$2f_{31}^{\,c_2}$	$1f_{32}^{\,c_1}$	0	$1f_{34}^{\,c_1}$	$2f_{35}^{\,c_2}$	
	4	WL_4	$3f_{41}^{\,c_3}$	$2f_{42}^{\,c_2}$	$1f_{43}^{\,c_1}$	0	$1f_{45}^{\,c_1}$	
	5	WL_5	$4f_{51}^{\,c_4}$	$3f_{52}^{\,c_3}$	$2f_{53}^{\,c_2}$	$1f_{54}^{\,c_1}$	0	

Figure 2—The superimposed tableau

This representation of the objective function will be called the Superimposed Tableau (Figure 2). Normally the tableau will be shown without the jump distance values or the jump distance unit costs since they remain fixed with respect to the Tableau position. Only the jump frequencies move upon the tableau as the optimization procedure is performed. This corresponds to the changes in the distance between the nodes without any change in the transition weights.

The Superimposed Tableau representation for the objective function possesses several beneficial structural properties. These include:

—representation of a feasible solution
—definition of a node string corresponding to each tableau stage
—grouping of objective function terms with respect to jump distance (Level)
—a convenient tableau for evaluating the objective function
—a convenient tableau for performing node interchanges

Feasible solution

Entering the given initial jump frequencies into the Superimposed Tableau establishes an initial feasible solution. The rows and columns of the Tableau possess both a set of Working Labels corresponding to the given node identification and a set of Tableau Labels which do not change. If all operations (node interchanges) performed upon the tableau preserve feasibility, then the Superimposed Tableau will always represent a feasible solution to the problem. This feasible solution corresponds to a node string consecutively labeled 1, 2, . . ., N using the Superimposed Tableau Labels. Transformation of the labels to the working node labels requires a one-to-one mapping of the node Working Labels to their corresponding Superimposed Tableau Labels. This information is directly available from the tableau.

Local triangles

The developed optimization procedure tests the so-called 'local triangles' to identify promising node interchanges. A local triangle consists of a given element in the Superimposed Tableau plus two equal distance, lower cost level elements. Within the tableau the three elements form an isosceles triangle with the single element of higher cost level being called the vertex element of the local triangle. In particular, the set of all local triangles of size D corresponds to the set of all permutations interchanging exactly two nodes of distance D apart. Formally, for $1 \leq D \leq N-1$ the local triangle may be defined by

$$(f_{i,j-D}, f_{ij}, f_{i+D,j}) \quad \text{for } j-D \geq i,\ j=2, \ldots, N,\ i=1, \ldots, N-1$$

$$(f_{i-D,j}, f_{ij}, f_{i,j+D}) \quad \text{for } i-D \geq j,\ i=2, \ldots, N,\ j=1, \ldots, N-1$$

When referenced to the corresponding node string the vertex element represents the element which might be moved to a lower level by interchanging two nodes in the node string. Examples of local triangles and their relationship to the node string are delineated in Figure 3. For the local triangle (f_{12}, f_{13}, f_{23}) the vertex element (f_{13}) could be moved to a lower level only by interchanging either nodes 1 and 2 or nodes 2 and 3. By interchanging these nodes, one element of a lower cost level must in turn be moved to a higher cost level. A local triangle cost equals $(2f_{ij}-f_{i,j-D}-f_{i+D,j})$ for $j-D \geq i$ and $(2f_{ij}-f_{i-D,j}-f_{i,j+D})$ for $i-D \geq j$. The following classifications result:

Local Optimum Triangle—Exists whenever the cost of the local triangle cannot be reduced by a node interchange.

Local Gain Triangle—Exists whenever the cost of the local triangle can be reduced by a node interchange.

For a node interchange to show a net gain there must exist at least one local gain triangle. This follows since the net gain of a node interchange is calculated from the summation of a set of local triangle costs.

In general, for an N node string, the number of local triangles of size D (D equals the jump distance between nodes) can be expressed as:

$$(N-D)\cdot(N-D+1)$$

For example, in the $N=5$ node string there exists 20 local triangles of size $D=1$. The developed procedure tests the tableau for triangles of size $N-1, \ldots, 2, 1$. By summing the number of local triangles of each size, the total number of local triangles per tableau equals

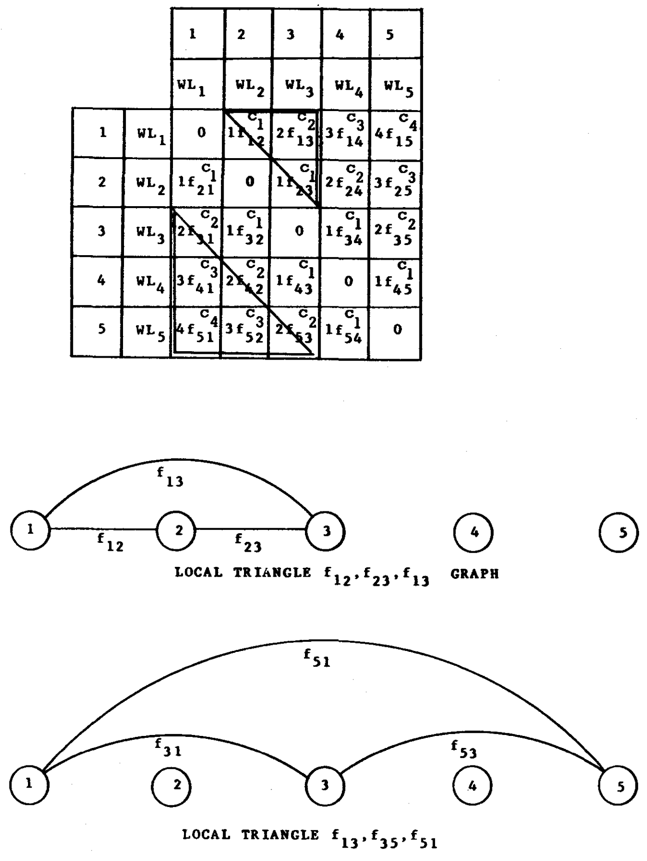
$$\sum_{D=1}^{N-1} (N-D)[(N-D)+1]=(N^3-N)/3$$

Node interchange rule

The optimization procedure to be presented introduces different permutations (changes in the tableau) by interchanging exactly two nodes. Since it is desired to always maintain feasibility of the tableau (correspondence to a realizable node string) the following operation rule is employed:

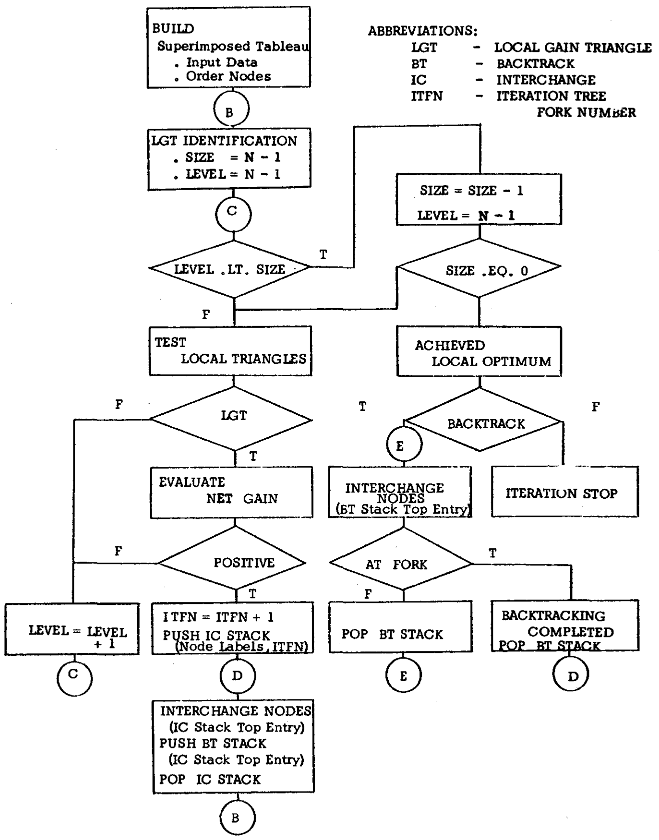
To interchange node i and node j, the following operations are required:

1. Interchange column i and column j



		1	2	3	4	5
		WL_1	WL_2	WL_3	WL_4	WL_5
1	WL_1	0	$1f_{12}^{c_1}$	$2f_{13}^{c_2}$	$3f_{14}^{c_3}$	$4f_{15}^{c_4}$
2	WL_2	$1f_{21}^{c_1}$	0	$1f_{23}^{c_1}$	$2f_{24}^{c_2}$	$3f_{25}^{c_3}$
3	WL_3	$2f_{31}^{c_2}$	$1f_{32}^{c_1}$	0	$1f_{34}^{c_1}$	$2f_{35}^{c_2}$
4	WL_4	$3f_{41}^{c_3}$	$2f_{42}^{c_2}$	$1f_{43}^{c_1}$	0	$1f_{45}^{c_1}$
5	WL_5	$4f_{51}^{c_4}$	$3f_{52}^{c_3}$	$2f_{53}^{c_2}$	$1f_{54}^{c_1}$	0

Figure 3—Local triangles



BUILD Superimposed Tableau . Input Data . Order Nodes
B
LGT IDENTIFICATION . SIZE = N - 1 . LEVEL = N - 1
C
LEVEL .LT. SIZE
T
F
SIZE = SIZE - 1 LEVEL = N - 1
SIZE .EQ. 0
TEST LOCAL TRIANGLES
ACHIEVED LOCAL OPTIMUM
LGT
F
T
BACKTRACK
T
F
E
EVALUATE NET GAIN
INTERCHANGE NODES (BT Stack Top Entry)
ITERATION STOP
POSITIVE
F
T
AT FORK
T
F
LEVEL = LEVEL + 1
ITFN = ITFN + 1 PUSH IC STACK (Node Labels, ITFN)
POP BT STACK
BACKTRACKING COMPLETED POP BT STACK
C
D
E
D
INTERCHANGE NODES (IC Stack Top Entry) PUSH BT STACK (IC Stack Top Entry) POP IC STACK
B
ABBREVIATIONS:
LGT - LOCAL GAIN TRIANGLE
BT - BACKTRACK
IC - INTERCHANGE
ITFN - ITERATION TREE FORK NUMBER

Figure 4—Procedure flowchart

2. Interchange row i and row j
3. Update the Superimposed Tableau Working Labels

Evaluation of node interchanges

The procedure detects promising node inter-changes by testing local triangles. But the net effect upon the objective function is unknown since the local triangle involves only the cost relationship between three nodes. When evaluating the net effect it is economical to do so without actually executing the interchange. The Superimposed Tableau allows such a procedure to be defined.

PROCEDURE STEPS

The preceding definitions and developments provide the necessary background for the procedure which is detailed in the flow-chart shown in Fig. 4. Since no proof of the optimality of the results is presented, the procedure must be considered heuristic.

The major steps of both the Normal Procedure and the Modified Procedure will be briefly discussed.

Normal procedure

Initial ordering

Initially the node string is ordered with respect to the weight of each node's cut set of jump frequencies. For a given node i the weight of the cut set of jump frequencies is calculated by summing the column i entries and the row i entries in the Superimposed Tableau. The string is then symmetrically ordered with respect to the weights by interchanging the required nodes.

This initial ordering strongly contributes to a faster solution. Since those nodes having a great number of jumps (transitional affinity) are initially arranged close together, the expected number of node interchanges required to achieve an optimum arrangement is reduced.

Starting arrangement

Since the Normal Procedure performs only one iteration, the symmetrically ordered starting arrangement is used.

Local gain triangle identification

The key to the procedure's advantage over an exhaustive search of all $N!$ permutations is the identification of the Local Gain Triangle. As previously explained for a node interchange to reduce the objective function there must exist at least one Local Gain Triangle involving the node pair. Therefore, by testing all local triangles 'promising' node interchanges can be detected.

In order to minimize the number of required node interchanges, testing starts with the maximum size local triangle ($D=N-1$) and proceeds to the minimum size local triangle ($D=1$). For example, consider two nodes of distance L apart whose interchange would reduce the objective function value. The procedure would detect the condition with a Local Gain Triangle of size L and one node interchange would result. Conversely, if the procedure proceeded from minimum to maximum size local triangles, several Local Gain Triangles of size less than L might be detected and each could reslut in a node interchange.

Since the Local Gain Triangle identification procedure starts with the maximum size local triangle, it follows that the location of the triangle's vertex elements proceeds from the highest level, Level $N-1$, toward Level 2, the smallest level capable of containing a local triangle of size 1. Also, proceeding from the higher to the lower levels allows the procedure to delay generating strongly connected substrings. Once generated, such substrings are usually not broken and commonly produce undesirable local optimum arrangements.

Evaluation of local gain triangles

If a Local Gain Triangle of size D is detected in a given level, testing is interrupted after completing that level. Every node interchange indicated by the Local Gain Triangles is evaluated with respect to the objective function. Those interchanges possessing positive gains are pushed into the Interchange Stack. Should no positive gain interchanges be found, testing for Local Gain Triangles of size D resumes at the next level.

Interchanging nodes

The node interchange denoted by the top entry of the Interchange Stack is interchanged and a trace is generated by pushing the information into the Backtrack Stack. This stack will be employed to backtrack to the proper fork after a local optimum arrangement has been achieved. Note that at each level of testing for Local Gain Triangles multiple positive gain interchanges may be detected. Each of these alternatives corresponds to a branch of the iteration tree. When more than one alternative is detected, a tree fork is established.

After interchanging the node pair the procedure completely restarts the Local Gain Triangle identification procedure since a Local Gain Triangle of any size and at any level may now exist.

Local optimum arrangement

Whenever the Superimposed Tableau no longer contains any Local Gain Triangles corresponding to profitable node interchanges, then a local optimum arrangement of the node string has been achieved. Such a condition represents a terminal node of the iteration tree. The procedure must next test if backtracking is required to complete exploring all positive gain interchanges.

Backtracking

If after achieving a local optimum, entries remain in the Interchange Stack, then the Superimposed Tableau must be rearranged using the trace available in the Backtrack Stack. The Superimposed Tableau is rearranged to correspond to the string arrangement which existed at the tree fork possessing the unexplored node interchange indicated by the top entry of the Interchange Stack.

Iteration stop arrangement

Whenever a local optimum is reached and the Interchange Stack is empty, then all branches of the iteration tree have been explored. The minimum local optimum detected during the iteration is selected as the Iteration Optimum Arrangement.

Heuristic optimum arrangement

Since the Normal Procedure performs only one iteration, the Heuristic Optimum Arrangement equals the Iteration Optimum Arrangement.

Modified procedure

The Modified Procedure extends the Normal Procedure by performing N iterations rather than only one. All steps described for the Normal Procedure are identical except for the Starting Arrangement and the Heuristic Optimum Arrangement steps.

Starting arrangement

The Modified Procedure employs a different starting arrangement for each of the N iterations. The N starting arrangements are generated by shifting end-around the initially ordered node string. Since the procedure employs only interchanges between two nodes, the end-around shift generates a maximally permuted node string which requires $N-1$ interchanges to undo. Restated, an end-around shift operation requires $N-1$ node interchanges to implement. Since N end-around shifts returns the arrangement to its starting arrangement, the set of N starting arrangements consists of the initially ordered arrangement plus $N-1$ arrangements generated by end-around shifts.

Heuristic optimum arrangement

From the set of N iteration optimum arrangements, the arrangement having the minimum objective function value is selected as the Heuristic Optimum Arrangement.

Comments

Since the first starting arrangement of the Modified Procedure equals the starting arrangement of the Normal Procedure, the Heuristic Optimum Arrangement of the Modified Procedure will at least be as good as that of the Normal Procedure. However, since N iterations are employed the run time will be much greater.

RESULTS

Both the Normal Procedure and the Modified Procedure have been implemented in FORTRAN IV. Using a CDC 6600 computer, numerous problems of up to 20 nodes have been processed. The path frequencies (f_{ij}) have been based upon random numbers as well as realistic values in laboratory applications.

The run time lower bound is examined using both theoretical values and timing run values.

Objective function values for the generated optimum arrangements of the Normal Procedure are compared to those of the Modified Procedure. Also the relationship between run time and the degree of optimization was studied.

Run times

Basically the procedure performs three functions which are:

(a) tests local triangles
(b) calculates net gain of detected local gain triangles
(c) searches a complete interchange alternative tree (interchanging nodes and backtracking as required)

Obviously the run time is strongly sensitive to the transition frequencies of the graph being optimized and depends upon the initial ordering scheme. See Figures 6 and 7. However, based upon numerous graphs the influence of whether the seek-time function was strictly linear or only piecewise linear appears to be slight and to depend upon the transition frequencies. A lower bound for the run time as a function of the number of graph nodes N can be expressed as

$$\text{RUN TIME (LOWER BOUND)} = K(N^3 - N)/3$$

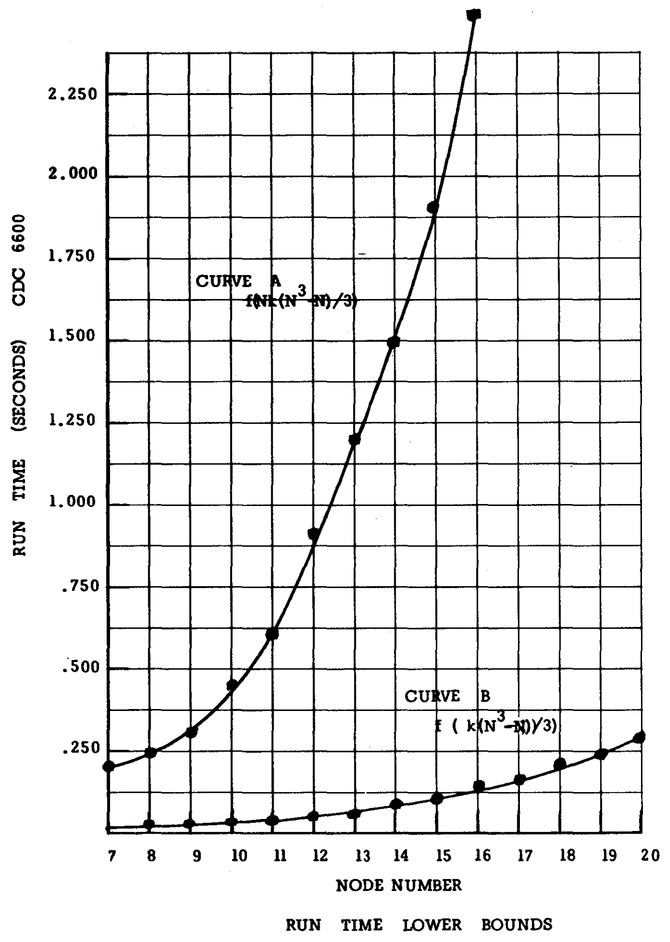


Figure 5—Run time lower bounds

where the constant K equals the time required to test one local triangle. For the Modified Procedure which employs N iterations the bound must be multiplied by N.

Note that the heuristic procedure tests local triangles by just subtracting two elements in a matrix and testing the sign of the difference. Time consuming operations which are functions of N result only if local gain triangles are detected. Each local gain triangle causes the net gain relative to the objective function to be evaluated (a matrix operation involving two rows and two columns). If a positive net gain results, the corresponding nodes are interchanged (a matrix operation which interchanges entries in two rows and two columns). Based upon timing runs (CDC 6600, FORTRAN IV code) the 10,912 local triangles for a graph of 32 nodes can be tested in approximately 1.1 seconds. If the procedure employs N iterations (where each starting arrangement is an end-around shift of the initially ordered arrangement) then the 349,184 local triangles can be tested in approximately 35 seconds.

SUBGRAPH	OBJECTIVE FUNCTION VALUES				RUN TIMES	
(N)	Symmetrically Ordered	Procedure Normal	Procedure Modified	Difference Percentage	Procedure Normal	Procedure Modified
20	365 968	341 367			22.080	
19	335 647	309 052			10.422	
18	280 261	262 470			7.960	
17	240 745	222 180			11.982	
16	202 539	191 682			2.912	
15	180 579	161 935	161 935	0.00	11.168	99.149
14	147 805	139 715	137 101	1.87	.944	49.449
13	131 288	115 035	113 995	0.90	.837	37.479
12	99 807	92 619	91 651	1.04	2.147	23.859
11	78 200	68 844	68 466	0.54	1.204	16.265
10	63 244	54 432	54 432	0.00	1.742	16.062
9	45 604	41 408	40 733	1.63	.460	6.428
8	32 330	29 649	29 294	1.19	.291	3.259
7	21 775	19 069	19 069	0.00	.245	1.802
6	15 334	13 120	12 734	2.94	.029	.741
5	7 074	6 235	6 094	2.26	.093	.243

Figure 6—Graph A optimization—Piecewise linear seek times

Note that this lower bound is achieved only if the given arrangement is already optimum.

Figure 5 plots actual values of the implemented procedure's lower bound. Since all jump frequencies (f_{ij}) were equal, the given arrangement was optimum. Curve A corresponds to the lower bound $K(N^3-N)/3$ of the Normal Procedure while Curve B corresponds to the lower bound $KN(N^3-N)/3$ of the Modified Procedure.

Optimum values

To evaluate the optimum arrangements generated by the procedure several graphs of up to 20 nodes were processed. Figure 6 shows the results for a sequential memory possessing a piecewise linear seek time. The seek time function was based upon the IBM 2314 Disk Unit (Figure 1). Figure 8 completely defines Graph A by showing the Superimposed Tableau corresponding to the initially ordered arrangement. Note that the

SUBGRAPH	OBJECTIVE FUNCTION VALUES				RUN TIMES	
(N)	Symmetrically Ordered	Procedure Normal	Procedure Modified	Difference Percentage	Procedure Normal	Procedure Modified
20	127 412	113 330			22.911	
19	115 086	102 490			44.094	
18	92 796	84 638			6.604	
17	79 782	69 686			16.914	
16	63 460	57 386			6.373	
15	55 558	47 746	47 746	0.00	8.624	125.444
14	43 454	39 290	39 164	0.32	1.503	52.065
13	38 180	31 566	31 566	0.00	4.092	63.938
12	26 850	24 774	24 774	0.00	1.003	30.912
11	20 868	18 146	17 892	1.39	.449	19.086
10	15 986	13 594	13 594	0.00	1.905	10.848
9	11 010	9 678	9 678	0.00	.412	5.850
8	7 200	6 760	6 760	0.00	.190	4.034
7	4 622	4 152	4 152	0.00	.369	2.057
6	3 184	2 724	2 684	1.46	.267	.577
5	1 432	1 280	1 236	3.43	.026	.261

Figure 7—Graph A optimization—Strictly linear seek times

```
                                        SUPERIMPØSED TABLEAU

NØDE LABELS

     7    14     4    10    17     2     9    19     5    18    12     3     1    13     6    11    15     8    16    20

NØDE WEIGHTS

  1370  1744  1830  1884  1910  1992  2054  2078  2114  2190  2216  2132  2114  2056  2044  1956  1900  1832  1776  1664




   0.0   9.0  22.0  92.0   7.0  44.0  12.0   9.0  64.0  82.0  51.0   1.0  67.0  61.0  56.0   4.0  40.0  36.0  11.0  17.0
   9.0   0.0  66.0  70.0  95.0  36.0  82.0   3.0  77.0  19.0  29.0  74.0  75.0  21.0  19.0  54.0  30.0   6.0  48.0  59.0
  22.0  66.0   0.0  88.0   4.0  23.0  25.0  47.0   6.0  98.0  62.0  12.0  17.0  93.0  49.0  54.0  31.0  96.0  84.0  38.0
  92.0  70.0  88.0   0.0  93.0  33.0  13.0  84.0  21.0  18.0  43.0  64.0  51.0  95.0  63.0  10.0  15.0  23.0  38.0  28.0
   7.0  95.0   4.0  93.0   0.0  39.0  29.0  81.0  71.0  39.0  37.0  54.0  12.0  95.0  29.0  81.0  94.0  39.0  27.0  29.0
  44.0  36.0  23.0  33.0  39.0   0.0  57.0  79.0  93.0   9.0  73.0  24.0  39.0  85.0  94.0  89.0  89.0  40.0  36.0  14.0
  12.0  82.0  25.0  13.0  29.0  57.0   0.0  33.0  83.0  81.0  95.0  53.0  98.0  35.0  42.0  19.0  38.0  51.0  92.0  89.0
   9.0   3.0  47.0  84.0  81.0  79.0  33.0   0.0  90.0  81.0   7.0  92.0  78.0  28.0  47.0  30.0  88.0  35.0  73.0  54.0
  64.0  77.0   6.0  21.0  71.0  93.0  83.0  90.0   0.0  81.0  77.0  35.0  81.0  16.0  21.0  34.0  41.0  89.0   8.0  69.0
  82.0  19.0  98.0  18.0  39.0   9.0  81.0  81.0  81.0   0.0  46.0  63.0  88.0  81.0  63.0  93.0  56.0  41.0  45.0  11.0
  51.0  29.0  62.0  43.0  37.0  73.0  95.0   7.0  77.0  46.0   0.0  32.0  79.0  92.0  83.0  74.0  81.0  57.0  75.0  15.0
   1.0  74.0  12.0  64.0  54.0  24.0  53.0  92.0  35.0  63.0  32.0   0.0  90.0  57.0  55.0  64.0  68.0  99.0  42.0  87.0
  67.0  75.0  17.0  51.0  12.0  39.0  98.0  78.0  81.0  88.0  79.0  90.0   0.0  31.0  87.0  44.0  28.0  43.0  26.0  23.0
  61.0  21.0  93.0  95.0  95.0  85.0  35.0  28.0  16.0  81.0  92.0  57.0  31.0   0.0  63.0  81.0  10.0  10.0  28.0  46.0
  56.0  19.0  49.0  63.0  29.0  94.0  42.0  47.0  21.0  63.0  83.0  55.0  87.0  63.0   0.0  72.0  91.0  20.0  37.0  31.0
   4.0  54.0  54.0  10.0  81.0  89.0  19.0  30.0  34.0  93.0  74.0  64.0  44.0  81.0  72.0   0.0  67.0  53.0  31.0  24.0
  40.0  30.0  31.0  15.0  94.0  89.0  38.0  88.0  41.0  56.0  81.0  68.0  28.0  10.0  91.0  67.0   0.0  37.0  30.0  16.0
  36.0   6.0  96.0  23.0  39.0  40.0  51.0  35.0  89.0  41.0  57.0  99.0  43.0  10.0  20.0  53.0  37.0   0.0  58.0  83.0
  11.0  48.0  84.0  38.0  27.0  36.0  92.0  73.0   8.0  45.0  75.0  42.0  26.0  28.0  37.0  31.0  30.0  58.0   0.0  99.0
  17.0  59.0  38.0  28.0  29.0  14.0  89.0  54.0  69.0  11.0  15.0  87.0  23.0  46.0  31.0  24.0  16.0  83.0  99.0   0.0

ØBJECTIVE FUNCTIØN VALUE =              365968.48
```

Figure 8—Initial stage of superimposed tableau—Graph A

graph is complete and that the minimum (1370) and maximum (2216) node weights differ by less than a factor of 2. Graphs without dominant nodes generally require more interchanges to optimize. The subgraphs were formed by deleting nodes from the original graph. Then the subgraph was again symmetrically ordered with respect to node weight; i.e., the node with the greatest weight was placed in the middle of the node string. Figure 7 shows the result for sequential memory possessing a strictly linear seek time.

For subgraphs of Graph A of up to 15 nodes the procedures yielded minimum objective function values differing by less than 4 percent. Since the Modified Procedure includes the Normal Procedure as its first iteration, its minimum objective function value is always less than or equal to that of the Normal Procedure.

Tradeoffs

Comparisons of the run times for the two procedures clearly indicate their major difference. Since the Modified Procedure performed N iterations (where each starting arrangement was generated by an end-around shift of the initially symmetrically ordered arrangement), it might be expected that the run time would be N times the run time for the Normal Procedure (where its single starting arrangement was the symmetrically ordered arrangement). Run time results for Graph A show increases greater than N. Upon closer examination such increases can be explained. The Normal Procedure employs only the symmetrically ordered arrangement which was chosen because the expected number of node interchanges required for optimization should be a minimum. Therefore, its run time is the expected minimum of the run times corresponding to arrangements generated by end-around shifting.

Based upon the results for Graph A, the Normal Procedure appears more economical than the Modified Procedure. Consider the subgraphs of 14 and 15 nodes. For the subgraph of 15 nodes both procedures achieved identical optimum values, but the run time of the Modified Procedure was approximately nine times greater. For the subgraph of 14 nodes the Modified

Procedure achieved an optimum value which was slightly better (2 percent), but required a run time approximately 50 times greater. A user should weigh the long term effect of the better optimum compared with the one time cost for the additional computer time. The economies of the optimization problem may easily justify the required additional computer time, but for real-time applications the Normal Procedure appears to be more useful.

CONCLUSIONS

As noted, achieving exact solutions becomes impractical for problems as small as 12-15 nodes. Since common problems usually involve more than 12-15 nodes, heuristic procedures offer the only practical solution. Besides achieving results within practical run times (real-time applications), our procedure accepts without limitations seek time functions which are only monotonically piecewise linear. Also, the procedure generates a plurality of locally optimum arrangements. Often such alternatives are more useful to a designer than a single absolute optimum arrangement. Also, by maintaining a feasible solution at all times during execution, the procedure generates useful results even if interrupted at arbitrary time limits.

The salient properties of the procedure can be generalized to attack many related problems such as arranging a closed node string (drums). Its fundamental property is the efficient identification of profitable permutations as Local Gain Triangles within the Superimposed Tableau.

Variations may be applied to enhance the procedure for specific problems. Since the run time is a function of the number of nodes, faster times may be achieved by node reductions. Techniques might include: fixing the position of one or more nodes, reducing strongly connected node clusters into single nodes, and partitioning large strings into substrings of strongly connected nodes. The starting arrangement influences both run time and the set of locally optimum arrangements. Numerous initial ordering and shifting schemes may be employed. A promising compromise between the Modified Procedure and the Normal Procedure would employ shifting bi-directionally the initial arrangement by a reduced number of shifts (about $N/4$).

As presented the procedure minimizes the objective function. Clearly, it can be employed to maximize the objective function by redefining the Local Gain Triangle.

ACKNOWLEDGMENT

The authors wish to record their sincere appreciation of the suggestions and advice of Professor K. M. Chandy of The University of Texas at Austin.

REFERENCES

1 E J BECKENBACH
Applied combinational mathematics
John Wiley 1964

2 P J DENNING
Effects of scheduling on file memory operations
1967 Spring Joint Computer Conference Proceedings Vol 30 pp 9–21

3 T C HU
Integer programming and network flows
Addison-Wesley 1969

4 R M KARP M HELD
Finite state processes and dynamic programming
SIAM Journal on Applied Mathematics Vol 15 No 3 pp 693–718 May 1967

5 B W KERNIGHAN
Some graph partitioning problems related to program segmentation
PhD Thesis Princeton University January 1969

6 D ZATYKO J DOBBIE
A mass memory system designed for the multi-program/mutli-processor users
Proceedings ACM 20th National Conference 1965 pp 487–500

APPENDIX A—BOUNDS FOR ACHIEVING LOCALLY OPTIMUM ARRANGEMENTS

Cost of executing the procedure is difficult to specify. However, to achieve a locally optimum arrangement the procedure examines local triangles in order to detect local gain triangles. Examination of a local triangle consists of just two subtractions involving tableau elements. Both a Lower Bound and an Upper Bound for the number of local triangles examined will be presented. The upper bound must be considered only as the very extreme bound since it is not realizable, but it is useful for proving the finiteness of the procedure.

Lower bound

To develop a lower bound for the number of local triangles which the procedure examines in order to achieve a locally optimum arrangement, assume that all f_{ij} of the given node string are equal. Under such assumptions no local gain triangles exist. Therefore, the lower bound corresponds to the total number of local triangles in the tableau, namely

$$\text{BOUND } L = (N^3 - N)/3 \approx N^3/3 \text{ for } N \gg 1$$

where N is the number of nodes in the string.

Upper bound

To establish an upper bound for the number of local triangles which the procedure examines in order to achieve a locally optimum arrangement, assume that the locally optimum arrangement is maximally permuted with respect to the starting arrangement.

Definition:

If the transformation of arrangement A of N nodes into arrangement A' requires a minimum of $N-1$ permutations which generate unique arrangements by interchanging exactly two elements (transpositions), then A' is said to be maximally permuted with respect to A.

Now consider the following two theorems from the study of permutation groups:

Theorem 1:

At most, $N-1$ two-node transpositions are required to transform arrangement A into A' where A' is any member of the set of all possible permutations of A.

Theorem 2:

A permutation interchanging exactly two elements separated by D elements can be factored into a minimum product of $2D+1$ permutations interchanging exactly two adjacent elements.

Since the starting arrangement is maximally permuted with respect to a locally optimum arrangement, a minimum of $N-1$ permutations interchanging exactly 2 elements are required to achieve a locally optimum arrangement. Furthermore, assume that the properties of the corresponding graph are such that the procedure achieves the locally optimum arrangement using only interchanges of adjacent elements.

Under such assumptions, the maximum number of interchanges required to achieve a locally optimum arrangement equals

$$(N-1)(2D_{\text{AVG.}}+1)$$

where

$$D_{\text{AVG.}}=\left(\sum_{D=0}^{N-2} D\right)/N-1=(N-2)/2.$$

Therefore

$$\text{MAXIMUM INTERCHANGES}=(N-1)^2$$

Since only interchanges of adjacent nodes are allowed, all local triangles must be examined to determine each interchange. The resulting Upper Bound for the number of local triangles examined equals the product of the number of local triangles per tableau and the number of interchanges

$$\begin{aligned}\text{BOUND } U &= (N-1)^2(N^3-N)/3\\ &= N(N+1)(N-1)^3\\ &- N^5/3 \quad \text{if} \quad N \gg 1.\end{aligned}$$

It should be noted that this upper bound is not realizable, but is useful for proving the finiteness of the procedure.

Example:

Let the starting arrangement be $A=bcdea$ and the locally optimum arrangement which is maximally permuted be $A'=abcde$. Therefore,

$$A'=(\pi_{45}\pi_{35}\pi_{25}\pi_{15})A$$

where π_{ij} denotes the interchange of elements in positions i and j.

Factoring each permutation into a minimum product of $2D+1$ permutations interchanging exactly two adjacent nodes

$$\pi_{45}=\pi_{45} \qquad \pi_{25}=\pi_{23}\pi_{34}\pi_{45}\pi_{43}\pi_{32}$$
$$\pi_{35}=\pi_{34}\pi_{45}\pi_{43} \qquad \pi_{15}=\pi_{12}\pi_{23}\pi_{34}\pi_{45}\pi_{43}\pi_{32}\pi_{21}$$

or $(N-1)^2=16$ permutations interchanging exactly two adjacent elements are required.

APPENDIX B—THEOREMS

Feasibility theorem

Every stage of the Superimposed Tableau defines a feasible solution.

Proof:

It is given that the procedure initially enters a feasible solution into the Superimposed Tableau. All operations performed upon the tableau employ only the Node Interchange Rule which preserves feasibility. Therefore, every stage of the Superimposed Tableau defines a feasible solution.

Finiteness theorem

The procedure terminates after performing a finite number of node interchanges.

Proof:

It has been shown that for a given starting arrangement the maximum number of node interchanges required to achieve a locally optimum arrangement is $(N-1)^2$. Since each interchange must reduce the objective function, a maximum of $(N-1)^2$ decrements could exist. Since only a finite number of alternatives exist at each fork of the iteration tree and each node interchange reduces the maximum number of remaining interchanges by one, the procedure terminates after performing a finite number of node interchanges.

Associative processing of line drawings

by NEIL J. STILLMAN and CASPER R. DEFIORE

Rome Air Development Center (EMBIH)
Griffiss Air Force Base, New York

and

P. BRUCE BERRA

Syracuse University
Syracuse, New York

INTRODUCTION

The marriage of computer graphics and an associative memory is a natural union. This is evidenced by the widespread use of software simulations of associative memories in today's most flexible graphical systems. The content-addressability of a hardware associative memory makes conventional addressing schemes superfluous and eliminates the need for pointers required to link related data, vastly reducing system overhead. The parallel retrieval and update functions possible with a hardware associative memory remove any need for multiple storage which is so prevalent in current systems and simultaneously increases processing speed. The capability of implicitly storing relations between data further decreases the storage requirements, while increasing flexibility.

After examining current graphical data structures, all of which rely on a maze of pointers or multiple storage of information to represent the naturally relational graphical data, and reviewing the fundamentals of associative memories, a data structure utilizing an associative memory to process line drawings is presented.

BACKGROUND

One of the first systems to allow graphical communication with a computer, SKETCHPAD,[18,8] utilized two-way pointers. The data about drawings were actually structured in two separate forms. The first was a table of display spot coordinates designed to make display as rapid as possible, while the other was a ring structure designed to contain the topology of the drawing and facilitate its modification. Each entity consisted of n consecutive storage locations, with standardized locations for information about the various properties of each entity type. All references to a particular entity block were linked together by a string of pointers originating within that block and pointing to the succeeding and preceding members of the string. Different rings thread through several levels in an element providing several paths to the same information. Sutherland comments that his ring structure was not intended to pack the required information into the smallest possible storage space and that some redundancy was included in the ring structure to provide faster running programs. SKETCHPAD placed a higher priority on speed than on the ability to store huge drawings.

Another ring-oriented data structure is CORAL,[19,8] (Class Oriented Ring Associative Language). It stores data in blocks of arbitrary but fixed length. The blocks represent objects which can be connected by rings; each object can belong to more than one ring allowing the multi-dimensional associations required for graphical data structures. Unlike SKETCHPAD, CORAL isn't limited to two-way pointers; all ring elements have a forward pointer to the next element in the ring, and pointers to the ring start (type identifier) are alternated with back pointers for all ring elements. By alternation of the less useful pointers, CORAL retains the flexibility afforded by each pointer type, but requires only half the space and does not incur a significant time loss. Since efficiency was not a major consideration during the system's development, storage space and processing time produce a high overhead.

Similarly, DAC-1[11] and its successor APL[6,8] at General Motors use blocks of entity descripters, each of which describes an entity and its properties, linking blocks in current use together in a ring structure. By decomposing a picture into entities a hierarchical structure is obtained.

ASP[12,8] (Associative Structure Package), is another ring implemented data structure, but differs from the others discussed in that it is a dual ring structure. All elements belong to two rings; the "upper" ring being those elements possessing the same property; and the "lower" ring being a series of rings of elements related to the master element by different properties. The ASP structure allows interrogation in seven associative forms (Feldman[16]). Lang concludes that the user, depending upon his application, should determine how the rings are to be implemented, i.e., with only forward pointers, or with backward and/or ringstart pointers. If the rings are small, forward pointers are probably sufficient while if the rings are large other pointers should perhaps be introduced.

A slightly different approach is the data structure of GRAPHIC-2,[3] basically a directed graph with no closed loops. The structure contains four types of blocks; nodes and branches of fixed length and leaf and data blocks of arbitrary size. By convention, only the leaf blocks can contain displayable material, while the other blocks provide structural information. Because space is a scarce resource in the GRAPHIC-2 computer, an abbreviated pointer system is utilized, including neither back pointers nor ringstart pointers. To quote Christensen, "Tracing one's way through the structure therefore may require more time, but time is a resource that is more readily available in GRAPHIC-2." The directed graph is used also by Cotton and Greatorex[4] in their remote computer graphics system, and serves as the basis for the graphics data structure used at the University of Utah.[2]

Van Dam and Evans,[20] in an effort to reduce the size of a given graphical item to the absolute minimum, have kept their data structure as pointer-free as possible. The general structure of an item is a block containing (1) a set of "keys" which name or identify the information within an item, (2) elements which may contain any type of information, i.e., data, program, or both, and (3) a table of contents which associates the keys with their related items and thereby allows the system to locate elements within an item. An item is retrieved by providing a "description," a logical expression combining keys, elements, and conditions on the values of elements. Schemes for keeping part of the picture in tree form and part in reduced form (points and lines) are being considered for future implementations. The fact that the points and lines form makes lightpen pointing impractical but is nevertheless considered implies that the storage space is at a great premium.

The improvement of these systems centers about two tasks regardless of the data structure discussed: the processing of data at a faster rate, and the storing of data in the smallest possible space. These two goals, to date, have been incompatible, i.e., processing speed has been gained at the expense of storage and storage can be minimized only at the cost of processing time. With the advent of an "associative memory," speed and storage compression become compatible. The parallel search capability speeds processing while the content-addressability, which eliminates conventional addresses and therefore data pointers and all housekeeping functions associated with them, both increases speed and decreases storage requirements.

FUNDAMENTALS OF ASSOCIATIVE MEMORIES[21]

An associative memory has three main features not possessed by conventional memories: (1) word-parallel access of the entire memory, (2) word-parallel performance of its basic operations in the entire memory, and (3) the inclusion of comparison as a basic operation. In addition, word operations may be performed either bit-serial or bit-parallel. Bit-serial operation will compare sequentially against a 1 bit by n word slice of memory (where n = number of words in AM) across the word while bit-parallel operation will compare the entire memory m bits by n words (m = number of bits in a word) simultaneously. It appears that this distinction is of minor consequence, introducing only the element of a time delay without affecting the three prime features noted above.

The fundamental operations of any memory are reading from, and writing into, any word or bit location. An associative memory adds comparison to these two universal requirements. It is the parallel execution of the primary operations that sets the associative memory apart. A natural associative memory strategy is that of two-block partitioning, i.e., in order to perform an operation on *some* members of the associative memory, the members that are not to be operated on must be segregated. This is accomplished by an initial operation performed in parallel on all locations of the associative memory to flag the members of interest. Then, for example, a parallel write could be used to zero out all flagged words simultaneously, or a parallel read could be used to read the contents of the flagged words simultaneously into an external buffer. In the comparison operation, the reference word (comparand) is simultaneously compared with all flagged words. The comparand is not restricted to an entire word but may be any arbitrarily specified field in the word.

The provisions of word-parallel access and simultaneous comparison make the conventional concept of a memory address obsolete. Formerly, when only one word of a computer could be accessed at a time, information was stored in an orderly fashion in uniquely numbered storage locations. In the associative memory, information is retrieved by content, not location, hence the term "content-addressable memory." An associative memory is ideally suited to cross-referencing because unlike a conventional memory which must maintain a separate index for each characteristic, information may be retrieved on any combination of characteristics.

The instruction capabilities of associative memories are usually grouped into two categories; search instructions and arithmetic functions. The search instructions allow simultaneous comparison throughout any portion of memory (i.e., any number of words) and upon any portion (field) of a word (i.e., any number of bits). The search instructions[9] include the following: equality, inequality, maximum, minimum, greater than, greater than or equal, less than, less than or equal, between limits, next higher, and next lower. The Boolean operations AND, inclusive OR, exclusive OR, and complement may be performed between fields to provide complex query capability. Arithmetic operations of addition, subtraction, multiplication, division, increment field, and decrement field are indispensable in such graphical operations as scaling and translation.

In summary, an associative memory is ideally suited to perform operations on large amounts of data since it can operate on all members of the data simultaneously, in the time of a single operation, the only constraint being memory size. An associative memory therefore, in theory, has a speed advantage in proportion to the number of words of data to be processed.

GRAPHICS AND THE ASSOCIATIVE MEMORY

Ring structures yield answers to questions such as "What are the coordinates of Square X?," and its converse "(X_1, Y_1), (X_2, Y_2), (X_3, Y_3), (X_4, Y_4) are the coordinates of what square?" by virtue of their forward and backward pointers. There are more than two ways to pose a query however. Consider the question "What is the relationship, if any exists, between point X and point Y?" or "What pairs of objects are associated by the relationship SIDE OF?" In all, there are seven associative forms[16] of a query as shown in Figure 1. Since the relation is not explicitly stored in any of the previously discussed data structures, there is no way of answering questions phrased in forms 4, 5, 6, or 7. In order to answer questions in the last four ways, conventional concepts of data processing must be abandoned. The new structure must store, in addition to the objects, the relationship associating them. This task has been accomplished in similar ways by Rovner and Feldman,[15] Ash and Sibley,[1] and Levien and Maron.[13] In these approaches the "triple" (form 1) ATTRIBUTE OF OBJECT = VALUE is the basic element of the data structure. Levien and Maron add a fourth parameter by giving each "triple" a name identifier. This parameter may be used as an element in another "triple." In these approaches, each "triple" is stored at least three times to *simulate* an associative memory and enable queries in all seven associative forms to be more efficient than in a single listing.

Searching is minimized by using a hashed addressing scheme which will translate the query directly into the address of the answer (or linked to the answer). A hashed addressing scheme does, however, produce conflict situations, i.e., more than one pair of elements can hash to the same address, producing a conflict that must be resolved, for example, by a chained search of other answers until the desired answer is identified. The tradeoff is between tolerable conflict and the size of the addressable space.

Two major improvements to the present-day simula-

	ATTRIBUTE OF	OBJECT	=	VALUE
(1)	SIDE OF	SQUARE 1	=	LINE 1
(2)	SIDE OF	SQUARE 1	=	?
(3)	SIDE OF	?	=	LINE 1
(4)	?	SQUARE 1	=	LINE 1
(5)	SIDE OF	?	=	?
(6)	?	?	=	LINE 1
(7)	?	SQUARE 1	=	?

Figure 1—Associative forms of a query

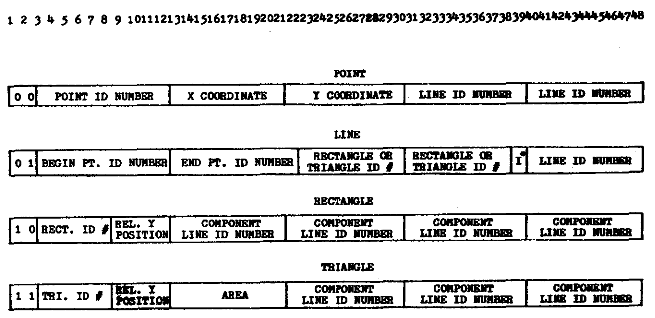

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
POINT
0 0 | POINT ID NUMBER | X COORDINATE | Y COORDINATE | LINE ID NUMBER | LINE ID NUMBER
LINE
0 1 | BEGIN PT. ID NUMBER | END PT. ID NUMBER | RECTANGLE OR TRIANGLE ID # | RECTANGLE OR TRIANGLE ID # | I* | LINE ID NUMBER
RECTANGLE
1 0 | RECT. ID # | REL. Y POSITION | COMPONENT LINE ID NUMBER | COMPONENT LINE ID NUMBER | COMPONENT LINE ID NUMBER | COMPONENT LINE ID NUMBER
TRIANGLE
1 1 | TRI. ID # | REL. Y POSITION | AREA | COMPONENT LINE ID NUMBER | COMPONENT LINE ID NUMBER | COMPONENT LINE ID NUMBER

Figure 2—Multi-relational graphics data structure

tions of the associative memory would be the elimination of multiple storage of all relations and the removal of the conflict situation caused by hashing. Both are provided by a true "associative memory," which processes a random list of triples in any of the seven query modes in the most efficient manner possible, having stored it only once while not requiring an addressing scheme. Even though the data triples are stored only once, by the nature of the word-parallel, bit-serial operation with masking of an associative memory, all seven associative form questions can be answered with equal ease.

In the above scheme each triple takes up one word in the associative memory, i.e., the data are structured one relation per word. Storing one relation per word, however, doesn't even begin to take advantage of the power of an associative memory. Utilizing a Control Data 1604B computer interfaced with a prototype associative memory built by Goodyear for Rome Air Development Center[9] and containing 2048 forty-eight bit words, a complete two dimensional line-drawing graphics system can be implemented. Imposing the following constraints on the system will assure that any drawing will fit completely in the associative memory. The maximum number of points per drawing will be 1024, and the maximum number of lines will be 512. Also, a maximum of 64 rectangles and 64 triangles can be defined. Two bits of each word are required to specify the entity type. Six bits defines a unique ID number for each rectangle or triangle, while it takes nine bits and ten bits to specify lines and points respectively with a unique identifier. The four relations

SIDE OF SQUARE X = LINE W

SIDE OF SQUARE X = LINE X

SIDE OF SQUARE X = LINE Y

SIDE OF SQUARE X = LINE Z

which are stored three times (approximately 12 words) in LEAP[15] and similar systems, and once (four words) in the same system using an associative memory can be stored in one word by placing the nine bit codes of the four lines in the word identifying the rectangle which they compose. This particular example therefore requires about 10 percent of the storage requirement of any system in existence today. Triangles, lines, and points are defined similarly (See Figure 2). The limits specified above provide for identification of all entities in 1664 words, leaving 336 words unused. A point may belong to more than two lines but only space to specify two is provided. At absolutely no overhead to the system another word may be used, repeating the first 30 bits of the point record, and specifying the identifiers of two additional lines. This may also be done with a line which belongs to more than two rectangles or triangles. Exclusive of time factors external to the associative memory (which would be incurred conventionally as well) it would take, for example, less than 60 microseconds to retrieve the record of a specific rectangle.[9]

Possibly the most notable feature of the planned implementation is its extremely fast update capability. Scaling and translation, which are merely multiplication by, and addition of, a constant respectively are accomplished, in their entirety and regardless of the complexity of the picture, in the same time that a conventional memory processes one coordinate. This fact that retrieval and update functions are completely independent of picture complexity (as long as the picture is contained completely in the AM) sums up the greatest advantage of the associative memory. Another notable feature is that the system overhead per picture, again regardless of the complexity of the picture, is always four words.

The update or modification of a picture is most dependent on "pointers" or "relations" and it is threading through all these that consumes most of the time in conventional approaches. The elimination of this maze due to the content addressing of the associative memory provides, for example, the deletion of a line and therefore all objects to which it belongs (i.e., rectangles, etc.) in about 140 microseconds.[9]

CONCLUSIONS

Feldman's simulator[7] raised the question whether it would pay to build hardware associative memories for general purpose use since it should be feasible to build a software system which loses a factor of about two in storage, and three-to-five in time, against an associative memory of the same basic speed.

It should be noted, however, that according to Minker,[14] present day relational data systems technology has emphasized retrieval to the exclusion of maintenance, i.e., update capability. Maintenance functions depend primarily on the "pointers" or "relations" and therefore associative memories will exert their maximum influence in this area.

In addition, the work of Sibley[1,17] is patterned after Rovner's[15] "triples" using hashed addressing. His view is that software simulations of associative memories "for the moment . . . are a stopgap measure."

Feldman's estimate of a loss of a factor of two in storage to an associative memory seems very conservative in light of the new data structure introduced above. Figures as to the time advantage of such a system will have to await implementation of the data structure but it is expected that timing results will show Feldman's estimate of a saving of three-to-five in time to also be very conservative.

The software simulated associative memory using hashing is limited to an exact match operation, and all other search strategies must be built on multiple use of the exact match operation, due to the fact that hashing requires a completely specified field on which to apply the hashing algorithm. On the other hand, a hardware associative memory has about a dozen different basic search capabilities indicating that a hardware associative memory is far more flexible than a software simulation of an associative memory.

As an example, consider the problem of finding all lines of length between four and six inches. Let the name and length be specified for each line. In the simulated associative memory if hashing is done by name only, or by name and length, the question cannot be answered; if hashing is by length only *and* the lengths are integral then an exact match on lengths four, five, and six will yield the answers. However, if the lengths are continuous between four and six, then again, for all intents and purposes, the simulated associative memory cannot yield an answer. In contrast, a hardware associative memory would do a single between limits parallel search and arrive at a complete solution in less than twice the time required for an exact match.

As integrated circuits come into widespread use and the price of an associative memory drops to about twice that of a conventional memory[5] more and more people will begin to examine its unique advantages.

REFERENCES

1 W L ASH E H SIBLEY
TRAMP—An interpretive associative processor with deductive capabilities
Proceedings ACM National Conference 1968 pp 143-56

2 S CARR
Geometric modeling
University of Utah Technical Report 4-13 1969

3 C CHRISTENSEN E N PINSON
Multi-function graphics for a large computer system
Proceedings Fall Joint Computer Conference Vol 31 1967 pp 697-711

4 I COTTON F S GREATOREX JR
Data structures and techniques for remote computer graphics
Proceedings Fall Joint Computer Conference Vol 33 Part I 1968 pp 533-544

5 C DEFIORE
Fast sorting
Datamation Vol 16 No 8 August 1 1970 pp 47-51

6 G G DODD
APL—A language for associative data handling in PL/I
Proceedings Fall Joint Computer Conference Vol 29 1966 pp 677-684

7 J A FELDMAN
Aspects of associative processing
MIT Technical Note 1965-13 April 1965

8 J C GRAY
Compound data structure for computer-aided design—A survey
Proceedings ACM National Conference 1967 pp 355-365

9 *Handbook of operating and maintenance instructions for the associative memory*
Vol II—Associative Memory Programming Manual
Goodyear Aerospace Corporation Akron Ohio
GER-13738 March 1968

10 A G HANLON
Content-addressable and associative memory systems—A survey
IEEE Transactions on Electronic Computers August 1966

11 E L JACKS
A laboratory for the study of graphical man-machine communication
Proceedings Fall Joint Computer Conference Vol 26 1964 pp 343-350

12 C A LANG J C GRAY
ASP-A ring implemented associative structure package
Communications of the ACM Vol 11 No 8 August 1968 pp 550-555

13 R E LEVIEN M E MARON
A computer system for inference execution and data retrieval
Memorandum RM-5085-PR Rand Corporation Santa Monica California September 1966

14 J MINKER J D SABLE
Relational data system study
Auerbach Final Report—Contract F30602-70-0097 July 1970

15 P D ROVNER J A FELDMAN
The leap language and data structure
January 1968

16 P D ROVNER J C FELDMAN
An algol-based associative language
Communications of the ACM Vol 12 August 1969 pp 545-555

17 E H SIBLEY D G GORDON R W TAYLOR
Graphical systems communications—An associative memory approach
Proceedings Fall Joint Computer Conference Vol 33 Part I 1968 pp 545-555

18 I E SUTHERLAND
Sketchpad—A man-machine graphical communication system
Proceedings Spring Joint Computer Conference Vol 23 1963 pp 329-346

19 W R SUTHERLAND
On-line graphical specification of computer procedures
Tech Report 405 Lincoln Laboratory Cambridge Massachusetts 1966

20 A VAN DAM D EVANS
A compact data structure for storing, retrieving, and manipulating line drawings
Proceedings Spring Joint Computer Conference Vol 30 1967 pp 601-610

21 A WOLINSKY
Principles and applications of associative memories
Third Annual Symposium on the Interface of Computer Science and Statistics Los Angeles California January 1969

The hardware-implemented high-level machine language for SYMBOL

by GILMAN D. CHESLEY and WILLIAM R. SMITH

Fairchild Camera and Instrument Corporation
Palo Alto, California

INTRODUCTION

Through the years since the specification and development of the first computer systems, machine instruction sets have undergone the least modification of any aspect of these systems. A process of evolutionary growth through accretion of new components has taken place in contrast to the revolution in the programming and systems areas. One can point to stack commands, indexing, and microprogramming, all developed several years ago, and then the list of new concepts in instructions runs out. Similarly, the number of papers in the literature relating to the hardware implementation of programming languages is sparse. This seems anomalous considering that reductions in the cost of logic associated with the development of integrated circuits allow the possibility of implementing much more complex functions in hardware than with past design practices.

The efforts as reported in the literature break down into several attacks. The first approach, contemporaneous with the development of procedural languages, was the NCR 304 as reported by Yowell[1] and described in Sammet's "Programming Languages", and was an attempt to implement an Autocoder level in hardware, bridging the gap between machine languages and high-level languages. This direction of development has ceased, probably because of the explosive growth of high-level languages as the user language.

The second approach is that of directly implementing a severely restricted subset of an existing high-order language[2,3,4] although none of these proposals seem to have resulted in hardware. A more recent effort that was consummated is the microprogramming of EULER as reported by Weber.[5] Sometimes evolutionary steps are suggested[6] to aid the system in the compilation and execution of languages without attempting to encompass the whole effort in hardware. The Burroughs family of machines growing out of the B5000 is a very practical realization of this approach.

Another direction, epitomized by ADAM,[7,8] is that of designing a unique high-order language to be hardware implemented with characteristics selected to ease the implementation problem. Of course, once outside the realm of general-purpose high-order languages, many languages become interesting candidates for hardware because of features designed with existing system structures in mind.[9]

The approach taken with the SYMBOL system was to design a concise general-purpose language, well within the main stream of contemporary language development, but with novel features as appropriate and with as few linguistic limitations as possible. The SYMBOL language can be characterized as a high-level, procedural, general-purpose, hierarchical language with variable-length processing and storage to allow the explicit representation of structures that are variable in size, shape, and field length, and without type and size declarations since conversion and space management are handled automatically. It is a highly efficient language for the user since all of the language features contribute directly to execution of the program rather than to easing compilation or memory management. Most of the language is directly implemented in hardware and is now running in a multi-processor virtual-memory environment.

SYMBOL DESIGN GOALS

The specification of the SYMBOL* architecture[10] and the design of the SYMBOL* language was a con-

* Although some confusion might ensue from calling both the hardware system and the language by the same name, this seemed proper in this case where the language is hardware implemented and thus becomes a part of the specification of the system.

joint effort with each affecting the other. Very early in the project it was recognized that the language should be very concise to allow for a practical implementation. At the same time it was felt that it should be a general-purpose, procedural, "state of the art" language with no functional restrictions imposed on it to cater to its hardware implementation. Outside the language area, a design philosophy existed within the project that memory space management and data type testing and conversion could be done with hardware substituting for the traditional software handling of these areas. It was possible to combine these concepts by designing dynamic variable-length data space management and type conversion hardware and eliminating all size and type declarations from the language. Similarly, by specifying the most general hardware capabilities possible, language limitations such as the number of subscripts or the size of a name were eliminated. In addition, the language is highly hierarchical, much in the spirit of ALGOL, further simplifying the translation and error checking hardware, as well as contributing to the ease of programming in the language. To minimize the conversion problem, all number processing is done in a decimal, floating point, variable field length internal mode. A special binary form exists for storing status information.

HARDWARE REQUIREMENTS

The major requirement imposed by the language on the hardware was complete variability of data size. Thus, both string and arithmetic processing had to be capable of operating on variable length fields (any length strings are allowed although numbers are limited to 99 digits). The system is capable of storing and accessing variable length data. This capability serves both the processing or execution aspect of the language as well as the compiling and system functions (e.g., variable length identifiers).

A subscripting capability implies structures, and, in keeping with the general philosophy, it was decided that no limitations should be imposed. Thus, structure handling hardware allows complete variability in size and shape, regular or irregular, and complete dynamicism whereby a complete structure may be assigned to a substructure or field of any other structure. This requires extensive and efficient space allocation and recovery mechanisms but removes these concerns from the language portion of the SYMBOL system.

LANGUAGE HIGHLIGHTS

The semantics of the language will be covered in the next section and the syntax is contained in Appendix A. This section will attempt to convey a flavor of the language by introducing some of the more novel features. First, however, the question "Why a new language?" must be considered.

In attempting to implement in the hardware an existing high-order, main stream language, FORTRAN, ALGOL, BASIC, and a subset of PL/I would probably be the chief candidates. The successful implementation of SYMBOL suggests that any of these languages could also be hardware implemented and that their rejection becomes more a matter of choice than of necessity. BASIC was felt to be too primitive. ALGOL lacks input/output facilities and thus cannot be considered a complete language. FORTRAN was a strong contender because of its ubiquity. As the first high-level language, it is far from being a state-of-the-art language. Also, as constructed it is quite machine dependent (consider the problems of COMMON and EQUIVALENCE in a variable field length environment). Thus, although FORTRAN is familiar to most of the programming world, it was felt that choosing this language would be both retrogressive and overly restrictive. Finally, PL/I was eliminated because full PL/I was considered too complex for efficient and economic hardware implementation and because it lacked the overall simplicity desired for an end user orientated language. In summary, it was felt that the existing languages were biased too much toward a multitude of data types and hardware influenced elements and that for hardware implementation a cleaner language with a high degree of generality was desirable.

Therefore, a new language, SYMBOL, was developed with only two design requirements: first, it should be free of declarations about the nature of an identifier and, second, structures should be literally representable in the program and the external medium. Other features that developed as the design evolved were the complete variability of field and structure size and the execution time dynamicism of this flexibility.

Considering first the conventional aspects of the language, SYMBOL is a general purpose, procedural language fully in the main stream of language development. Expressions can contain the normal set of arithmetic, comparison, and binary operators having standard precedence relationships. Operands can be of various degrees of indirection: literals, simple variables, subscripted variables with any number of simple or complex subscripts, or procedure calls. The input/output statements are flexible but simple, reflecting the ability of the system to do data space management. A standard conditional statement is supplied, but with the final group of embedded statements closed with an END to avoid the dangling ELSE problem. A very flexible loop statement is available, allowing an unlimited number of loop clauses which have conventional default values if not present. It is a block structure language with default local identifiers and a floating global statement to move identifiers out to the enclosing block. Full procedural capability is available to the user with call-by- name parameters, automatic recursion as required, and a return statement for functional use which returns a generated value to the calling point.

Turning now to the more unusual characteristics of the language, its declaration-free nature is evidenced by the absence of size or type declarations or of implicit restrictions on field or processing length. This characteristic is built on the dual assumptions of automatic convertibility between data types and variable field length processing and storage. The premise here is that these are burdens best removed from the user and given to the hardware. Another salient characteristic of the language is its ability to represent structures literally. Thus, a vector containing the integers one through nine would be represented:

⟨1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9⟩

with the left and right angle marks and the vertical bar (called group and field marks, respectively) specifically reserved in the language for this purpose. Regular or irregular structures may be represented in this manner with any depth or shape and when stored in memory may be subscripted normally. For example, the three dimensional array A may be created by an initialization statement in the program:

A⟨⟨11 | 12 | 13⟩⟨21 | 22 | 23⟩⟨31 | 32 | 33⟩⟩;

and a reference to A[3, 2] would access the number 32 and a reference to A[2] would access the vector ⟨21 | 22 | 23⟩. The declaration-free and dynamic features of structure handling can be illustrated by considering the execution of the following assignment statement:

A[1] = ⟨A[3, 2] | A[2]⟩;

giving in A:

⟨⟨32⟨21 | 22 | 23⟩⟩⟨21 | 22 | 23⟩⟨31 | 32 | 33⟩⟩

There are essentially no limitations on this kind of structure manipulation. The same flexibility is allowed at the field level:

A[1, 2, 1] = | STRING LITERALS ARE BETWEEN FIELD MARKS |;

giving:

⟨⟨32⟨STRING LITERALS ARE BETWEEN FIELD MARKS | 22 | 23⟩⟩⟨21 | 22 | 23⟩⟨31 | 32 | 33⟩⟩

Referencing a structure point that does not exist causes the necessary structure to be created and filled with null fields. Thus:

B[5,3] = 444;

causes in B:

⟨| | | ⟨| | 444⟩⟩

It was mentioned before that structures may be initialized in the language by writing the literal structure with its identifier as a statement in the language. A switching structure may be similarly initialized by preceding the structure identifier with the reserved word SWITCH which causes the structure fields to be interpreted as labels rather than as data. For example:

SWITCH L ⟨X | Y | Z⟩;

sets up the switching structure L and:

GO TO L[2];

causes a transfer to a program point labeled Y. This allows computed transfers to be performed by making the subscript of the transfer statement an expression to be executed.

A natural concomitant to VFL storage is VFL processing. All processing operations, numeric or string, are performed serially on variable length data fields with the restriction that numeric processing is limited to a 99 digit normalized fraction portion of a floating point number. The creation of irrational numbers is a special problem for a VFL system. Consider processing ⅓ which would require infinite time and space to complete the operation. Two techniques are available in SYMBOL for controlling the length of numeric processing and thus processing speed and storage space. The first is a hardware register, referenced in the language by the reserved word LIMIT, which controls the truncation and rounding point of processing. VFL processing continues until the length of the fraction portion of the result reaches the value of the limit register, causing processing to terminate. The default value of the limit register is nine and it can be manipulated as an ordinary variable:

LIMIT = LIMIT-4;

would give LIMIT a value of five allowing processing to be faster, particularly for multiply and divide. Another method for controlling processing length is a precision mode whereby any number entering the system may be tagged with an EM (empirical) suffix indicating that the number is accurate only to the precision (length) supplied. Any number not so tagged is assumed to be exact; that is, of infinite accuracy and having implied low order zeros. The precision of the result of processing is equal to the least precision of either operand or the limit register. The result is empirical if either operand is empirical or if limiting takes place. Consider the following examples:

LIMIT=9; $\frac{1}{3}$	is .333333333EM
LIMIT=5; $\frac{1}{3}$	is .33333EM
LIMIT=5; $\frac{1}{4}$	is .25
LIMIT=5; 1/4.00 EM	is .250EM

Along with the normal operators, SYMBOL also includes format control operators to reconfigure data into a desired format using character-by-character testing, replacement, insertion, etc. Two operators are available, FORMAT and MASK, with the former operating on numeric data and the latter operating on string data and often used to convert between less dense code sets and the internal set. Many of the format features follow those suggested in Reference 11. To illustrate formatting, consider the following format control fields operating on the internally stored number 12345.6789:

CONTROL FIELD	RESULTS
+9ZV	+12345
−9Z.FD	12345.6789
10D.10D	0000012345.6789000000
ZZCZZZCZZZ.DD	12,345.67
ZBZBZBZBZBV	1 2 3 4 5
'$'*C***C***.DD	$***12,345.67
8B+D.FD$_{10}$+ND	+1.23456789$_{10}$+4
B−D.FD$_{10}$−DDMB	1.23456789$_{10}$04
B−FD.FDMB	12345.6789

The last two control field examples are wired into the format unit and are used for automatic conversion from numeric to string type. The floating form is used if the absolute value of the exponent is larger than nine. The fixed form is used for all other cases. The full set of control characters is tabulated in Appendix B but a brief description will be included here to flesh out the above examples. Two replicators are used: a one or two digit number causing the following control character to be repeated that many times and the "enough" replicator (F) causing replication until the fraction or integer portion of the data field is used up. Literal insertion (of a dollar sign) is shown in the sixth example and "B" causes insertion of a blank and "$_{10}$" causes insertion of the exponent tag. Three leading zero suppression characters are illustrated: "N" causes suppression only, "Z" causes replacement by blanks, "*" causes replacement by asterisks, "C" causes a comma or the adjacent zero suppression character to be inserted. The signs cause both signs or minus only to be transferred. Decimal point placement is controlled with "." and "V" with only the former causing insertion. The M character caused the EM tag to be generated if the number is empirical.

A link mechanism exists in the language by which expressions can be assigned to data space, and whenever that data space is referenced, the expression is executed in its place. The link is created by an assignment statement having the reserved word LINK preceding the expression. Links may be assigned to substructure fields and may be used as often as desired. As an example of how the link mechanism might be used, consider the following:

```
Employment History [X]=LINK Military Record;
        Credit Rating [Y]=LINK Military Record;
```

The following statement accomplishes the updating of both uses of Military Record:

```
Military Record [K]= |Discharge Date 11/1/67|;
```

The expression of the link operand can include a procedure call so that more complicated functions may be performed via the link mechanism. Variable references are also included in the link operand as part of expression syntax so that links may appear on the destination side of an assignment statement causing indirect assignments.

The SYMBOL language includes a named data input mode, activated by the qualifier DATA in the input statement, which allows data items to be input without referencing their identifiers in the input statement. Instead, the name is included with the data, similar in form to the initialization statements, and the input loading mechanism causes the current location of the identifier to be referenced and the incoming data assigned to that location. This feature allows dynamic interaction to take place between the program and the user with the data requirements being determined in real time during execution rather than when the program is written.

The "on" mechanism, similar to the one introduced in PL/I, has been provided in SYMBOL. On's are blocks of statements executed out-of-line by implicit rather than explicit reference. In the SYMBOL implementation, the reserved word ON is followed by a list of identifiers which precede the on body. These identifiers may refer to variables, labels, or procedures. The on block is executed out-of-line after storing to a variable, and before transferring to a label or a procedure. The reserved word INTERRUPT may also appear in the on list and refers to a hardware register of internal and external conditions that activate the on. Disable/enable statements are included in the language to deactivate/reactivate on action as desired. The on mechanism can be used to monitor program performance of selected items and can be subsequently deactivated after the debugging phase is complete.

A DESCRIPTION OF THE LANGUAGE

Components

Reserved characters are graphics having a unique interpretation to the system. They are also required as delimiters for multi-character names. Reserved characters may be categorized as follows:

Break Characters:	space	CR	TAB
Grouping:	( )	[ ]	⟨ ⟩
Separators:	, | ; : ≠		
Operators:	* + − / =		

Reserved names are contiguous strings of alphabetic characters having a specific meaning to the system and interpreted as a single entity. In fact, if a large enough character set were available, all of the reserved names could be replaced by reserved characters, albeit at the cost of readability. They can be grouped as follows:

Operators: GREATER, GTE, EQUALS, NEQ, LTE, LESS, BEFORE, SAME, AFTER, AND, OR, NOT, ABS, JOIN

Statement: INPUT, OUTPUT, GO, IF, LOOP, BLOCK, GLOBAL, PROCEDURE, RETURN, ON, ENABLE, DISABLE

Statement Clauses: LINK, TO, FROM, THEN, ELSE, END, INTERRUPT, FOR, WHILE, FROM, BY, THRU

Non-reserved names, sometimes called identifiers, are alphanumeric strings starting with an alphabetic character, isolated by reserved characters, and optionally contain embedded spaces allowing multi-word names. They are used to identify variables, procedures, or label transfer points in the program.

Any string of characters between field marks is a string literal. That is, it is interpreted as data rather than as a reference to data. Literals meeting valid number syntax need not be enclosed within field marks. The form for numbers is a string of digits with an optional decimal point and an optional exponent suffix consisting of the character "$_{10}$" followed by an optional sign and one or two digits.

Expressions are language elements used to generate or modify data, unlike the simpler elements discussed in the previous paragraphs which reference or represent data. Expressions may be used any place that a name reference is proper (except, of course, as an assignment reference) and are formed out of the following elements: reserved character operators, reserved word operators, variable operands, string literal operands, numeric literal operands, parentheses, and other components to be covered later.

Because the SYMBOL language is declaration-free, variable names do not have an inherent data type associated with them. The issue of data type can be minimized but not ignored altogether because the standard operators of a high-order language partition into at least two categories: numeric and string (Boolean and binary data types can be considered a subset of string type and that is the approach taken with SYMBOL.) Thus, in SYMBOL, the operators require a data type rather than the variables and an error is called when an operator/operand mismatch occurs rather than when data is stored to a variable as with most high-order languages. The error is indicated only if the automatic conversion process between the two data types is not successful.

The operators are partitioned as follows:

Numeric Operands: ABS,+,−,*,/, GREATER, GTE, EQUALS, NEQ, LTE, LESS

String Operands: BEFORE, SAME, AFTER, JOIN, AND, OR, NOT

Numeric Result: ABS,+,−,*,/

String Result: GREATER, GTE, EQUALS, NEQ, LTE, LESS, BEFORE, SAME, AFTER, JOIN, AND, OR, NOT

The operators AND, OR, NOT require strings of 0/1 characters only. Only JOIN produces a full string while the comparisons produce a 0/1 string character and the binary operators AND, OR, NOT produce a string of 0/1 characters. Other than the previously discussed characteristics, all operations—operands and results—are variable field length.

Mathematical usage has established a convention with respect to precedence among the arithmetic operators. Thus, A+B*C means A+(B*C) rather than (A+B)*C. It has been customary with artificial languages to extend this notion to give all of the operators in the language a relative precedence. For SYMBOL, the precedence relationships are as follows:

```
HIGH  ABS
 |    +(monadic), −(monadic)
 |    *,/
 |    +(dyadic), −(dyadic)
 |    JOIN
 |    comparison operators
 |    NOT
 ↓    AND
LOW   OR
```

Parentheses are used for vitiating the precedence relationships so that interpretations such as (A+B)*C can be made.

Simple statements

Statements in the language are basically used either to transfer or modify data stored in memory or to change and control the sequence of execution of these data statements.

The basic form of the assignment statement is:

identifier = *expression*

(Italics will be used throughout this paper to represent syntax elements. The full syntax of the language is given in Appendix A.) The value generated by the expression is stored in the variable represented by the identifier, replacing any previous value without regard for field size or data type.

The forms of input/output statements are:

INPUT[FROM *expression,*] *L, identifier*

OUTPUT[TO *expression,*] *L, expression*

(final bit of metalanguage: capital letter words represent the actual reserved word, brackets enclose an optional syntax element, the list operator *L* causes a list of one or more elements to be constructed and separated by the given punctuation mark).

The I/O statements cause data to be transferred between memory and external devices. If no TO/FROM clause is supplied, the transfer takes place between memory and the default device. Lists of data are terminated by an end-of-record mark (≠). Each input datum in sequence is assigned to a different variable, and output data generated by each expression is placed on a different line (if a printing medium is being used). A STRING qualifier may be used which prevents automatic spacing action on output or structuring on input. A DATA qualifier requires/causes identifiers to be associated with the data being transferred. EX/EM qualifiers may be used on input to cause numeric data to be packed and tagged unless specified otherwise externally.

The basic form of the transfer statement is:

GO[TO] *label*

Any statement may be preceded by one or more identifiers followed by colons. This establishes the statement as a program transfer point and the identifier as a label. The transfer statement is the basic language mechanism to cause a transfer of the sequence of statement execution, when encountered, from one point in the program to another.

The basic form of the conditional statement is:

IF *expression* THEN *body* [ELSE *body*] END

where the expression must generate a Boolean result. The body is a list of zero or more statements separated by semicolons, and may contain conditional statements, allowing nesting of conditionals. Either the first body or the second (optional) body is executed, depending on whether the expression is true or false.

Blocks

The basic form of a block is:

BLOCK *body* END

where the body can contain one or more global statements:

GLOBAL *L, identifier*

The block mechanism is a device for preventing interference and interaction between identifiers when several program segments are brought together to form a single package. It causes all uses of an identifier to be local to the body of the block containing the identifier and thus have no connection to uses of the same identifier outside the block.

The GLOBAL statement provides the capability of establishing an identity between the same identifier in different blocks by causing all identifiers in the global list to become known to the program body immediately outside the block (i.e., global statements carry identifiers up one block level). This allows two or more different blocks packaged together in the same program to communicate via common identifiers contained within their global statements. The following uses of identifiers are automatically local and thus these identifiers may not appear in a global list: initialization, switch, label, on list, procedure declaration and procedure formal parameter. Although global statements only carry identifiers up one block level, blocks can be nested to any depth and enough global statements must be supplied to boost identifiers up to a desired level.

Procedures

A procedure is a block with a name identifier. The basic form of a procedure is:

PROCEDURE *identifier; body* END

The reserved word PROCEDURE causes the procedure body to be established as a program block to be executed in place of and whenever the identifier occurs in the program. Referencing a procedure name causes an automatic transfer to the start of the procedure body, execution of that body, and an automatic return to the reference point when the body is completed. Procedures can only be entered by reference to the procedure name although they can be exited via a transfer statement. A procedure body is transferred around when encountered in-line. Procedures can be recursively entered when the procedure is referenced within its own block.

Procedures are often used in a functional manner—after execution of the procedure body there is a need to return a generated value to the calling point, replacing the procedure reference. The return statement provides this capability and has the following form:

RETURN *expression*

The return statement causes its expression operand to be calculated and returned to the calling point replacing the reference before terminating the procedure block. Assignment destinations may be returned to the calling point with the return statement, allowing procedure calls to appear on the destination side of an assignment statement. More than one return statement may be used in a procedure, and if none are encountered, the procedure terminates in a normal (non-functional) manner when the END is reached.

Parameters may be used to transfer constants, variables, expressions, or labels (syntactically called references) between the calling point and the body of a procedure by following the procedure name at the calling point with a list of these items surrounded by parentheses. These "actual parameters" are correlated one-to-one to a list of "formal parameter" identifiers immediately following the procedure identifier and preceding the procedure body. Whenever the formal parameter is encountered in executing the procedure body, it is replaced by its corresponding actual parameter which is executed in its place. This is referred to as a "call-by-name" parameter correlation. The complete form for procedures is shown in Figure 1.

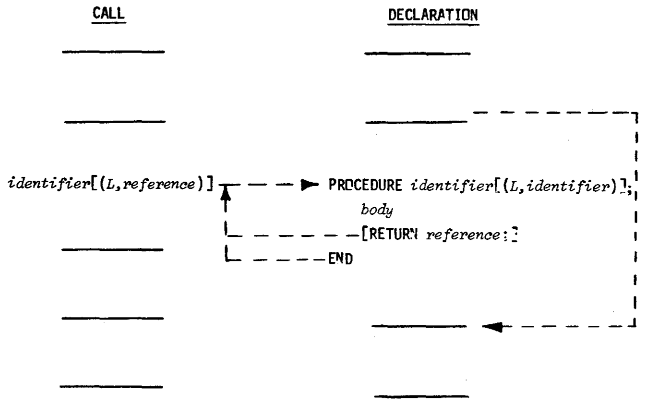

CALL
DECLARATION
identifier[(L,reference)]
PROCEDURE identifier[(L,identifier)];
body
[RETURN reference;]
END

Figure 1—Procedure form

The implementation of the parameter mechanism in the SYMBOL system allows a "procedureless parameter" to be defined. It is created by an assignment statement of the following form:

identifier = LINK *reference*

The parameter mechanism is activated whenever the identifier appears in any context in a program causing the reference to be executed in its place. The link assignment is a dynamic (execution time) action and has no connection with a procedure.

ON block

The on mechanism is a technique whereby a block of programming may be executed out-of-line by implicit rather than explicit (as with procedure) reference. The form of the on block is as follows:

ON *L, identifier; body* END

where the identifiers may refer to data variables, procedures, or labels. An on block is automatically executed by the following actions to its list of identifiers: after storing to a data identifier, before execution of a procedure, before transfer to a label. Since an on block is called implicitly it has no need for parameters or the return mechanism. After execution of the on block, processing continues at the point where the on block was activated.

The reserved word INTERRUPT may appear in the list of on conditions and refers to a hardware register that may be set by a number of external and internal conditions, thus causing that on block to be executed.

The disable/enable statements are used to control the deactivation and reactivation of on identifiers. The form is as follows:

DISABLE *L*, *identifier*
ENABLE *L*, *identifier*

When an identifier is disabled, the on block is no longer activated by a use of that identifier.

Loop

The loop statement provides a convenient mechanism for repeating the execution of a group of statements while varying an (optional) variable within the group until a given criterion is met which causes an exit from the loop. Loops may also be exited via a transfer statement within the loop. The form of the loop is as follows:

LOOP [*identifier*][*L*, *clause*]; *body* END

where the clauses are as follows:

FOR *L*, *expression*
[FROM *expression*][BY *expression*][THRU *expression*]

where the latter elements can be in any sequence and have default values of 1,1, infinity respectively. Either clause can be terminated by an optional "while" element.

The loop body is repetitively executed with the loop variable (if present) taking on the designated value, until the clause is satisfied or the while modifier becomes false. Then the next clause in sequence is executed, executing the body each time, until no more clauses remain, in which case the loop is terminated. Variables in the loop header may be modified at any time by actions within the loop body.

Structures

A structure is a group (or vector) of one or more elements consisting of basic elements or structures. Basic elements are separated by field marks (|) and groups are contained within group marks (< >). Structures need not be regular in size, shape, or data type and may vary dynamically. They may be nested to any depth. Depending on their program context they may contain data, labels, or expressions.

Structure points may be referenced on either side of an assignment statement or may be written with program segments contained within the group marks on the data side of an assignment statement. Explicit structures may be written and used as data. In this structure assignment case, each program segment within the literal structure is executed, a result is generated, and then the whole structure is assigned to the destination point. Using the structure marks, literal structures may be created at the input device and subsequently input to a variable. Similarly, structured variables may be directly transferred to an output medium.

Structures containing data or label identifiers may be given an initial value by writing them in the program, preceded by an identifier, as a separate statement. To initialize label structures, the reserved word SWITCH must also precede the structure identifier. Simple, non-structured variables may be initialized by the variable identifier followed by the literal data enclosed within field marks. Initialization takes place only once in a program when the initialization statement for a variable is first encountered. Subsequent passes through the program transfer around the initialization statements.

Full structures are accessed by reference to the structure identifier without subscripts. Substructures are accessed by enough subscripts to select the desired substructure level. Most substructure references will be with enough subscripts to select the field level. Subfields are accessed by a bound pair (expressions separated by a colon) as the last subscript after the field level. Characters are selected from the field starting at the left value of the bound pair and continuing to the number of characters indicated by the right value of the bound pair. All substructures above the field level are complete and valid structures and thus cannot be used in expressions (automatic vector or array operations are not included as part of the hardware language). Basically, structures can only be assigned or transferred between the external medium.

There are essentially no limitations to structure manipulations. Fields or vectors can be extended, contracted, or removed by assignment. Fields or vectors created by reference are filled with nulls. The only restriction is that data cannot be destroyed by *referencing* below an existing field level although it is legitimate to destroy it by assignment. The reserved word IN tests whether a particular structure point referenced by a subscripted variable exists, returning either a true or false value. Oversubscripting by an in reference does not cause null structure to be generated.

A switch structure containing labels can be the subscripted variable operand of a transfer statement, allowing a computed go to be performed.

Data Control

Two techniques are available to control the length of variable field length arithmetic processing. The first is a precision limiting register, acting as a hardware variable in the language and referenced by the reserved word LIMIT. The default or standard value is 9 causing all numeric processing to cease at a maximum value of 9 digits of precision. Any other positive value up to 99 may be assigned to the limit register.

The second technique tags (directly or with an input qualifier) all numbers entering the system with an EM (empirical) suffix which indicates that they are accurate only to the number of digits (precision) supplied. All numbers not specified as EM are tagged by the system as EX (exact) and these numbers are assumed to possess an infinite string of low order zeros. These tags affect processing in an obvious and natural way and combine with the limit register to give precision controlled processing with a size override for the sake of efficiency. Any exact calculation terminated by the limit register produces an empirical result (when limiting takes place an internal status indicator is set which may be accessed and reset by the reserved word LIMITED). With this system, when the accuracy of an algorithm is not taken into account, meaningless results will be indicated by the precision shrinking to zero.

Many times a standard data format, automatically provided by the system will be sufficient. Often, however, an application demands a particular format, and two operators are provided by the SYMBOL language to control format. The FORMAT operator reconfigures numeric data and the MASK operator reconfigures string data. They both have a control operand and a data operand with the characters of the control operand operating on (selecting, inserting, replacing, etc.) characters of the data operand. Many of the characters of the format operator are concerned with check and payroll formats while many of the mask characters cause conversion between special data modes. The full set of format and mask control characters is summarized in Appendix B. To further aid in format control, the output statement has a STRING mode option which suppresses the normal carriage return spacing features associated with the output statement.

Software patches to hardware

As a hardware-implemented language, SYMBOL is vulnerable to the criticism of inflexibility. To obviate this objection, hardware breaks to system software were included in the language as well as the ability to control the basic memory operations of the system. The reserved word TRAP is a separate language statement which, when encountered, causes the hardware Translator to interrupt the translation process and turn system control back to the master hardware scheduler. At this point, if desired, a system program would be executed. The reserved word SYSTEM performs a similar function for the execution unit (the Instruction Sequencer), causing a system interrupt when encountered. It has no effect on the Translator, passing through the translation process without change.

The Privileged Memory Operations (PMO) are language statements causing actions identical to the actual hardware memory operations of the system which are described in the companion paper on system architecture. They are of the nature: "*fetch* the word at the address I give you and *follow* any indirect links to give me the next address in sequence" or "*store* the word I give you into the address I give you and *assign* new space and link it in if no following space exists and return the next address to me." The operands of the PMO's, as seen by the user, are hex coded character strings (so that they can be manipulated as valid data) of the data and address fields and are automatically packed and unpacked by hardware. Since all the status of the system is ultimately kept in memory, the PMO's have the ability to control and mimic all aspects of the system, albeit at a slower rate than hardware can accomplish the same task.

Since SYMBOL is a time-sharing system, it is necessary to prevent programs from inadvertently interfering with each other. As a hardware language, the protection is inherent: until now, there are no language mechanisms by which one program can interfere with another since all data is manipulated under symbolic names rather than as memory addresses. The introduction of memory operations into the language recreates this danger. Thus, they are designed as privileged operations which can only be used under special restrictions. Basically, one must either be a system programmer or be using a system procedure to utilize the PMO's.

CONCLUSIONS

Hopefully, the SYMBOL language has been described in enough detail to give the reader familiar with programming languages the flavor of the language and to give the language expert enough information to actually use the language. As yet, few perfomance figures are available since the project is just entering the operation and evaluation phase, although preliminary measurements indicate potential compiling rates of hundreds of thousands of statements per minute. Of course, per-

APPENDIX A

SYMBOL SYNTAX

```
digit :: = 1|2|3|4|5|6|7|8|9|0
letter :: = A|B|C|---|Z|a|b|c|---|z|
character :: = _any character except ‡_
break-char :: = _a carriage return, tab or space_
identifier :: = letter[[letter|digit|break-char]...(letter|digit)]
label :: = identifier
decimal-number :: = digit...[.] | [digit...] . digit...
exponent-part :: = ₁₀[+|-] [digit] digit
number :: = decimal-number|exponent-part|decimal-number exponent-part
string-number :: = [+|-] [(digit|,)...] number [EX|EM]

string :: = _sequence of zero or more of any characters except < > | and ‡_
field :: = string-number|string
data-field :: = "|" field "|"
named-data :: = identifier (data-field|data-structure)

ds :: = [L "|" field]
data-structure :: = <ds [data-structure ds]...>
ls :: = [L "|" label]
label-structure :: = <ls [label-structure ls]...>
as :: = [L "|" exp]
assignment-structure :: = <as[assignment-structure as]...>

subscription :: = identifier "[" L,exp "]"
subfield :: = identifier "[" L,exp:exp"]"
designator :: = identifier|subscription|procedure-call|LIMIT
procedure-call :: = identifier ["(" L, [reference] ")"]
constant :: = data-field|number
value :: = designator|subfield|constant|IN(subscription|subfield)|LIMITED
reference :: = designator | exp | label
arithmetic-op :: = +|-|*|/
string-op :: = JOIN|FORMAT|MASK
binary-op :: = AND|OR

arithmetic-relation :: = GREATER[THAN]|GTE|EQUALS|NEQ|LTE|LESS[THAN]
string-relation :: = BEFORE|SAME|AFTER
relational-op :: = arithmetic-relation|string-relation
dyadic-op :: = relational-op|arithmetic-op|string-op|binary-op
monadic-op :: = +|-|ABS|NOT

exp :: = value|"("exp")"|monadic-op exp|exp dyadic-op exp
```

```
assignment-element :: = assignment-structure|exp|LINK reference
assignment-stm :: = L,designator = assignment-element
go-to-stm :: = GO[TO] (label|subscription|procedure-call|identifier)
call-stm :: = [CALL] procedure-call
break-stm :: = PAUSE|SYSTEM|TRAP
dummy-stm :: = [CONTINUE]
comment-stm :: = NOTE_any characters except ;_

output-stm :: = OUTPUT ([TO exp,]L,exp|STRING[TO exp,]L,exp|DATA[TO exp,]
                L,identifier)
input-stm :: = INPUT([EX|EM][FROM exp,]L,designator|STRING[FROM exp,]
               L,designator|DATA[FROM exp])
list-data :: = L ‡ (field|data-structure)‡
string-data :: = L ‡ [character...]‡
self-defined-data :: = named-data...‡

initialization-stm :: = named-data
switch-stm :: = SWITCH identifier label-structure

for-clause :: = FOR L, (exp|exp WHILE exp)
step-clause :: = (FROM exp|BY exp|THRU exp|WHILE exp)...
loop-header :: = LOOP [designator] L, [for-clause|step-clause];
loop-stm :: = loop-header body END

on-element-list :: = L,(identifier|label|INTERRUPT)
on-header :: = ON on-element-list;
on-control-stm :: = (DISABLE|ENABLE) on-element-list

return-stm :: = RETURN [reference]
procedure-header :: = PROCEDURE identifier ["("L,identifier")"];
conditional-stm :: = IF exp THEN body [ELSE body] END
scope-stm :: = GLOBAL L,identifier
block :: = (BLOCK|on-header|procedure-header) body END
stm :: = label: stm|assignment-stm|go-to-stm|call-stm|break-stm|dummy-stm
                |comment-stm|output-stm|input-stm|initialization-stm|switch-stm
                |loop-stm|on-control-stm|return-stm|scope-stm|conditional-stm|block
body :: = [L;stm]
program :: = body ‡
-------------------------------------------------------------------------------
privileged-stm :: = (fetch|store|delete|assign)
                    (identifier|subscription)

fetch :: = FF|FR|FL|FD|FT
store :: = SA|SO|SI|SD|ST
delete :: = DE|DS|DL
assign :: = AG|IG
```

METALANGUAGE

(X|Y)	Select one alternative from group
[X|Y]	Select zero or one from group
X...	Repeat zero or more times
L XY	List of Y's separated by X's
_X_	A comment
"X"	Not a metasymbol

Appendix A

formance in the SYMBOL system is an admixture of two opposing factors, hardware implemented functions and an extremely dynamic language, and it will be very difficult to separate these two contributions. It is probably true that the SYMBOL language, if compiled and executed on a conventional architecture, would be as slow as any other high-level language having comparable execution time dynamicism coupled with automatic data space management. Default PL/I would probably provide a fairly relevant basis for prediction. As implemented in the SYMBOL hardware, however, any task requiring the variable length processing and storage or the dynamic structure features of the language should show a considerable performance gain over conventional software/hardware systems.

Early experience within the SYMBOL project has shown the language to be an extremely simple one to learn and use, considering its powerful capabilities. The use of structures seem to indicate that their great flexibility and dynamicism completely eliminate the need to develop exotic strategies for data base management. For example, hash sorting buckets in SYMBOL can simply be vectors of an array since vectors automatically expand as the subscript increases. Since collisions are of little consequence, scatter storage techniques[12] need not be complicated. Little practical experience has been gained with the precision controlled processing mode although it is expected to be a boon to numerical analysts as well as a source of surprise to less sophisticated users who have never considered the question of numerical accuracy since conventional systems tend to mask accuracy loss.

SYMBOL is a break with the existing trends in language development, as epitomized by ALGOL 68 and full PL/I. That is, the SYMBOL system hardware takes over from the user and performs the complete memory management task including all space allocation and recovery, structure expansion and contraction, and virtual memory paging between disk and main memory. The other languages do the opposite by

APPENDIX B

FORMAT/MASK OPERATORS

	CODE	ACTION TO RESULT	# CHAR. SELECTED	# CHAR. IN RESULT	REPLICATION*	RESTRICTIONS
FORMAT	Q'α'	Literal string α inserted if source negative; otherwise nothing.	-	0,nα	NO	
	R'α'	Literal string α inserted if source positive; otherwise nothing.	-	0,nα	NO	
	-	Space (positive) or - (negative) inserted as first non-space char.	-	1	1	
	+	Sign inserted as first non-space character.	-	1	1	
	N	Suppress leading zeros.	1	0	n	
	Z	Spaces replace leading zeros.	1	0,1	n	
	*	Asterisks replace leading zeros.	1	0,1	n	
	C	Comma or zero-suppression character inserted.	0	1	NO	
	$	Dollar sign inserted as first non-space character.	0	1	1	
	$_{10}$	Exponent tag inserted.	-	1	1	$_{10}$ must be followed by correct exponent format.
	X	EX (exact) or EM (empirical) inserted.	-	2	1	
	M	Null (exact) or EM (empirical) inserted.	-	0,2	1	
	.	Decimal point inserted. Also serves as reference point for format field.	-	1	1	
	V	Serves as reference point for format field.	-	0	1	
MASK	A α	Source character inserted unless same as α	1	0,1	F,n	(Space) default.
	E	Expands source character to two Symbol hex characters.	1	2	F,n	(:: ::) default.
	H	Converts two hex characters to one standard character.	2	1	F,n	Result must have leftmost bit false.
	U	Unpacks hex character to four binary (0/1) characters.	1	4	F,n	(0000) default.
	P	Packs four binary characters to one hex character.	4	1	F,n	0,1 source only; (::) default.
	C	Produces four digit count of source length.	ALL	4	F	C must be preceded by F.
COMMON	D/S	Digit (FORMAT)/Character (MASK) transferred.	1	1	F,n	(zero/space) default.
	I	Ignore source character.	1	0	F,n	
	B	Space character inserted.	0	1	F,n	FB with MASK only.
	/	Carriage return character inserted.	0	1	n	
	'α'	Literal string α inserted.	0	nα	NO	Quote represented within literal by doubling.

*Replication Classes

n	Codes indicated by n may be preceded by a 1- or 2- digit number. Effect is of repeating the code that many times.
F	Codes indicated by F may be preceded by an F, which causes the code to be repeated until the source field is used up. In FORMAT specifications, each F applies only to one side of the decimal point (i.e., integer <u>or</u> fractional portion is used up).
NO	No replication allowed but code can be used more than once in a field.
1	No replication and can be used only once in a field.

NOTE: Standard Format Conversion:

B-FD.FDMB (Fixed - exponent one digit) or

B-D.FD$_{10}$-DDMB (Floating - exponent two digits)

Appendix B

giving the user more and more control over space management and, in fact, require a fairly high level of expertise in arranging and programming problems for efficient use of space. Herein lies the sticking point: will future users of high-level language systems want more or less control over the mechanics of their data base? The SYMBOL philosophy is that user data space management will go the way of manual transmissions on automobiles: fine for special purposes but the average user will be willing to give up some performance and flexibility for ease-of-use.

In a way, the situation is analogous to the introduction of high-level languages with FORTRAN. At the time there was great concern within the FORTRAN project that the system must have an execution time performance roughly equivalent to a machine language program for the same task. After introduction of the system it was soon found that this criterion was much less important than was initially thought because the programming leverage given by a high-level language more than compensated for execution inefficiencies. The same argument applied to space management should be even more persuasive today with the ratio between programming and hardware costs undergoing a dramatic reversal.

REFERENCES

1 E C YOWELL
A mechanized approach to automatic coding
Automatic Coding Journ Franklin Inst Monograph No 3 Phila Pa April 1957 pp 103-111

2 J P ANDERSON
A computer for direct execution of alogrithmic languages
Proc EJCC Vol 20 1961 pp 184-193

3 A J MELBOURNE J M PUGMIRE
A small computer for the direct processing of FORTRAN statements
Computer Journ Vol 8 No 1 April 1965 pp 24-27

4 T R BASHKOW A SASSON A KRONFELD
System design of a FORTRAN machine
IEEE Trans Elec Comp Vol EC-16 No 4 Aug 1967 pp 485-499

5 H WEBER
A microprogrammed implementation of EULER on the IBM System/360 Model 30
Comm ACM Vol 10 No 9 Sept 1967 pp 549-558

6 W M McKEEMAN
Language directed computer design
Proc FJCC Vol 31 1967 pp 413-417

7 A P MULLERY R F SCHAUER R RICE
ADAM-A problem-oriented symbol processor
Proc SJCC Vol 23 1963 pp 367-380

8 A P MULLERY
A procedure-oriented machine language
IEEE Trans Elec Comp Vol EC-13 No 4 Aug 1964 pp 449-455

9 K J THURBER
System design of a cellular APL computer
IEEE Trans Comp Vol C-19 No 4 April 1970 pp 291-303

10 W R SMITH et al
SYMBOL: A large experimental system exploring major hardware replacement of software
This volume

11 D E KNUTH
A proposal for input/output conventions in ALGOL 60
Comm ACM Vol 7 No 5 May 1964 pp 273-283

12 R MORRIS
Scatter storage techniques
Comm ACM Vol 11 No 1 Jan 1968 pp 38-43

SYMBOL—A major departure from classic software dominated von Neumann computing systems

by REX RICE and WILLIAM R. SMITH

Fairchild Camera and Instrument Corporation
Palo Alto, California

INTRODUCTION

The prime goal of the SYMBOL research project was to demonstrate, with a full-scale working system, that a very high-level, general-purpose, procedural, "state of the art" language and a large portion of a time-sharing operating system could be implemented directly in hardware and achieve a significant increase in overall computational rates. A further objective was to create hardware design and construction techniques which could be easily applied by a small number of people to implement such a system in a reasonable time and at a relatively low cost. Although this was a research project, there was a high dedication to developing automation, hardware, test equipment and documentation to support the project. The name SYMBOL was chosen to signify direct hardware symbolic addressing.

Another goal of the project was to develop hardware/software algorithms which directly aid a casual as well as a professional user working with non-numeric data. Particular attention was given to the manipulation of data structures for file maintenance coupled with powerful field and character manipulating instructions.

The general-purpose SYMBOL language[1] was developed after studying a large number of modern languages including ALGOL, FORTRAN, PL/1 LISP and EULER. It was decided early that the new language had to perform all useful operations on application problems without being cluttered with machine-dependent operators; also, since it was a research project there was no strong necessity to be compatible with available programs written in other languages. On the hardware side it was decided that no appreciable restriction to the language would be permitted and that hardware would have to be invented to match the language. As the project progressed it became desirable to include conversational-mode multiprocessing and multiprogramming as well as source language text editing.

The architectural philosophy of the SYMBOL system led to hardware implementation of a variety of features which have been and are software functions of current systems. Some of the interesting features directly implemented in hardware in SYMBOL are:

Dynamic Memory Allocation
Dynamic Memory Reclamation
Dynamically Variable Field Lengths
Dynamically Variable Structures
Automatic Virtual Memory Management
Automatic Data Type Conversion
Automatic Time-Sharing Supervision
Direct Symbolic Addressing
Precision-Controlled Arithmetic Processing
Direct Hardware Compilation
Alphanumeric Field Manipulation
Direct Text Editing

PROJECT HISTORY

From the beginning, this research project was committed to producing a real and functioning system rather than a simple simulation. A brief outline of the project history appears below. It should be noted that considerable emphasis was placed on hardware design and implementation techniques. In fact, the complete project was treated as a "closed system" where no item (such as a user with his application) was considered separately from the language, operating system, or hardware.

1963-64: Hardware Technology Development
Fairchild CTμL Family for Circuits
Power Distribution Techniques

Large Two-Layer Printed Circuit Boards
Dual In-Line Package Invented

1964-65: Data Flow Model[2] Process Unit Construction
Two-Layer Printed Circuit Board System Interconnections
Cam-Operated, Zero Insertion Force Contacts
High-to-Low Order Variable Length Arithmetic Processing
Use of CTμL

1964-67: Language Development for SYMBOL
Dynamically Variable Field Length Data
Structured Data Literally Represented
Complete Syntax and Semantics
Source Text and Program Editing
Definition of Operating System Hardware/Software

1965-69: Computer-Aided Engineering Design Package Development
Equation Expansion and Checking
Timing, Loading and Offset Checking
System Logic Function Factoring
System Interconnections
Placement
Wire Routing

1964-70: Hardware Development for SYMBOL
December, 1967—Logic Flow Charts Complete
December, 1968—Partial Construction and Testing Started
December, 1969—Fabrication Complete and Debugging Started
June, 1970—SYMBOL Hardware Operational
January, 1971—SYMBOL Delivered to Iowa State University

SYMBOL FROM A USER'S VIEWPOINT

Much of the written or printed communication in our society is conducted on what may be described as "Typewriter English." We communicate with upper and lower case alphabet, decimal numbers and a group of commonly accepted special symbols. These communications are generated in all sorts of forms and may vary from handwritten, to typewriter produced or to automatic computation output, etc. Several features become evident on examination of the preferred communication forms.

Variable field length

People use a free flowing written communication style which requires a complete variability in the number of characters in a word (English that is). The information contained in groupings of characters or words (i.e., fields) is also of variable length and can change as manipulation upon that field occurs. Although past computing practice has developed abbreviated mnemonics and codes to fit fixed word machines and to reduce writers cramp, it may be noted that deciphering one's program or data base a year later is often laborious and is especially difficult for a third party. To ease this problem in SYMBOL, techniques were developed which allow complete and dynamic variability in the length of a string of characters used as a name, a word or a field. Each user is under no software/hardware constraint and may use any word or field size he wishes. He never needs to predetermine field size by declarations. In both the source program and in the data base this variability is provided.

A new character called a Field Mark was introduced to define the start and end of a variable length field in the data base and for non-numeric literals in the program. The Field Mark is a long vertical line and was chosen so as to be easy to see and not to conflict with commonly used characters. The field mark is entered from keys, or automatically generated by the system, as the data base is developed.
Examples:

```
| 123 |
| This is a field |
| A field may be as long as desired |
| A field may be short |
| A field may be short and then expanded |
| or contracted |
```

Decimal arithmetic

People have been raised with the decimal system. Even though they adapt to computers and the use of the hex/binary system, it is unnatural. The SYMBOL system accepts either fixed or floating point decimal numbers with positive or negative mantissas varying from one to ninety nine digits with or without a decimal point. Exponents, if any, may contain a plus or minus sign and may have up to two decimal digits. The operands for arithmetic may be both fixed, both floating or mixed. The system hardware automatically converts them to an internal floating point form and computes a left justified floating point decimal result. Since arithmetic computations on variable field length numbers can produce even longer results a LIMIT register is provided to truncate the resulting computations at a desired number of significant digits. The LIMIT value may be dynamically changed by the user's program as he explores the effect of precision versus computing time on his solution. The machine identifies truncated results with an automatically generated symbol "EM" following the number.

Examples:

Limit = 5
  $1/3$ = .33333EM
Limit = 25
  $1/3$ = .3333333333333333333333333EM
Limit = 10
  $-7000.00\text{EM}/-.3_{10}1 = 2333.33\text{EM}$

All numbers without the EM tag are assumed to be exact with an unlimited number of trailing zeros. Note that in the last example the first operand limited the precision of the result in contrast to the LIMIT register limiting the first two cases.

Character manipulation

In a string and character oriented environment output for human consumption is of paramount importance. The easy and efficient manipulation of data within fields both for data base computations and for efficient report generation must be provided. SYMBOL includes direct hardware implementation of two operators to reconfigure string fields into desired formats. The FORMAT operator is used to manipulate numeric operands by applying a "pictorial string" mask against the operand field. The operation proceeds on a character by character basis from left to right (i.e., high order to low order). Zero suppression, left justification, right justification, decimal point alignment, floating dollar sign, check protection and comma insertion are some of the operations provided. The MASK operator provides a similar manipulation capability for alphanumeric string fields. Literal string insertion, character deletion, character insertion, field length counts, carriage control and space insertions, are some of the operations provided. The hardware implemented FORMAT and MASK operators have demonstrated exceptional performance when compared to conventional software procedure controlled character manipulation.

FORMAT examples follow for 12471.2342 which is the operand field:

FORMAT Pictorial String	Result Field
DDDDDDD.DDD	0012471.234
$ZZZCZZZCZZZ.DD	$12,471.23
'$'B***C***C***.DD	$*****12,471.23
$+D.5D_{10}+DD$	$+1.24712_{10}+04$

MASK examples follow for | 6N491-XMT | which is the operand field:

MASK Pictorial String	Result Field
IISSS	491
SBSBSSSBISSS	6 N 491 XMT
FC	0009
5F' PART TYPE'	6N491 PART TYPE

The first FORMAT example illustrates control of the number of digits (D) in the result. The next uses zero suppression (Z), floating dollar sign ($), and conditional comma insertion (C). The next shows check protection (*) with conditional comma insertion and the last shows floating point notation ($_{10}$) and picture replicator usage. The MASK examples illustrate character ignore (I), blank insertion (B), field length counting (FC), and literal insertion ('—'),

Data structure manipulation

In both manual and in automatic record handling the difficulty of generation and maintenance of data bases determines the usefulness and efficiency of the total process. In this area SYMBOL departs further from tradition and places all field, group and structure delineation directly in the data base. This is in contrast to having most delineation present in the addressing portions of object codes in more conventional systems. Complete dynamic variability (at execute time) of field size, vector size and structural configuration is provided. This is directly implemented in hardware to provide for competitive execution rates and more importantly to relieve the programmer from any necessity of declaring data base sizes and attributes. Early work in this area was reported in a research study on a system called ADAM.[3] SYMBOL extends these concepts to allow open-ended dynamic data base flexibility and to our knowledge, for the first time, resulted in full scale hardware for supporting these features.

Group marks (i.e., ⟨ ⟩) are added to the character set to provide field grouping (i.e., vectors) in the data base. The following examples illustrate the use of field and group marks in the data base. These delimiters are also used in the instruction stream to define various items such as constants, literals, structures, etc. Consider the following examples of data structures.

String Fields:

| Joe Doaks |
| Flight No. 306, SFO to JFK |
| Cape Code House, 2 Bedroom, 1 Bath, Living Room, Dining Room, Fireplace, Basement, 2 Car Attached Garage |

Numeric Fields:

| 374.12795368448793$_{10}$-72 |
| -1234.5 |
| $*****1,576,265.46 |

Dynamically Varying Structures (Time Sequence)

```
⟨123 | 456 | 789⟩                            --INITIAL Structure
⟨123⟨456 | ABC⟩789⟩                          --Modified Once
⟨123⟨456 | ABC⟨DEF | GHI | JKL⟩⟩789⟩        --Modified Twice
```

The last sturcture above can be visualized as:

```
⟨123                                789⟩
      ⟨456 | ABC                   ⟩
                 ⟨DEF | GHI | JKL⟩
```

Few limits are placed on data structures. Fields may grow to the size of main memory. No restriction is placed on the depth of nesting in a structure.

Operating system complexity

SYMBOL directly implements almost all of a time-sharing and multiprogramming system directly in hardware. Further, the internal machine language is the source language. This direct implementation of source language significantly reduces the layers of software normally found between source and object codes. This in turn reduces the "hidden rules" or system dependent constraints which plague the casual or professional user of the system. These features combine to make the system exceptionally easy to use for problems where data base manipulation on alphanumeric data is of prime consideration. Since the system provides powerful arithmetic operations on variable field decimal data it is also excellent for most engineering and scientific uses.

A valid point can be made that if all language is "hard wired" then error correction, extensions, new language elements, etc., are difficult to achieve. This need was recognized in SYMBOL and features are provided in the hardware/software interface to allow expansion or extension. First, interrupts are provided for traps at hardware compile and/or execution time. These interrupts allow a break-out from the high-level language and may call a "system program" to perform some desired task. Second, privileged memory operations are provided which allow a privileged program to initiate directly any memory operation available to the hardware. A combination of regular and privilege operations may be used to create new language elements and/or new macro instructions.

Facilities for "file management," for example, are supported with software that uses a combination of ordinary language and privileged instructions. These algorithms could have been directly implemented using SYMBOL techniques, but they were not sufficiently clear to be stabilized and did not need the higher performance of direct implementation. Further research evaluating th's type of tradeoff would prove most interesting.

The objective in SYMBOL was to support the high duty cycle and basic features of the operating system in hardware. Many of the algorithms are hardware executed with software established parameters so as to obtain higher performance without loosing the needed flexibility. This support provides a significant simplification in the overall operating system.

SYMBOL FROM THE ARCHITECTURAL VIEWPOINT

Studies of modern large computer systems have shown that a large portion of the logic in the main frame hardware is idle most of the time. Some of the largest systems have achieved more parallelism using a main CPU and several auxiliary smaller computers to handle input and output tasks, etc. (i.e., CDC 6600 and 7600). The GAMMA 60 developed earlier by the Bull Company in France departed from tradition by exploring the running of several problem segments in small units each containing sufficient registers and logic to operate autonomously for short computation sequences. The matrix-type systems such as the ILLIAC IV allow many identical, or nearly identical, programs to operate simultaneously on the same type of large problem.

The SYMBOL system architecture shares some similarities to and has some differences from these systems. SYMBOL achieves parallelism and execution efficiency by using time-sharing with multiprogramming and multiprocessing done through seven simultaneously operating autonomous processing units sharing a common virtual memory. The hardware contains a large main memory, used as a virtual memory buffer, and the "Autonomous Functional Units" (AFU). In the tradition of GAMMA 60 the autonomous units have sufficient logic registers, control, etc., to perform sequences of operations without being under control of a conventional CPU. SYMBOL departs from previous systems at this point. Each AFU is dedicated to a portion of the computing process and its logic (i.e., instruction sequences) is hard wired so that source language is essentially machine language.[1,4] The gross block diagram in Figure 1 shows the common communication bus structure of the system. Each AFU is a special purpose processor.

The architecture is designed so that the time-sharing supervision is managed by the System Supervisor (SS). Provided the job load permits, each AFU can be performing its tasks on a different users job while simultaneously sharing virtual memory with other AFUs, The SS maintains queues of jobs ready for each AFU and schedules the system tasks. Communication is conducted along the main bus and by several local buses.

A unique feature of the system is automatic and dynamic hardware memory management. The assignment, and access of memory is done by the Memory Controller (MC). With memory control as a service function the logic for generating and manipulating data is distributed to the various other AFUs.

Figure 2 presents a summary of the functions hard wired into each of the AFUs. The Interface Processor (IP) provides source text editing without use of the Central Processor (CP). The Translator (TR) converts the source language into a reverse Polish string form ready for processing by the CP. Below each box a number gives the count of large printed circuit cards, each containing 160 to 200 integrated circuits, used for the AFU. This gives the reader a rough feel for the relative sizes of the units.

SYMBOL FROM THE PHYSICAL VIEWPOINT

At the start of the SYMBOL project it was decided to use a functionally-factored, bus-oriented system. Preliminary studies showed that large printed circuit boards (i.e., 12″×17″) with about 200 integrated circuit packages (providing the equivalent of about 800 two-input gates) would be sufficient to minimize system interconnections (Figure 3).[5]

It was also obvious that two-layer printed circuit boards were much less expensive than multi-layer boards. A substantial and successful effort was mounted to develop placement and wire routing algorithms and to obtain a computer-aided engineering package which enabled us to effectively use two-layer boards.

The choice of a circuit family drastically affected the type and number of interconnections required. The Complementary Transistor Micro Logic (CTμL) family was chosen because of its useful "wired OR" capability which has proven to reduce interconnections between 20 and 40 percent compared to other circuit families.[6]

The system bus implementation was also given much consideration. After preliminary studies it was decided to use a set of 200 Interconnections running as parallel lines for the length of the main frame. Figure 4 is a view of one of the system bases which is a simple two-sided printed circuit board. One hundred and eleven lines run the full length of the system and are used as the main bus. The main base is partitioned as follows:

64	bidirectional data lines
24	bidirectional address lines
10	bidirectional priority lines
6	operation code lines
5	terminal number lines
1	system clock
1	system clear

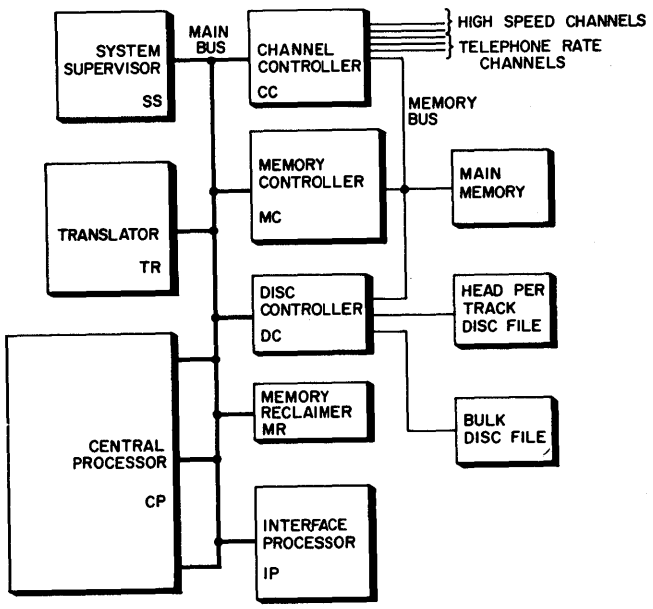


Figure 1—Gross block diagram of the SYMBOL system

The balance of the 200 lines are used for local interconnections within AFUs of two to seventeen boards in size. As the design progressed it was necessary to add an additional 200 "bypass" lines on the bases. The final design allows each large board to contact up to 200 bus lines and have 200 lines bypass it. Each board contact can be connected to the same contact number on the next card or alternatively can be connected via a bypass line to a board several slots distant. Using these techniques the whole system was implemented with a maximum of 600 parallel lines, with cuts, on two-layer printed circuit bases.

There have been many interesting debates within the project on the size of boards chosen and on the number of interconnecting lines needed on the bases. It is not clear that our choices are optimum; however, it is now clear that they were sufficient. The completed system used about 102 large boards inserted between the two system base structures. It is illustrated in Figure 5. It was interesting to find that all the main

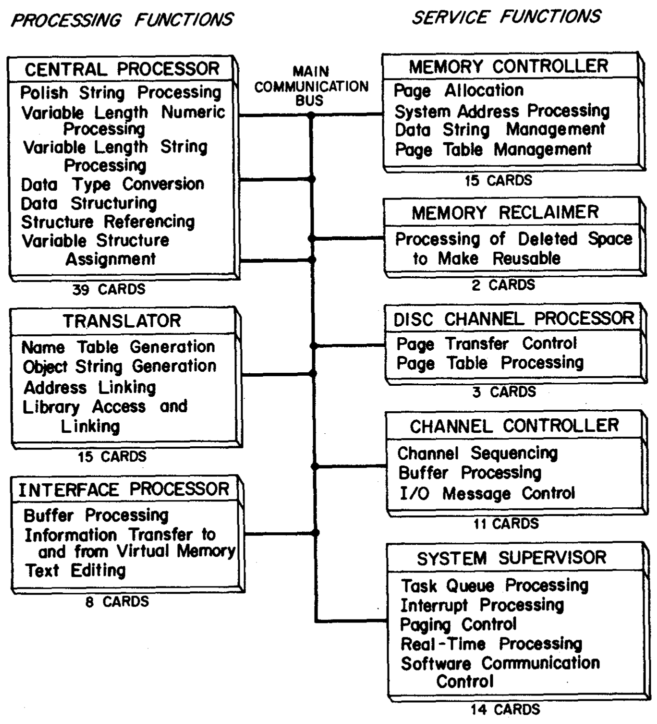


Figure 2—Functions performed in the SYMBOL main frame

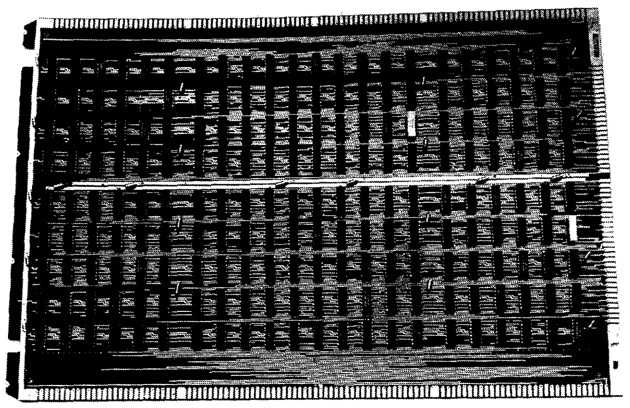


Figure 3—Basic 12"×17" two sided printed circuit board with up to 220 dual in-line components

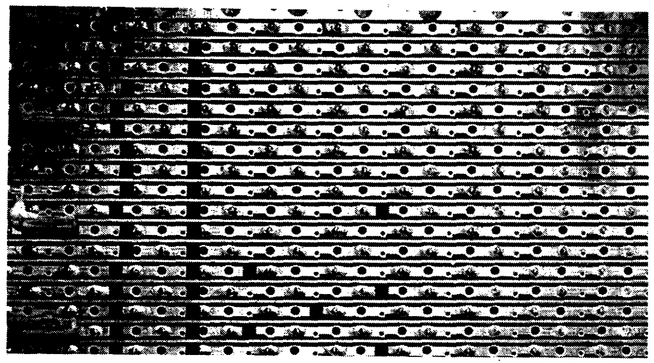


Figure 4—SYMBOL interconnection base

frame including I/O, memory, disk and channel interface logic took less than 20,000 CTμL packages.

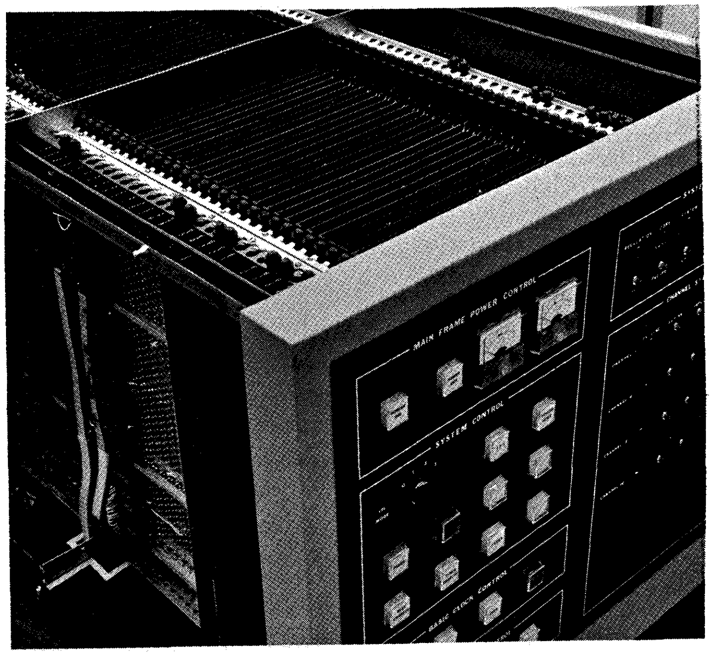


Figure 5b—Detail view of SYMBOL main frame

SYMBOL FROM A PERFORMANCE VIEWPOINT

The evaluation phase of SYMBOL IIR is just beginning with the hardware near completion. In order to obtain a preview of the performance a set of measurements has been made on the hardware.

Basic operation rates

The clock period on SYMBOL IIR now stands at 320 nsec and may be later reduced to about 200 nsec. All measurements were taken at the 320 nsec period. The basic clock period in SYMBOL IIR contains long logic chains allowing relatively complex tasks to be performed. Many of the key logic chains contain 20 to 25 levels of AND-OR logic. The system uses Fairchild CTμL, type I throughout. The core memory is a 1964 model with a basic 2.5 μsec cycle. Due to a semi-serial interface on the core memory it has an effective cycle of 4 μsec,

An improved system (referred to as SYMBOL II) has been studied and has been partially specified. This system is based on the technology of the experimental system, SYMBOL IIR, but has been considerably optimized. SYMBOL II is also specified to use the latest cost orientated hardware technology. Conservative performance estimates of SYMBOL II will be made to give a comparison of how the SYMBOL algorithms would stand up in a contemporary hardware technology design. They will be based on a clock period of 100 nsec using a circuit family such as CTμL, type II and an LSI memory with a 200 nsec period. One should keep in mind that the following comparisons are between SYMBOL, which is a VFL machine running in a very dynamic execution time environment, and a more conventional fixed field machine running a language with the data boundaries determined at translate time. The former places more demands on the hardware while the latter shifts the burden of data management to the user.

Figure 5a—The SYMBOL main frame

For the purposes of comparison SYMBOL IIR will be referred to as SIIR and SYMBOL II as SII,

Field processing operations

SIIR performs all field operations in a VFL serial-by-character mode. It was always assumed that after system evaluation and bottle-neck analysis, if warranted, certain operators such as those shown below would be executed in a more parallel mode by using additional hardware. SII estimates are based on serial processing and known algorithm improvements that reduce or do not materially increase the hardware required.

The following table gives processing times measured on SIIR and estimated for SII. The execution time values are specified in microseconds and do not include the instruction fetch time or single word operand fetching and storing.

SYMBOL IIR MEASURED EXECUTION TIMES IN μSEC

OPERATION	SIIR	SII
1234+4321	5.6	1.2
12345678-87654321	10.0	1.6
50 digits + 50 digits	45.0	5.0
Convert to floating point 1234	5.2	1.2
Convert to floating point 12345678	12.5	1.8
Convert to floating point 50 digits	120.0	18.0
Compare 12345678,87654321	4.0	1.0
Compare 12345678,12345670	6.5	1.2
|abc|join|def|	4.5	1.2
|12345678|join|12345678|	60.0	12.0
1234 format |ZZZ.DD|	9.0	3.0
1234 format |ZBZBV|	8.0	2.6
12345.6789 format|'$'*C***C***.DD|	76.0	15.0

Compilation

Several programs were compiled on SIIR and the overall times and space usage measured. The SIIR results are tabulated below.

SYMBOL IIR MEASURED COMPILE TIMES IN μSEC

	NO. STATEMENTS	BYTES OF SOURCE	BYTES OF OBJECT CODE	AVERAGE TIME PER STATEMENT
Program A	195	8330	7315	820
Program B	70	3528	5112	1280
Program C	157	7560	6025	760

This represents about 75,000 statements compiled per minute on SIIR.

A comparative table for SII assuming added flexibility on SII for handling various other languages in addition to the SYMBOL language is given below. The data is based on a sampled study of object code and projected execution times of several recently developed algorithms.

SYMBOL ESTIMATED COMPILE TIMES IN μSEC

	STATEMENTS	BYTES OF SOURCE CODE	BYTES OF OBJECT CODE	AVERAGE TIME PER STATEMENT
Program A	195	8330	2350	185
Program B	70	3528	1735	220
Program C	157	7560	2110	185

This would give a compilation rate of 300,000 statements per minute.

Paging overhead

SYMBOL has very low overhead for paging. The algorithms are based on direct hardware execution using parameters set up by software. A count of worst case paging overhead for SIIR in terms of memory cycles for a CP page out is given below.

SYMBOL IIR PAGING OVERHEAD IN MEMORY CYCLES

ITEM	WORST CASE	AVERAGE
CP Shut Down	7	7
SS Queuing and Push Selection	50	30
SS Disc Servicing	8	6
CP Start Up	6	6
TOTAL Memory Cycles	71	49

Assuming an average of 5 μsec per memory cycle counting internal cycles this gives 355 μsec worst case. In SII using an improved algorithm the overhead would be less than 20 μsec.

Input/output

The overhead for I/O for a time-sharing system becomes an important factor in providing adequate terminal response time. To illustrate the effect of the hidden software overhead an operation trace of a IBM 360/44 during FORTRAN IV output was performed. A similar operation was performed on SIIR. The equivalent output statements in both languages are shown in the table below.

The trace of the FORTRAN statement indicated 1753 instructions being executed. Each instruction requires an average of two memory cycles. The trace program does not trace any of the supervisor or channel operations so that well over 3,000 and more likely near 4,500 memory cycles were used in executing the FORTRAN statement.

SYMBOL VS FORTRAN OUTPUT STATEMENT
TRACES IN MEMORY CYCLES

LANGUAGE	STATEMENT	TRACED	EST. OVERHEAD NOT TRACED
SYMBOL	OUTPUT 12345.56 FORMAT |$D.DDD_{10}DD$|;	130	0
FORTRAN	WRITE (6,10)X		
	10 FORMAT (1X,E9.3)	3466	1000

Task control overhead

In order to measure the overhead for compilation and execution a program consisting of one CONTINUE statement was executed on SIIR. This causes a null program to be entered, translated, and executed and thus places a large demand on any system resources required, isolating overhead from "useful" actions. All memory cycles were traced with the following distribution:

PROCESSOR USED	MEMORY CYCLES
SS	41
TR	20
CP	18
TOTAL	79

This could be compared with any contemporary system where the entire compiler would have been paged in and much of the supervisor would have been executed to establish many resources that would not have been needed.

Subscripting

It would seem that VFL data structures imply slow data referencing. However, the SYMBOL project demonstrated that efficient handling of dynamically varying data can be achieved with sophisticated list processing techniques. SYMBOL IIR established the foundation and the algorithms have now been developed to be competitive with conventional fixed field indexing while retaining the VFL features. A few references and their equivalent memory cycles for SIIR are given below. The subscript Fetch cycles are not counted.

REFERENCE	TYPICAL MEMORY CYCLES REQUIRED
A[4,9]	4-6
A[16,32,6]	8-10
A[3]	2-3
A[70]	9-12
A	2

A substantial improvement has been obtained for SYMBOL II promising to make it as fast or in some cases faster than conventional indexing.

SYMBOL FROM A COST VIEWPOINT

A study of a modern computer installation and its users as a total "system" reveals where and how the computing dollar is divided. Consultants fnom Iowa State University made available all the necessary data for such a study early in the program.[7] Figure 6 illustrates the I.S.U. IBM 360/50 installation in 1966 at

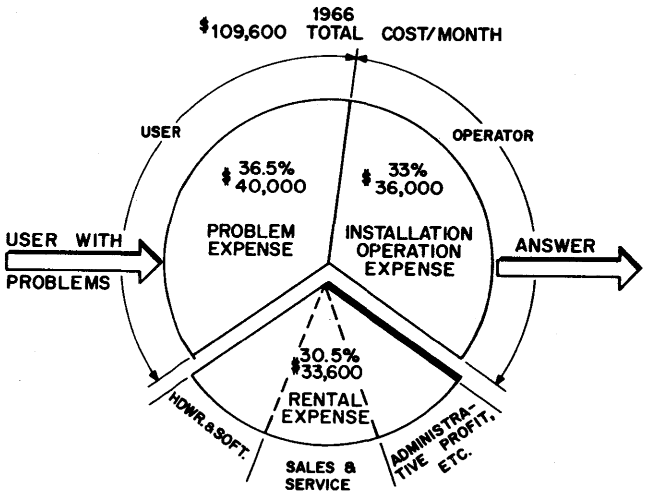


Figure 6—The computing pie illustrated for Iowa State University 360/50 installation

the time the study was made. This "pie" has since been compared with many other business and scientific installations of varying sizes with different computer systems. There is general agreement that the minor variations in the size of the slices for different installations do not materially affect the picture. This applies to most modern "classic software-dominated systems."

The objective of data processing is to solve problems where the "user with a PROBLEM" is the input and the "ANSWER" is the output. It is assumed that the user has his problem well defined and has the data available but the data is not yet programmed. The conversion of his problem to a computable language and the debugging necessary for correct execution is included in the total cost of operating an installation.

I.S.U. calculated the total system operation on this basis as approximately $109,600 per month. The rate and labor costs were adjusted to normal commercial standards for the calculations. Both commercial and scientific problems were run in the problem mix. The following sections discuss the breakdown of the overall cost.

About 37 percent or $40,000 is used by the problem originator and/or the professional programmer to convert the problem to a debugged, high-level language and to obtain answers.

Thirty three percent or $36,000 is required for operating personnel, key-punch operators, file clerks, systems programmers, administration, space, power, etc.

Thirty percent of the total pie or $33,000 goes for

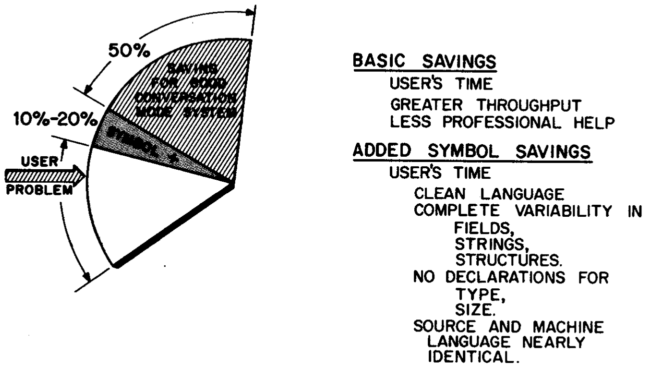


Figure 7—Savings in problem expense

machine rental. It is estimated that about one third of the rental expense goes for direct development of hardware and system software (perhaps half and half), one third for sales, service, and application support, and one third for administrative costs, overhead, and profit.

The choice of a hardware configuration and its machine language is the tail wagging the dog. Inexpensive hardware and a good, easy-to-use programming system can reduce the size (i.e., total cost) of the pie but in conventional systems will not materially alter the relative size of the slices.

In the following text the computing pie is used to illustrate SYMBOL concepts from a cost point of view. Each major slice will be further subdivided into its own percentage parts (i.e., each major slice will be 100 percent of the portion under consideration and will be divided into its constituent parts).

Figure 7 shows the potential problem expense saving to be obtained from any good conversation-mode, high-level language, time-sharing system. It has been estimated that approximately 50 percent of the problem expense slice can be saved in reduced user learning time, increased throughput, less professional programming support required, etc. We estimate the SYMBOL system will further reduce these costs with its "clean" and "concise" directly implemented high-level language and simplified operating system.[1]

The savings in the operation of an installation comes from four sources. This is illustrated in Figure 8. First: A good time-sharing system will reduce the administrative help such as file clerks, keypunch operators, etc. It is estimated that this saving can be ten to fifteen percent of the installation operating expense exclusive of system rental. The SYMBOL system with conversation-mode multiprocessing and multiprogramming will also share in this saving. Second: The "system software" support required in a conventional installation is a very significant portion of the expense. Here SYMBOL shows a definite added saving. What system software remains can be written in the high-level, general-purpose language and will be easier to write, debug and understand later. This will reduce the number of professional personnel required. Third: The SYMBOL language is directly implemented in hardware and thus uses less main memory for "system software." For example, a resident compiler is not required. In addition, much less program swapping occurs and thus less virtual memory transfer time is needed. Hardware execution of algorithms is also faster and results in enhanced instruction execution speed. These features will require less programming attention and also provide more throughput per installation dollar spent. Fourth: The SYMBOL hardware is designed with modern integrated circuits and large two-layer printed circuit boards. The total system hardware package is compact and does not

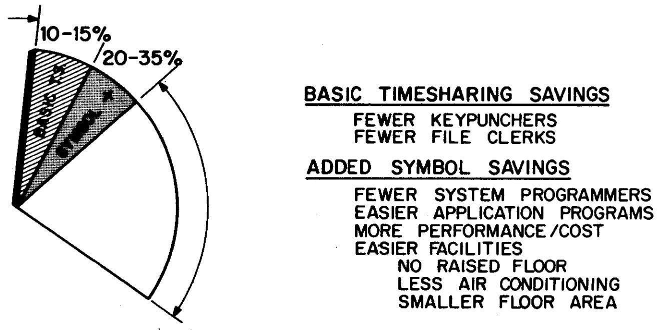


Figure 8—Savings in installation operation expense

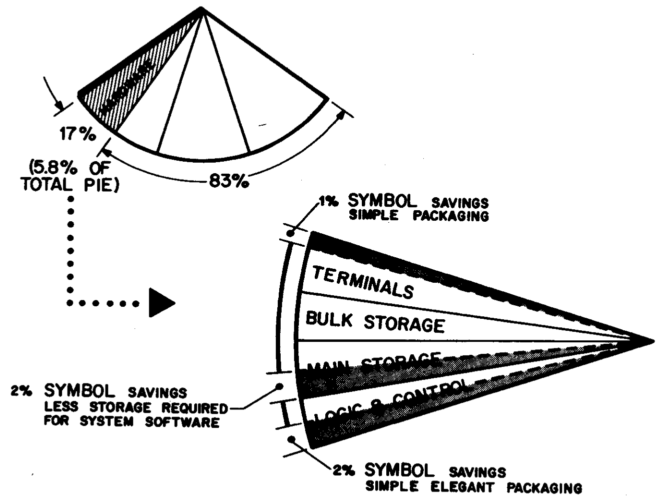


Figure 9—Manufacturer's direct hardware expense

need raised floors, special air conditioning, or vast amounts of floor space. It is estimated that these SYMBOL features will reduce installation operating expense by an additional 20-35 percent or a total of 30-50 percent.

The slice of the computing pie representing the computer manufacturer's hardware contribution is illustrated in Figure 9; approximately seventeen percent of this slice is attributable to hardware. For large systems the peripheral equipment and the bulk files can approximate about one half of the total cost. The main storage is another quarter and the CPU logic is another quarter. Naturally some variation in these amounts will occur from installation to installation and for different system types.

The SYMBOL approach saves costs in several ways: The first area of savings is in the use of large two-layer printed circuit boards and two-layer printed circuit bases with cam-operated contacts for *all* system interconnections.

Except for cables to peripherals and wires used for correction of design errors and for logical extensions no wire exists in the system. Figure 3 illustrates a logic board and Figure 5 illustrates the main frame of the SYMBOL system. This type of packaging lowers production costs for logic. It is estimated as much as a fifty percent saving will be achieved over small board, wire-wrap back panel, multi-cabinet conventional systems. This same technique reduces costs in terminal equipment but not to such a large degree. We estimate that three percent of the manufacturer's slice of the pie can be saved by this functionally-factored, bus-oriented, large printed circuit board design philosophy. The second way savings are obtained is in the hardware efficiency gained by the SYMBOL system. Since most of the normal system software is hard wired, very little resident main memory is used, thus providing much larger percentages of main memory for application programs. The execution of system instructions is done at "clock speeds" in a "macro" rather than a "micro" manner. This provides much faster high-powered instruction execution. Finally, more of the system hardware is simultaneously operating due to the system organization which allows multiple jobs to be in the main frame for overlapped execution. We estimate that an additional two percent of the manufacturer's slice of the pie is saved here.

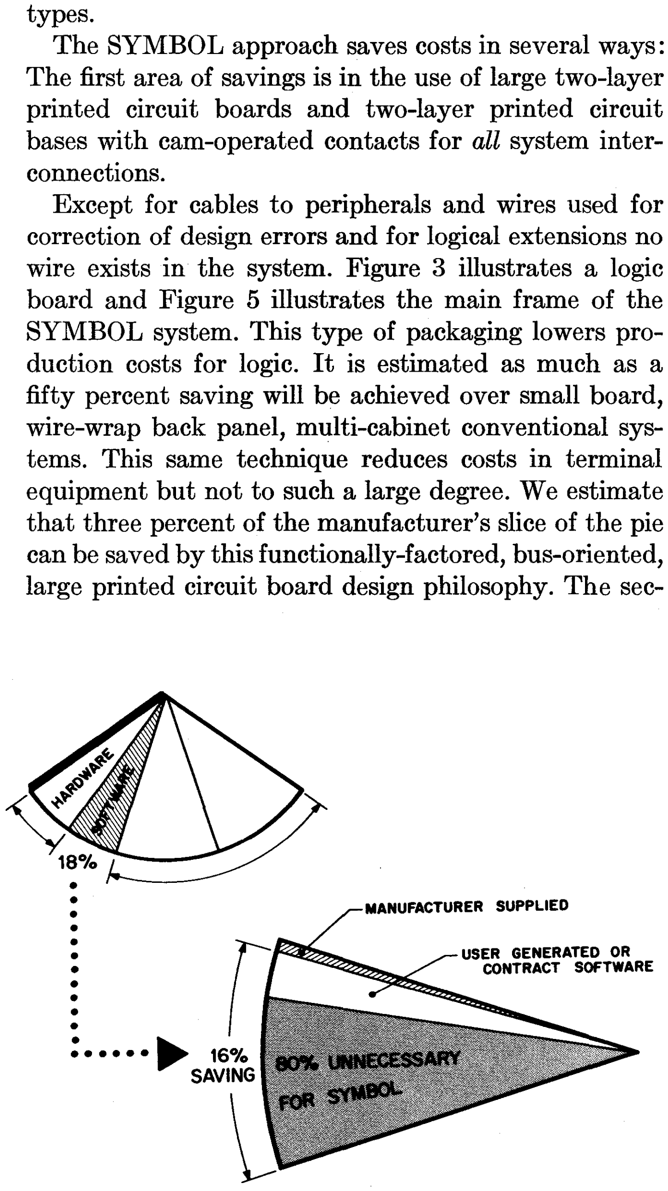

HARDWARE
SOFTWARE
18%
MANUFACTURER SUPPLIED
USER GENERATED OR CONTRACT SOFTWARE
16% SAVING
80% UNNECESSARY FOR SYMBOL

Figure 10—Manufacturer's system software expense

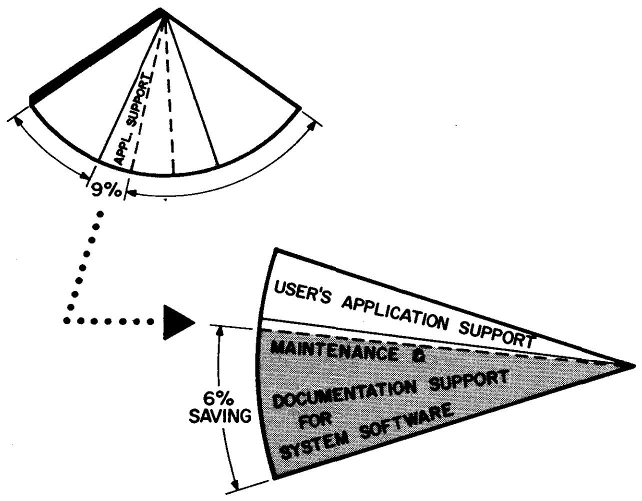

APPL. SUPPORT
9%
USER'S APPLICATION SUPPORT
MAINTENANCE & DOCUMENTATION SUPPORT FOR SYSTEM SOFTWARE
6% SAVING

Figure 11—Manufacturer's software application expense

The largest and most important single saving for SYMBOL is in the "System Software." Figure 10 illustrates this point. Irrespective of whether the system manufacturer or someone else produces the software for a conventional computer this large expense is real. The SYMBOL features directly implemented in logic (i.e., hard wired) make unnecessary at least 80 percent of the conventional system software used in large timesharing machines. This represents an estimated 16 percent saving in the system manufacturer's slice of the computing pie.

The field support of the system software is a major expense. The sheer volume of paper and record keeping to keep current with the latest changes is a major problem. In the design of the SYMBOL system this problem was given great attention. In studying the software delivered with large systems using a relatively static high-level language, we note that most (if not all) of the changes made were on the programmed implementation or were due to programming errors. Many levels of machine and assembly language programs and machine runs were between the hardware language and the programmers' source language. This quite naturally introduces confusion (and errors) either in original programming or in understanding the hidden rules when using the system.

It may also be noted that as more and more applications are programmed in a language it automatically becomes more rigid. We believe that the "clean," high-level, general-purpose SYMBOL language is excellent for most uses. Since direct hardware implementation requires little field support in the software sense, we estimate approximately a six percent saving in the manufacturer's support expense. This is illustrated in Figure 11.

Good service is a must in a large system. The SYMBOL hardware has been engineered for good reliability and at the same time easy maintenance. We do not anticipate any added expense for SYMBOL hardware maintenance over conventional systems with equivalent storage and logic circuit counts. Our experience on the SYMBOL model has verified this belief.

The previous material has split the computing dollar up in parts and has described how major savings can be realized with a "total systems" approach. The SYMBOL techniques described herein together with good time-sharing, conversation mode practice can reduce computing costs up to 50 percent. Referring to Figure 12, one may visualize how the savings in the whole computing pie add up.

CONCLUSION

The full scale running SYMBOL system has demonstrated the following:

1. A very high-level, general-purpose, procedural, "state of the art" language and a large portion of a time-sharing operating system can be effectively implemented directly in hardware.
2. Complete dynamic variability in data fields, data structures, processing length of strings and numbers and depth of structural nesting (subscripting) can be efficiently and directly implemented in hardware. Competitive execution speeds can be obtained as compared with more conventional fixed field and structure machines and in certain areas such as language translation extremely high performance rates can be demonstrated.
3. Design and construction techniques using only large two-layer printed circuit boards for all system interconnections and buses together with a functionally factored system results in an economical, serviceable and reliable system.
4. The direct hardware implementation of a general-purpose, high-level language, the use of the SYMBOL construction techniques and a good conversation mode system can save up to 50 percent of computing costs. This is contrasted to a good conventional system using a general-purpose, high-level batch oriented system.
5. The use of the SYMBOL system (software plus hardware) has shown that significantly fewer hidden rules exist to plague the casual or the professional user in debugging programs.

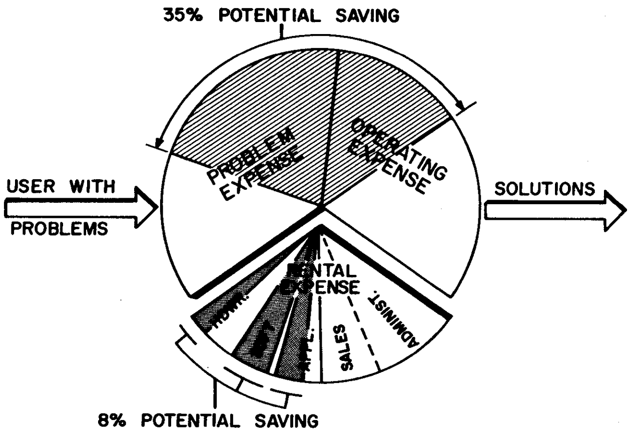


Figure 12—Potential savings with a good conversation mode hardware/software system

ACKNOWLEDGMENT

The authors wish to express their thanks to the Fairchild Camera and Instrument Corporation for the opportunity to do this research.

Special thanks are due to Professor Robert Stewart, Professor Arthur Pohm and Professor Roy Zingg of Iowa State University for their constructive help and support. We are most grateful to the National Science Foundation for the grant which made it possible for the SYMBOL IIR system to be delivered to Iowa State University for evaluation and continuing research on this technology.

Finally, we are deeply indebted to the past and present members of the Digital Systems Research Department at Fairchild. Their hard work and enthusiasm made this research project result in a real and operating system.

REFERENCES

1 G D CHESLEY W R SMITH
The hardware-implemented high-level machine language for SYMBOL
This volume

2 J R TENNANT G D CHESLEY
Design and layout of large integrated circuit boards
Second Annual Seminar on Integrated Circuits January 1965

3 A P MULLERY R F SCHAUER R RICE
ADAM-A problem-oriented SYMBOL processor
Proc SJCC Vol 23 1963 pp 367-380

4 W R SMITH et al
SYMBOL: A large experimental system exploring major hardware replacement of software
This volume

5 B E COWART R RICE S F LUNDSTROM
The physical attributes and testing aspects of the SYMBOL system
This volume

6 W R SMITH
System design based on LSI constraints: A case history
Digest of 1968 Computer Group Conference June 25-27 1968 International Hotel Los Angeles California IEEE 345 East 47th Street New York New York

7 R RICE
Impact of arrays on digital systems
IEEE Journ of Solid-State Circuits Vol SC-2 No 4 December 1967

An expanded set of references to cover work on the SYMBOL can be found in References 1, 4, and 5.

The physical attributes and testing aspects of the symbol system

by B. E. COWART, R. RICE and S. F. LUNDSTROM

Fairchild Camera and Instrument Corporation
Palo Alto, California

INTRODUCTION

The major goals of the SYMBOL computer research project have been to provide a more effective man-machine interface and to reduce the total cost of a digital system to the user. The development of the multi-processing/multi-programming computer architecture with much of the executive system, memory management, and high-level language implemented in hardware is described in other related papers.[1,2,3] The SYMBOL project also has investigated low-cost construction techniques suitable for equipment to be used in commercial/industrial environments.

The packaging system utilized (See Figure 1) is a free-standing, self-contained unit with the electronics fabricated entirely on two-sided printed circuit (p.c.) boards. In addition to low-cost mechanical packaging, special low-cost test equipment has been developed. It utilizes the same mechanical packaging and allows very rapid and efficient testing of individual logic boards and of the full SYMBOL system. These techniques result in a significant savings in the amount of engineering manpower required for hardware prototype debugging.

PHYSICAL CHARACTERISTICS

One of the first problems attacked by the SYMBOL research project was the excessive cost and interconnection complexity of the mechanical packaging techniques commonly used for digital systems. The mechanical packaging system developed allows the use of free-standing, self-contained units requiring no special or raised floors and no special cooling, and using commercially-available power supplies. The SYMBOL computer shown in Figure 1 utilizes this packaging approach. The section above the table top contains all the logic and system interconnections for the CPU, I/O channels, memory controls and system executive controls.[2] The lower section contains all of the power supplies, power controls and the cooling components. In Figure 1B the section above the table top and in the foreground is the SYMBOL logic. The smaller enclosed section in the rear is a hand wired system maintenance unit.

The logic (upper section) is fabricated on large (12″ × 17″) two-sided printed circuit boards with plated-through holes. (See Figure 2). These logic cards contain all of the active logic components in the SYMBOL system and are easily removed and repaired. As shown diagramatically in Figure 3, the logic boards are mounted on one-half inch centers between a pair of two-sided printed circuit "motherboards."

Interconnections between logic cards are made on the motherboards with parallel lines for bus connections. The local buses which intraconnect Autonomous Functional Units (AFU's)[2] are physically identical to, and intermingled with, system buses which interconnect the AFU's (global signals). These printed circuit buses provide all required system interconnections. Every signal pin position has two associated buses. One bus connects to the pin while the second bus bypasses the pin. A signal on the connect bus and a signal on the bypass bus may "cross-over" between board positions (Figure 3). In addition, either or both buses may have continuity breaks to isolate groups of boards (Figure 5).

Due to practical limitations on motherboard size, the SYMBOL system has been implemented using five motherboard modules. Each motherboard module can interconnect thirty logic boards. In the SYMBOL system four modules are used for the computer logic and one module is partially used for the system maintenance unit. As mentioned above, all signals in the system are distributed on the printed circuit wires of the two-sided logic boards and motherboards. The only exceptions are cables to external peripheral devices and bulk

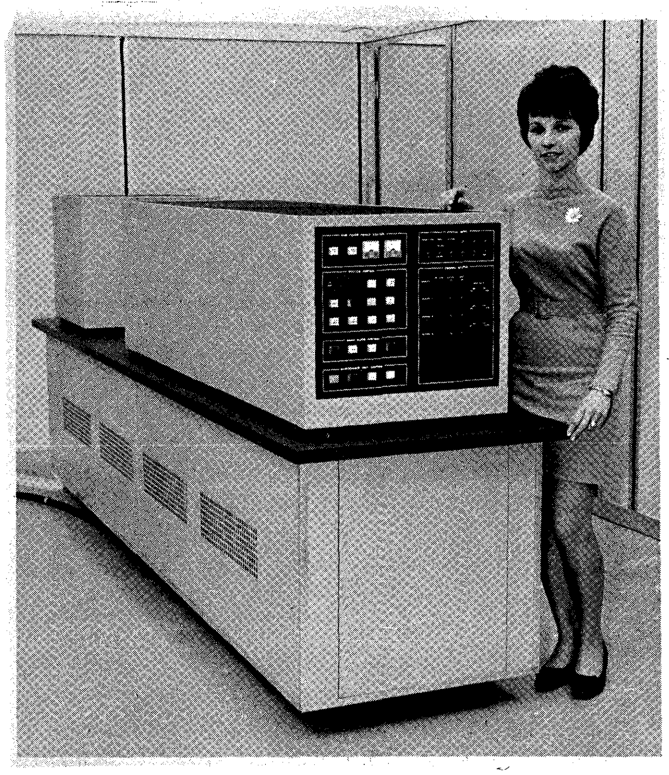


Figure 1a—SYMBOL main frame including power, cooling, logic and interconnects

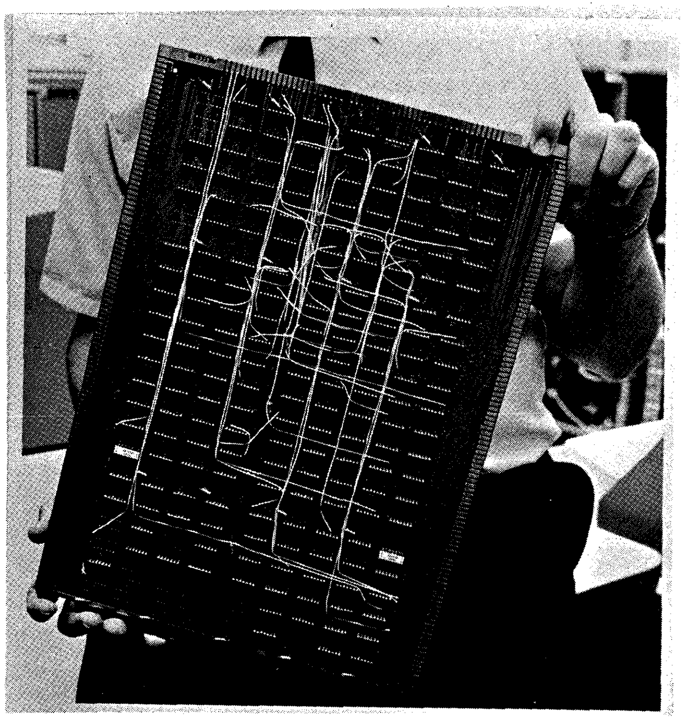


Figure 2—Typical 12″×17″ two-sided logic board with additions or changes in discrete wires

storage (Figure 4) wires which fix errors or implement design improvements (Figure 2), and the short jumper wires used to connect the motherboard modules (Figure 5).

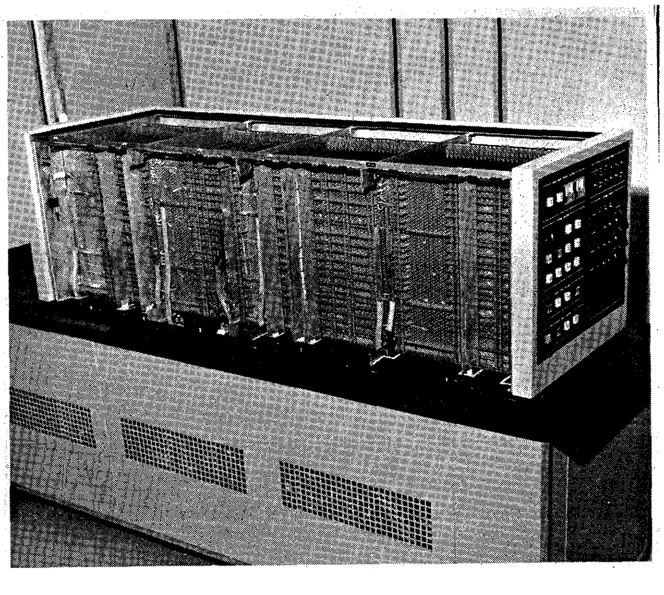


Figure 1b—SYMBOL main frame with covers removed

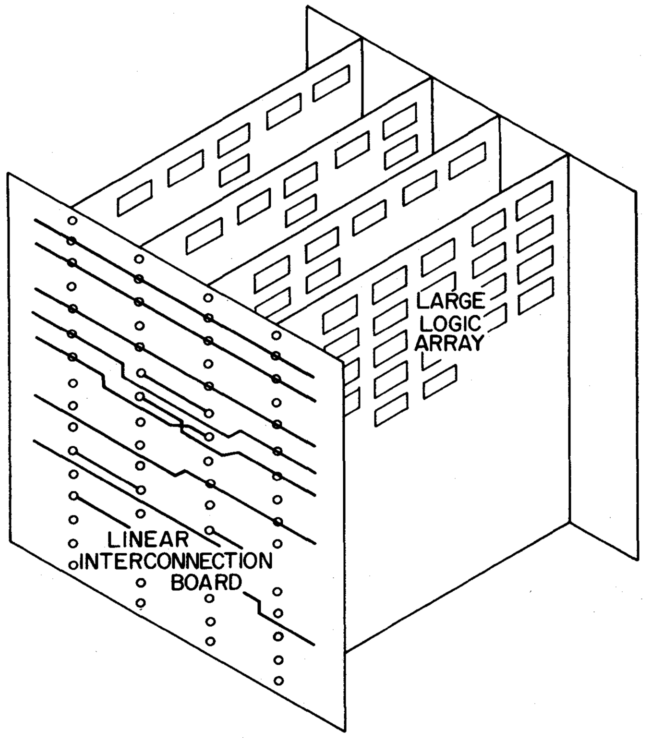


Figure 3—Conceptual physical organization of SYMBOL

As Figure 3 shows, connectors are required on two edges of the logic boards to provide signal paths to the motherboards. A cam-operated contact system was developed to allow zero contact-force insertion and removal of logic boards (from the top) and to provide high reliability, high pressure contact when engaged. The contacts themselves are soldered into the motherboards and, together with some molded plastic card guides, become a permanent physical part of the motherboard interconnection system (see Figure 6). To cam or uncam a board involves rotating two cam pairs using a cam key. No special board handles, locks, etc., are required.

In addition to logic board contacts on the inside of a motherboard, contacts may be provided on the outside of the motherboard at each board position. The connector positions on the outside of the motherboard allow lamp panel maintenance aids called wing panels (Figure 7) or printed circuit board cable connectors called paddleboards (Figure 4) to be connected to any board position in the system. In practice, only those positions specified before fabrication are used for external connection.

The signals interconnected by the motherboard system originate on the logic boards. Figure 2 shows a typical logic board. The 200 signal pins, a clock pin and a number of power pins on the two long sides connect to the motherboards. An additional fifty pins used for test points are available on the top of the board for monitoring during system operation and test. A typical logic board holds about 175 dual in-line packages; the maximum capacity is 220 packages. High packaging density has not been a major system goal. On the contrary, emphasis is placed on functional completeness and interconnection density on a board. An additional effort was made during prototype fabrication to guarantee some extra package positions on each board (at least ten percent) to allow implementation of logic functions "overlooked" and design improvements. Discrete wires are used to repair board fabrication errors and to implement additional logic. No degradation in performance has resulted and no special electrical considerations have been required. Surprisingly, not one board has had to be remade for the system. Discrete wire and component additions to the original boards have sufficed to bring the system to an operational state.

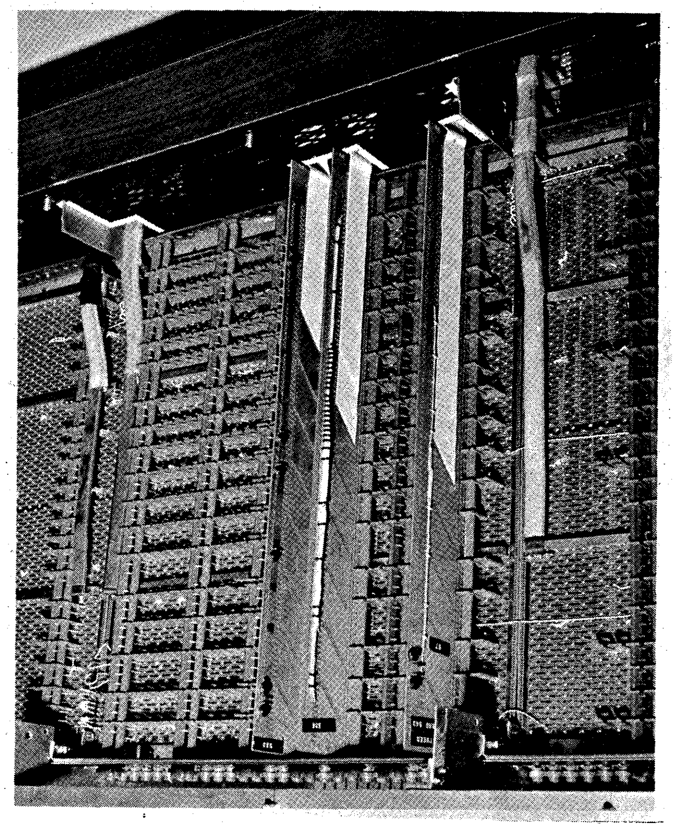


Figure 4—Paddleboards with cables to external peripheral equipment and bulk storage devices

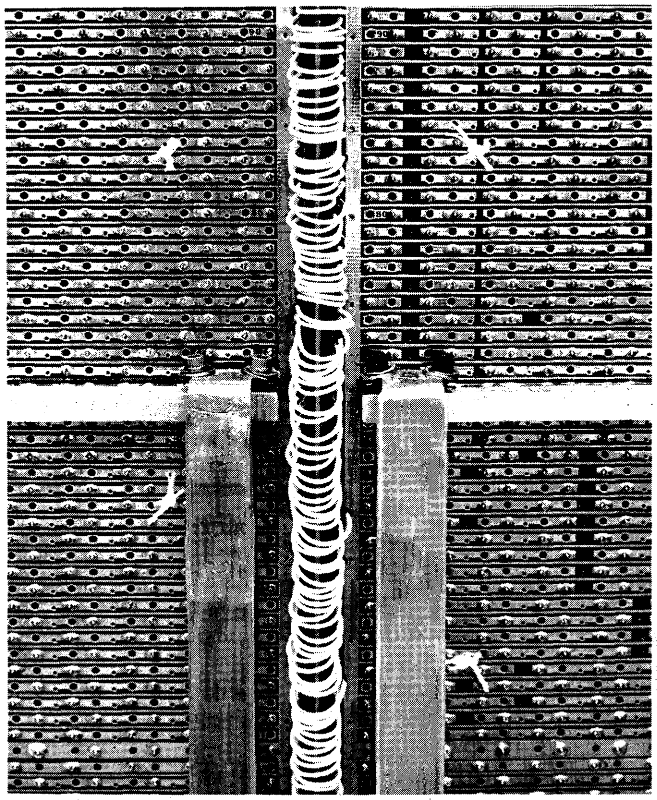


Figure 5—Motherboard module interconnect wires and main power distribution system

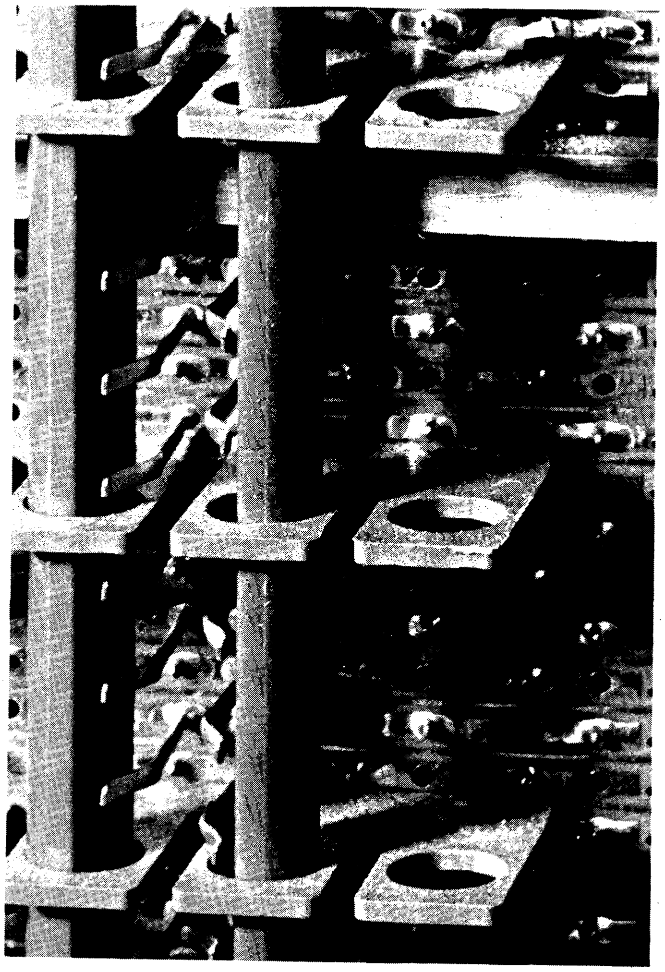


Figure 6—Zero insertion force, cam-operated connector system

ELECTRICAL CHARACTERISTICS

A mechanical packaging system is useless unless the digital system itself can operate successfully in that package. The electrical characteristics of the SYMBOL system were chosen to provide a relatively high-speed system using conventional technology. Design decisions were made with speed enhancement and noise suppression high on the list of design criteria. Few concessions which compromise economy have been permitted.

Circuit family

The Complementary Transistor Micro-Logic (CTμL) circuit family chosen facilitated both the ease of design and the operating speed of the system. The CTμL family[4,5] is a positive logic family with wired-or capability at every point except at the flip-flop outputs. The basic AND gate is a emitter-follower device with relatively slow rise and fall times (7-10 nsec for the 3V swing) but with a relatively fast propagation delay through the gate (typically less than 5 nsec). The slow rise and fall times significantly reduce the problems of fabricating a high-speed system on two-sided printed circuit boards. However, since the basic gate is non-saturating, there is a slight voltage degradation through each gate. Thus, level restoring elements (AND, NOR, LATCH, FLIP-FLOP) are required at various places in a chain of logic as shown in Figure 8. In SYMBOL (a synchronous system) the total chain allowed in one clock cycle is four sets of the gate and level restorer combinations. The experience gained in SYMBOL shows the importance of the wired-or capability very clearly. There are a significant number of five, six, or seven-way wired-or ties. If discrete OR gates had been required, the size of the system would have been significantly larger. Most AND functions are four inputs or or less with the average about two-inputs. There are some 10-way and larger OR-ties.

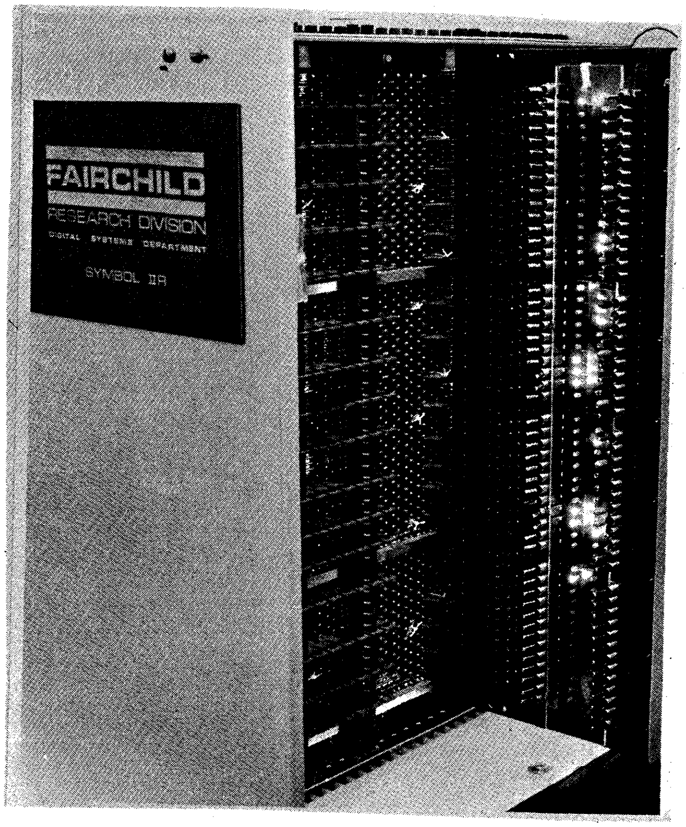


Figure 7—Wing panel in motherboard position for monitoring signal buses

CAD support

A computer-aided design (CAD) system was developed to assist appropriate design and fabrication operations in the development of SYMBOL. The approach was not one of developing a general system and then using it for the SYMBOL design. Instead, when a need developed where CAD programs could help, a new section of the system was generated specifically for that need. Usually the new section of the CAD system was generated by the engineer or designer first requiring that section. The resulting system greatly enhanced the power of the designers and the reliability of the final system.

One of the sections of the CAD system is a logic design assistance sect.on. In this section, each device is characterized as it can be expected to perform in the system environment. This system characterization is in terms of quantized, normalized units of time delay, signal voltage degradation, threshold variations, and fan-in/fan-out effects. Time delay characterization includes an allowance for typical propagation delays as well as device delays. Special noise conditions (such as global signals or cables to peripherals) are accounted for by modifying the characterization of the devices used in these services. Special time or loading conditions can be forced, especially in the case of long bus signals. The CAD system points out violation of noise margins, load, or time delays so that the designer can modify a design to comply with the system rules.

In the section of the CAD system which performs the logic placement and wire routing functions, no attempt was made to specifically minimize wire lengths or to provide the maximum number of devices per board. Instead, placement and routing were based on interconnect capability of the two-sided printed circuit board. The rules for partitioning of logic to various boards by the designer were also based on this same criteria, modified by an added criteria that the printed circuit cards should consist of complete functions.

Signal distribution

Many signal distribution problems are encountered when using high speed circuits.[4,6,7] The choice of the CTμL logic family considerably reduces the problems of signal distribution in systems with two-sided printed circuit boards. Signals between boards within an AFU are treated no differently than on-board intraconnects. These signals are the AFU intraconnections on the motherboards and are less than eight inches in length. No attempt is made to provide terminations or impedance controls.

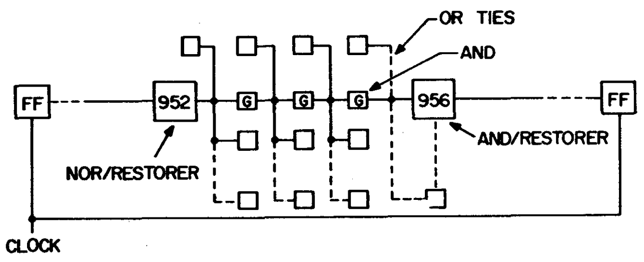


Figure 8—Representative *CTμL* logic chain in SYMBOL

Global signals, on the other hand, can be as much as six feet long. The base interconnects for these signals are treated as transmission lines with distributed discrete loads, and an approximate termination is placed at each end of the lines. The clock line is a particular example of this type of signal (Figure 9). The copper trace for this signal is as wide as practical to reduce its impedance. The wide trace reduces the effect of the loads on the propagation velocity and minimizes reflections and distortion on the line. Each of the boards (107 total) in SYMBOL presents a load on the clock line of approximately 500 ohms to ground. To drive this low impedance load a special clock driver is required, one of the few discrete circuits in the system. The other main bus signals (except for system clear) did not require discrete circuits because each AFU was constrained to connect to the system bus only once.

POWER

The SYMBOL electro/mechanical system is designed to deliver properly regulated power at each circuit in the system and to remove the generated heat. Cooling of the system is accomplished with miniature axial fans blowing air upward through the card cage. No special room air conditioning is required. Power distribution is somewhat more complicated. Although the power distribution system was designed to provide excellent DC regulation, ease of installation and economy took precedence over requirements for dynamic performance. However, power distribution had to be appropriate for the high-speed circuits in the system.[8] Power is distributed throughout the system using a combination of large copper bus bars on the motherboards and a power distribution grid of printed-circuit wires on the logic boards.

The current in a thirty card module is approximately 200-250 amperes. To distribute this much current over the fifteen inch span of the module, a large copper bus is used. This bus in turn is connected to the main distribution bus by laminated copper risers (see Figure 5).

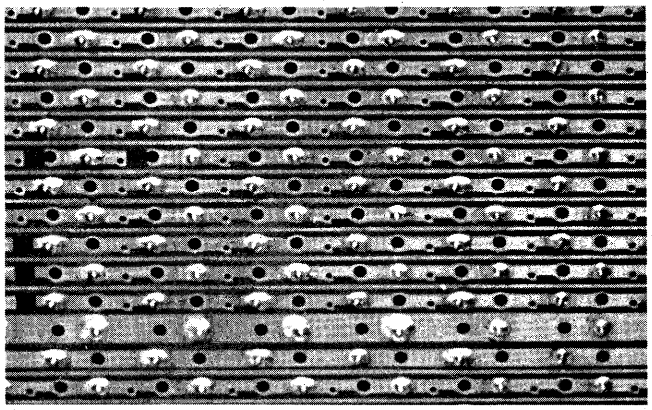


Figure 9—Close-up of motherboard showing wide clock line with wide and narrow ground lines for shielding

The system power is distributed longitudinally near the lower edge of the motherboards. The voltage regulators use remote sensing at the main distribution buses. The riser, distribution bus, and printed circuit board distribution system are sufficiently massive to provide regulation of less than fifty millivolts from the sense point to the remote sectors of the printed circuit board.

The power distribution system on the logic boards is standardized to provide a pseudo-ground plane made up of the composite of the positive, negative, and ground distribution grids (Figure 10). The grid geometry was chosen so that the circumference of a net cell is less than a half wave length for the highest frequency of appreciable magnitude. The size of the logic board is such that local filtering on the boards is more efficient and productive than attempts to provide super-low distribution impedance. The decoupling capacitors can be seen on the logic board in Figure 2.

The CTμL circuit family chosen for use in SYMBOL requires two supply voltages in addition to ground. The three-terminal DC equivalent circuit can be represented as in Figure 11. As can be seen, the ground terminal can be viewed as a current source to the power system, requiring that approximately thirty percent of the positive terminal current be diverted by an appropriate sink. In the SYMBOL system, a shunt regulator stabilizes the potential between the negative terminal and the ground terminal. This technique is used instead of the normal two supply (+V, −V) system because only 4I worth of supplies and related inefficiencies are required (3I for +V and I for ground regulation) instead of 5I (3I for +V and 2I for −V).

The power supply system used is a floating supply which uses a combination of unregulated and regulated supplies to generate 800 amps at 6.7 V. Since the power load of CTμL does not change appreciably with the state of the device, the variation in the current within the system is less than thirty percent. Most of the current is supplied by unregulated sources (a sort of poor man's current source). The remainder is provided by tightly regulated, commercially available voltage sources which are distributed along the main power bus and provide current required to maintain regulation. Each component of the power system is small enough for easy installation and maintenance, yet large enough to provide economical power. In addition to being economical, the distribution elements are self-adjusting in case of failure. Consider the two possible modes of failure, over voltage or loss of voltage. If a single unit fails by over voltage, all the other regulated sources shut off. The resulting load of the system imposed by these sources shutting off prevents the voltage from rising to destructive levels. If a single unit loses voltage, remaining supplies have a sufficient reserve of current capacity to provide the necessary current with only a minor degradation in regulation in a localized sector of the system.

TESTING TECHNIQUES

Since logic designers and logic devices never seem to be perfect, all the advances in architecture and mechanical packaging would not be useful unless some techniques were developed to allow the system to be tested. Many approaches to solving the testing problem have been reported.[9,10,11] SYMBOL has unique prob-

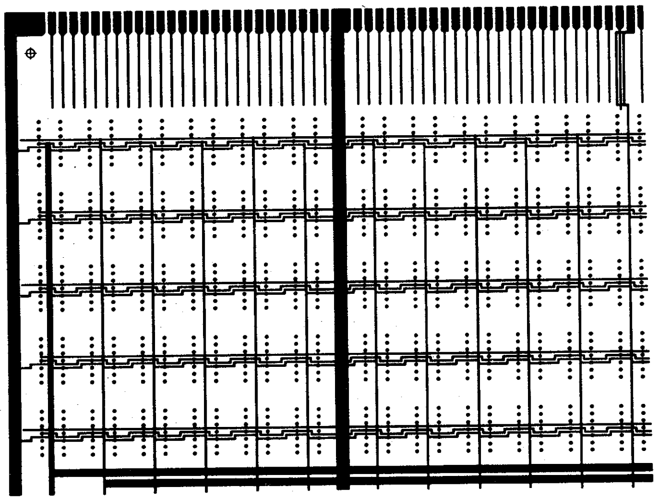


Figure 10—Composite +V, −V, GND power distribution grid on SYMBOL logic boards prior to signal routing

lems analogous to trying to test both the CPU and most of the executive system simultaneously. During design and system partitioning it was found that functional splitting of logic tended to minimize bus lines used. Such a split also became useful during testing. A combination of manual and automatic test aids has been developed to support the SYMBOL project. The SYMBOL documentation, board test, system monitoring, and system test techniques (discussed below) are low-cost approaches to solving the unique SYMBOL testing problems.

Documentation

Documentation of the system was one of the first problems encountered. The documentation used to support SYMBOL fabrication and testing is quite simple. A computer-generated listing from the CAD system describes (in a pseudo-equation form) the logic implemented on each node, the name and reference number of each signal involved, the physical location of the pins on the logic element which implements the node, and the origins and/or destinations of all signals involved. Since all cross-references for a node are given at one point in the document and since the signals are listed in both alphabetic (by signal name) and numeric (by reference number) order, engineers and technicians waste little time trying to find related logic. The CAD system includes comprehensive document editing features to allow easy modification of the original logic and specification of added logic. Both logical and physical descriptions in the document can be edited. The editing features combined with simple procedures to obtain up-to-date documents have provided excellent communication of logic changes to all those involved with the testing process.

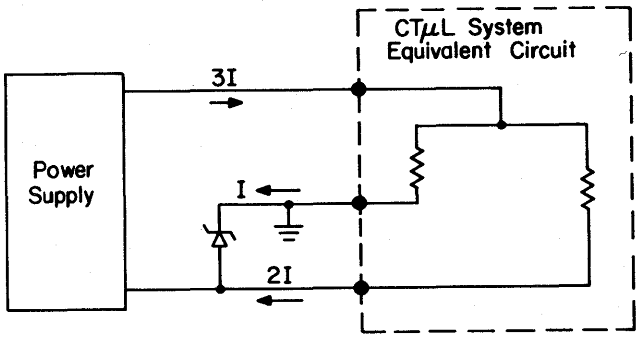


Figure 11—DC equivalent circuit of SYMBOL

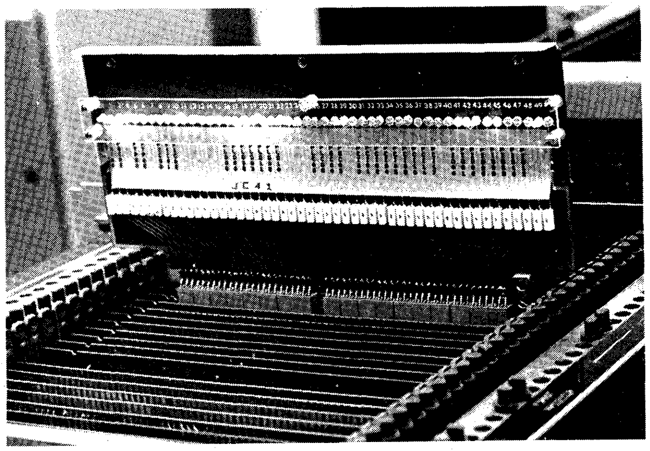


Figure 12—Test point panel used to monitor logic boards during system operation

Test point and bus monitoring

It was noted earlier that the printed circuit logic boards have fifty test points along the top edge. In SYMBOL the designer assigned these test points to signals he thought were of interest. These signals are usually phase-counters and their controls, state flip-flops and central logic decision points. In a production environment, the test points would probably be chosen to allow efficient isolation of fabrication errors (rather than logic design errors as in the prototype). A special indicator panel, called a test point panel, has been designed to connect to the test points and to receive power from the board under test. Figure 12 shows this panel installed on a printed circuit board. Note that there is an indicator light and a probe point for each of the fifty test points. The lamp corresponding to a particular test point will light if the test point is at a high voltage (logic "one"). In some cases, a panel with pulse stretchers between each of the test points and the lamps is used so that transient logic conditions can be detected.

The design of the motherboards allows the straight line buses to be broken at various points in the system. By using breaks and bypasses, the separate AFU's can have a local signal bus in addition to their connection to the global bus lines. To allow monitoring of the signals on these buses, a monitoring panel called a "wing" panel has been designed to insert in a connector on the outside of the motherboard much as a cable connector paddleboard does. Figure 6 shows the wing panel in more detail. Note that the wing panel has one indicator light and one switch for each of the 100 signal pins on the motherboard at the point of connection. The indicator lamp is lit if the associated signal line is

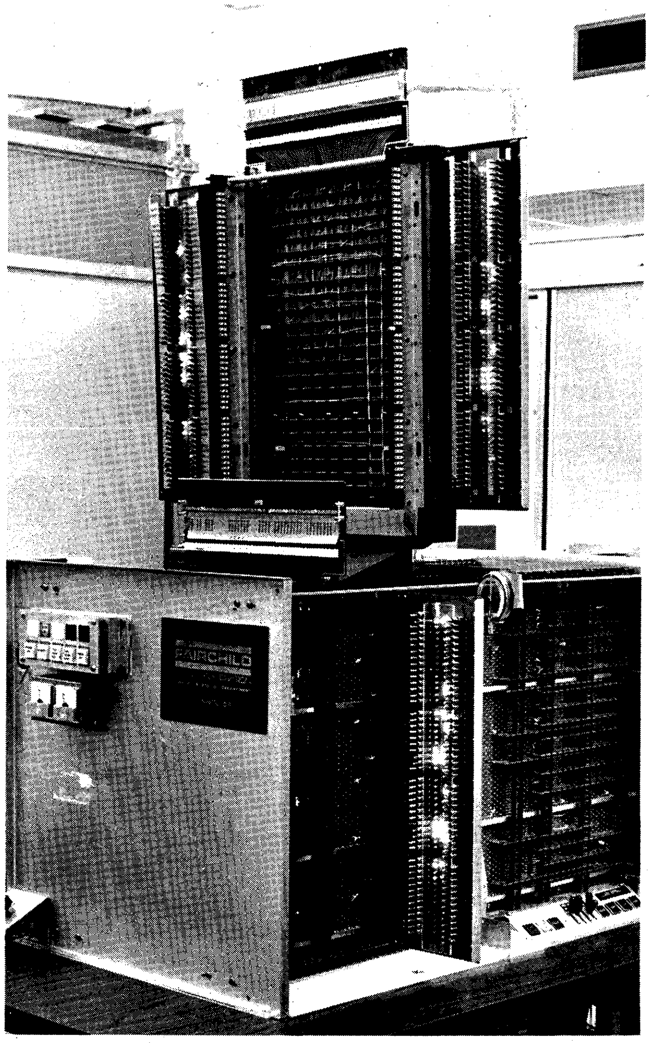


Figure 13—Typical SYMBOL system test configuration with board extender, test point and wing panels in place

at a high voltage and off otherwise. Two types of wing panels are available, a right wing panel and a left wing panel. The two types allow viewing of wings on each of the two motherboards from the same end of the SYMBOL system. One of the reasons that the bused-signal concept is successful is that the CTμL family provides a true wired-or capability. Because of this capability, a switch on a wing panel can be used to force (i.e., "or") a logic "one" on the associated signal bus on the motherboard. It should be noted that the test point panel and the wing panels are also standard two-sided printed circuit boards.

Unfortunately, it is necessary, from time to time, to probe signals on the logic board itself during operation of the system. To do this, a board extender was developed, again using two-sided printed circuit boards and simple mechanical parts. The wing panels can be connected to the board extender also. Figure 13 shows a typical system debugging setup with test point panels, wing panels and extender in place. Because of the system timing rules followed during design, no problems have been encountered due to the extra signal propagation time when a logic board is extended.

Logic board testing

In the early stages of the SYMBOL project, an automatic programmable logic board tester was constructed. However, it soon became clear that too great an investment of engineering time was required to perform successful tests. The major problem was that the boards being tested were not necessarily logically correct. An added complication was that the tests were programmed by hand. As a result, the tests required an excessive amount of time to program and as much time was spent debugging the test program as debugging the board under test.

A human-factors analysis of the specific operations required for functional board testing (in combination with the usually limited R and D budget) resulted in the development of a board tester used for most of the SYMBOL project. This tester (Figure 14) was designed to minimize the number of physical motions required during manual testing of prototype logic boards. In addition, every attempt was made to eliminate the need for any other instrumentation during testing. The same test point panel and wing panels used for system testing mount on the board tester and are used to force and/or monitor logic conditions at the 200 signal pins and fifty test points on the boards. Switch bounce is no problem because board testing is not dynamic in this tester.

During functional board test, signals on the logic board must also be traced and/or monitored. The tester control panel has been designed to help support these requirements. A single clock pulse with the same dynamic characteristics as seen in the SYMBOL system is generated by the CLOCK switch. The LOGIC test probe is used to monitor signal points on the board under test. The condition of the signal probed is indicated by both an audio and a visual signal. Three indicator lamps, one on the control panel and one on top of each of the two uprights, light when a logic "one" is probed. However, that is not sufficient to show that a particular signal line is electrically active or that the probe even made contact. Therefore, an audio signal is generated to indicate both logic one and logic zero via high and low frequencies; for an open circuit, no tone is generated. A set of lamps and switches is connected through a cable to a fourteen pin package probe. This probe can be used both to monitor and to test a logic element. All lamps on the control panel have associated pulse stretchers to detect transient logic conditions.

During the initial testing of a logic board, continuity of signal traces is often suspect. The board tester can also assist in continuity checking. When the power to the board under test is turned off, the jack under the board power switch on the front panel is energized with a logic "one". A probe from this jack and the LOGIC probe can be used in place of an ohmmeter to check continuity. A similar operation can be done through use of the LOGIC probe and the package probe. The LOGIC probe generates an audible confirmation of continuity precluding the need for separate observation of a meter.

The ease of use of the board tester is best seen in the average time for testing. The automatic board tester averaged about two man weeks per board. The manual functional tester required about one man day per board. It is anticipated that when a system such as SYMBOL enters volume production it might be economical to return to the automatic tester since the logic design will have been debugged and the investment in tester programming could be amortized over the production run.

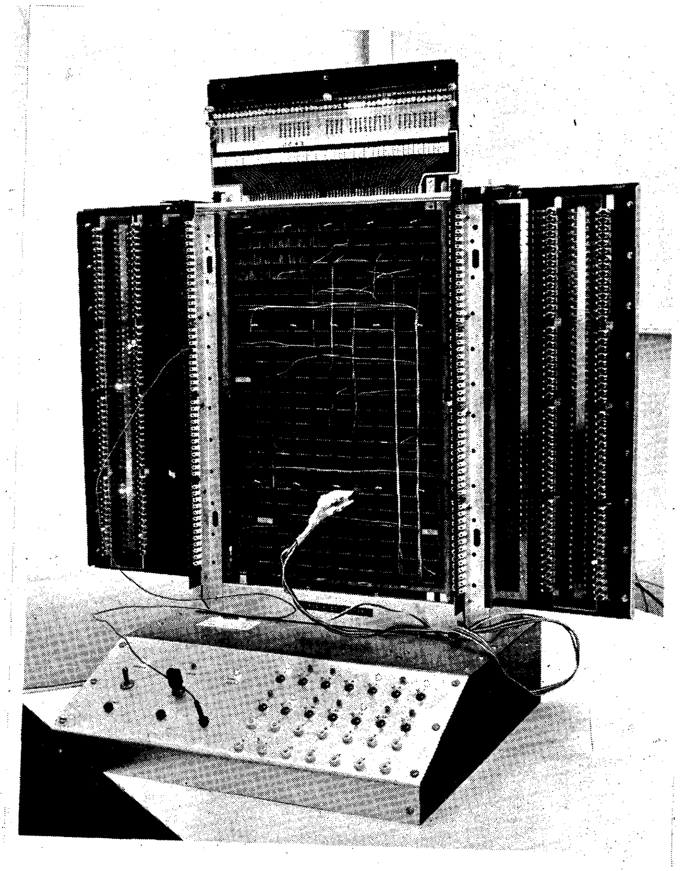


Figure 14—Manual, functional, logic board tester

System test

The problems encountered during system test on the SYMBOL system have been typical prototype test problems. The SYMBOL system is a synchronous design. Thus signals must propagate through whatever logic chains are required before the arrival of the subsequent clock. Although no timing rules were violated in the initial design, some logic corrections could be made less complex if the system timing rules could be relaxed. Also, some of the requirements of the initial system tester (described below) required a longer clock cycle. Thus, a second, slower, system clock speed was introduced at the outset.

Both automatic and manual aids are used for SYMBOL system test. Automatic testing aids are used to help isolate problems to a specific area in an AFU while manual aids are used to isolate the specific problem. Many of the manual test aids have already been shown. The board extender (Figure 13) allows a particular board to be accessible at its position in the system. The wing panels (Figure 7) can be mounted on the outside of the motherboards at positions which were specified by the designers during AFU design. The test point panels (Figure 12) monitor boards in the system. The normal complement of black boxes for monitoring the logic conditions of all pins on a package, for stopping the system clock on a logic condition, for determining how far operation proceeded through a sequential network before halting or forking, etc., has been constructed and is used during detailed logic debugging.

Extremely few areas of system test require the use of an oscilloscope. The electronics controlling the magnetic disk and magnetic core memories requires an oscilloscope to set up and verify timing. The portions of the system involved with various I/O clock frequencies such as peripheral controllers and transceivers also often require an oscilloscope. Very few other system problems have been discovered through the use of oscilloscopes. These few problems generally have been fabrication errors such as the case where the power supply filter capacitors were incorrectly inserted between ground and a signal line (resulting in *very* slow rise and fall times of the signal).

It is impractical to use manual techniques for system debugging. Such techniques would be analogous to single stepping an executive system program to find errors in loops buried deep within a software system.

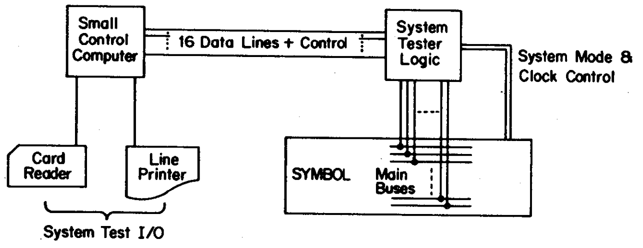


Figure 15—Programmable SYMBOL system test support configuration

Some kind of program trace must be generated. The analogy is particularly apt in the case of SYMBOL since the hardware includes most of the executive system. The SYMBOL system construction schedule was such that I/O equipment was one of the last things to become operational. Thus, self-diagnostics were impractical until late in the program. Some way of exercising and/or monitoring various AFU's and portions of the system automatically and of recording the results of these tests was absolutely necessary. In response to that need the SYMBOL system tester was developed (See Figure 15).

The system tester consisted of a small control computer with on-line disk storage, card reader, line printer and typewriter, interfaced to the SYMBOL main buses and to the SYMBOL mode and clock control lines. Since all SYMBOL system operation is coordinated via signals on the main signal bus, the control computer was programmed to substitute for any missing part of the system and to exercise any portion of the system accessible via the main bus. In addition, the main bus could be "watched" during independent system operation and desired data could be recorded. The discrepancy between the speed of the control computer with a limited bandwidth data channel and the SYMBOL clock speed was resolved by allowing the interface to turn off the SYMBOL clock temporarily when any data needed to be transferred to or from the main bus buffer register in the interface. Control of the clock in this manner significantly reduces the execution rate of the SYMBOL system while monitoring the actions of the system although this approach is considerably faster than simulation techniques. Except for real-time support logic, all logical sequences were executed in the same order with respect to each other no matter what the actual clock speed.

The general use of the system tester consisted of first initializing the memory with appropriate data, setting the system mode to AFU test or system test, starting the appropriate portion of the system, and monitoring the system operation via the line printer. Sixteen extra data lines, in addition to the SYMBOL main bus lines were provided to allow arbitrary signals in the system to be patched into the system tester. These lines could be connected to local buses, local control lines, etc. A language called DEBUG was developed to allow direct execution of any of the virtual memory operations and system supervision cycles under control of the small computer. The system supervision cycles allowed starting and stopping of the various AFU's in the system. An automatic memory exercising mode was included. Any arbitrary pattern could be written into all words, read and checked. Any errors detected were automatically listed on the line printer. The "watching" of the bus signals was done in a TRACE mode. During TRACE, all memory operations performed by a particular AFU and/or inter-AFU control communication were listed on the line printer. Since the control computer was programmable, special problems such as illegal use of particular memory words could be tracked through the use of special monitoring programs.

To provide independent maintenance and debugging capability, a substitute for the programmable test system was devised. An additional AFU called the Maintenance Unit (MU) was defined and implemented. Figure 16 shows the final system configuration of SYMBOL with the Maintenance Unit. I/O equipment from the remote interactive batch terminal is used as the input/output medium. A special punched card format (produced by the SUPERBUG language processor) was defined and is used as input to the

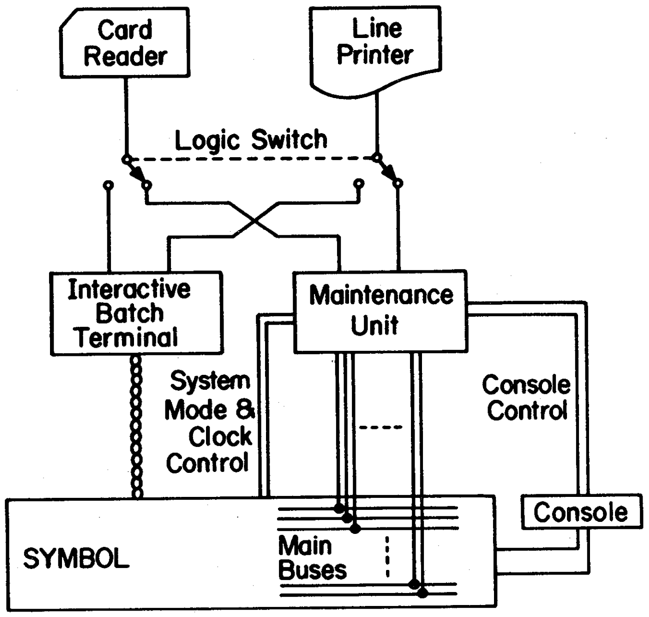


Figure 16—SYMBOL system maintenance unit configuration

DEBUG operations which are hard wired into the MU. The MU implements a majority of the most frequently used features of the programmable system tester and does not implement those features which were relatively unused. The SYMBOL control console also controls the MU.

The use of the limited bandwidth channel from the SYMBOL main buses to the control computer in the system tester proved the feasibility of performing remote testing of complex computing equipment. No good solution has yet been discovered for isolating errors in peripheral equipment using these techniques. The small number of system interconnections has been instrumental in making this system testing approach possible.

CONCLUSIONS

The SYMBOL project has developed a generally useful and inexpensive technology which includes system packaging, computer-aided engineering design and circuit board and system testing techniques.

The system packaging techniques use two-sided printed circuit boards for both signal and power distribution intraconnections. A cam-operated zero-insertion force, high pressure connector gives reliable but inexpensive pluggable connections. All system level interconnections are made on two-layer printed circuit motherboards. No wire wrap or other forms of wired connections are used. The cam-operated contacts solder directly onto the motherboards for mechanical stability and electrical reliability. Simple forced air cooling is used. Inexpensive power is provided by a high current main supply and a low-voltage, lower-current shunt supply.

A complete engineering design package using computer aids was developed for the project. From logic equation input, electrical and timing checks are made (human aided), factoring and functional unit selections are made, automatic placement is accomplished, wire routing is done, and finally, printed circuit board artwork is automatically generated.

The testing techniques allow single circuit boards, several boards, a functional unit, several functional units, or the complete system to be exercised. Single boards may be tested in a separate "board tester." Any board in the main frame of the system may be placed on a board extender for scoping, etc. The parallel line motherboards permit "wings" to be placed at any desired main frame location for observing or controlling logical states. The main frame can be driven by a small computer to simulate operating conditions at any level from a single board up to the complete system. A programmed debug package was developed for the small computer to activate the system and to help diagnose trouble.

These SYMBOL technologies have already proven useful in the implementation of the following projects:

- Minus; a smaller-than-mini computer with a 512 word, 9 bit semiconductor memory,
- SYMBOL Terminal; an interactive, remote-batch terminal;
- LSI Memory; the 2048 word $\times$ 64 bit production bipolar memories used for the ILLIAC IV computer Process Element Memories,
- Board Testers; the memory and SYMBOL logic board testers,
- Automatic AC Functional Memory Tester; a special tester for AC functional testing of LSI memory components

We believe that, taken together, these techniques will be very useful in low-cost commercial computing systems for years to come.

ACKNOWLEDGMENTS

The authors wish to acknowledge special contributions to the development of the hardware system and testing techniques. William R. Smith contributed to many facets of the hardware system development and was largely responsible for the CAD system mentioned above. He, together with Theodore A. Laliotis, developed many of the details of testing procedures. Myron A. Calhoun contributed the programming support required for the success of the system tester and, together with Thomas G. Cook, the design of the Maintenance Unit. Wayne Willis contributed materially to the mechanical design of the system.

REFERENCES

1 R RICE W R SMITH
SYMBOL: A major departure from classic software dominated von Neumann computing systems
This volume

2 W R SMITH R RICE G D CHESLEY
T A LALIOTIS S F LUNDSTROM
M A CALHOUN L D GEROULD T G COOK
SYMBOL: A large experimental system exploring major hardware replacement of software
This volume

3 G D CHESLEY W R SMITH
The hardware-implemented high-level machine language for SYMBOL
This volume

4 R B SEEDS W R SMITH R D NEVALA
Integrated complementary transistor nanosecond logic
Proceedings of the IEEE Dec 1964 Vol 52 No 12 pp 1584-1590

5 R RICE
Functional design and interconnections: Key to inexpensive systems
Electronics Feb 7 1966 Vol 39 No 3 pp 109-116

6 D B JARVIS
Effects of interconnections on high speed logic circuits
IEEE Transactions on Electronic Computers October 1963 Vol EC-12 pp 476-487

7 W T RHOADES
Logic circuits don't live alone
Electronics March 6 1967 Vol 40 pp 162-164

8 W T RHOADES
Power distribution considerations for nanosecond circuitry
IEEE Pacific Computer Conference T-147 1963 pp 139-154

9 R E FORBES D H RUTHERFORD C B STIEGLITZ L H TUNG
A seld-diagnosable computer
FJCC 1965 Proceedings pp 1073-1086

10 K MALING E L ALLEN
A computer organization and programming system for automated maintenance
IEEE Transactions on Electronic Computers Dec 1963 pp 887-895

11 S SESHU D N FREEMAN
The diagnosis of asynchronous sequential switching systems
IRE Transactions on Electronic Computers August 1962 pp 459-465

SYMBOL—A large experimental system exploring major hardware replacement of software

by WILLIAM R. SMITH, REX RICE, GILMAN D. CHESLEY, THEODORE A. LALIOTIS, STEPHEN F. LUNDSTROM, MYRON A. CALHOUN, LAWRENCE D. GEROULD, and THOMAS G. COOK

Fairchild Camera and Instrument Corporation
Palo Alto, California

INTRODUCTION

The SYMBOL system is the result of a major developmental effort to increase the functional capability of hardware. Part of the charter of the broad based project was to reexamine the traditional division between hardware and software, to reexamine the respective roles of program instruction and data storage, and to reduce the overall complexity and cost of computing.[1] In order to adequately evaluate the concepts that had been developed it was concluded that an experimental, usable, real system must be built. The SYMBOL system, now operational, is the embodiment of this effort.

The system was developed in an environment with hardware and software design considered in common. Virtually no one associated with the project could refer to himself as a hardware or software specialist exclusively. As an example, the logic design of the field process units was done by an individual with a basic programming background.[2] The wire routing automation was developed by an engineer who was formerly a pure logic design specialist.

Even before the system became operational much had been learned about the practical aspects of building highly capable hardware. No claim is made that SYMBOL represents an optimum general purpose, time-sharing, multiprocessing system. In contrast, numerous simplifying assumptions were made in the system where they did not serve the goals of the project. Certain modularity restrictions are examples of this. It is claimed that SYMBOL represents a significant advance in systems technology and provides the foundation for a significant reduction in the cost of computing. As the system moves into an intensive evaluation phase it should prove to be a real asset for advanced systems research.

This paper represents an overview of the SYMBOL organization. An attempt is made to give simplified examples of various key features in contrast to a broad brush treatment of many topics.

GROSS ORGANIZATION

The system has eight specialized processors that operate as autonomous units. Each functional unit is linked to the system by the Main Bus. See Figure 1. Consider some of the features of the system and their relationship to the gross processor organization as outlined in the following sections.

Dynamic memory management

Direct hardware memory management is perhaps the most unique feature of the SYMBOL system. The memory management centers around a special purpose processor called the Memory Controller (MC). The MC effectively isolates the main memory from the main bus and the other processors and in turn provides a more sophisticated storage function for the various processors. In contrast to simple read/write memory operations the MC has a set of fifteen operations that are available to the other processors of the system. The MC is a special purpose processor that allocates memory space on demand, performs address arithmetic, and manages the associative memory needed for paging. The Memory Reclaimer (MR) supports the MC by reprocessing used space to make it available for subsequent reuse. It is a separate unit to allow the task to be performed using a low priority access to the memory.

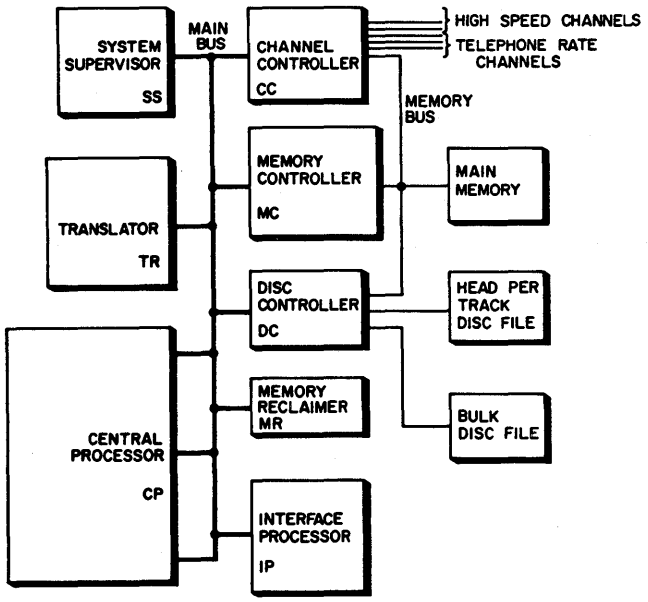


Figure 1—Gross block diagram of the SYMBOL system

Direct compilation

The Translator (TR) accepts the high level SYMBOL language[3] as input and produces a reverse Polish object string and name table suitable for processing by the Central Processor (CP). The TR performs the direct hardware compilation using only a small table of about 100 words stored in main memory.

Dynamic variable field length

Within the Central Processor all field processing is done with dynamically variable field lengths. All alphanumeric string processing is done by the Format Processor (FP) while all numeric processing is done by the Arithmetic Processor (AP). The resources of the MC are used extensively by the CP in handling the storage of data.

Dynamically variable data structures

Complete variability of data structures is allowed. They can change size, shape, and depth during processing. Within the CP the Reference Processor (RP) manages the storage and referencing of all data arrays and structure. The MC functions are used extensively by the RP.

Time-sharing supervision

The System Supervisor (SS) is the task scheduler for the system. All transitions from one processing mode to another are handled by the SS. Queues are maintained for all of the time-shared processors. The SS executes two important hardware algorithms, job task scheduling and paging management. A real-time clock is used in the process of rationing out critical resources such as central processor time. The SS also performs key information transfers needed to tie hardware algorithms into software system management procedures.

Direct text editing

The Interface Processor (IP) and Channel Controller (CC) perform the input/output tasks of the system. The IP has ability to handle general text editing in support of interactive communication via a special terminal. Input/output and text editing do not use the CP resources.

Virtual memory management

When the MC detects that a page is not in main memory it notifies the requesting processor and the system supervisor. The SS then utilizes a paging algorithm to supply the appropriate disk transfer commands to the Disk Controller (DC). Each user of memory must, upon receiving a page-out response, be able to shut down and save its current state and status and restart after paging is complete.

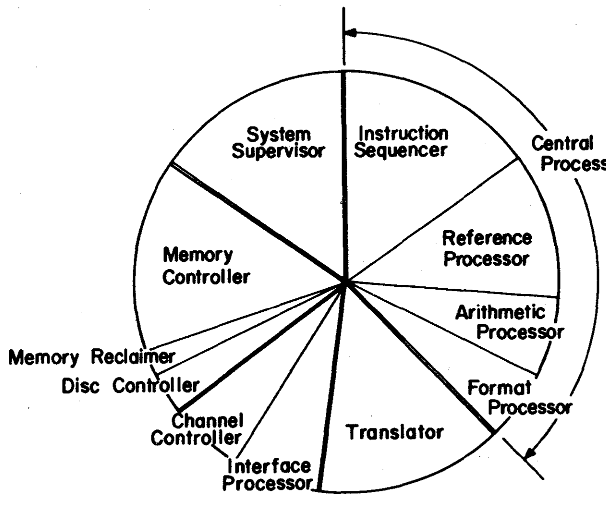


Figure 2—Breakdown of the SYMBOL hardware showing the relative sizes of the various processors

SYSTEM CONFIGURATION

The system has a small complement of peripheral and storage equipment associated with the main frame. This complement of equipment has proven sufficient for the experimental purposes of the system. The main memory is an 8K word × 64 bit/word core memory with a cycle time of 2.5 microsec. It is organized into 32 pages with 256 words/page. The main paging memory is a small Burroughs head-per-track disk divided into 800 pages. The bulk paging memory is a Data Products Disk-file organized into 50,000 pages.

The Channel Controller is designed to handle up to 31 channels. This low limit was deemed sufficient for evaluation of the experimental system. As of this writing one high speed (100,000 bits/sec. effective data rate) channel and three phone line (up to 2400 baud) channels have been implemented. More can be added during the evaluation phase.

The main frame contains about 18,000 dual in-line CTμL components. Its physical properties are described in other papers.[4, 5] In order to get a relative measure of the size of the various autonomous processors a chart is given in Figure 2.

SYSTEM COMMUNICATION

The main bus of the system is a time-shared, global communication path. It uses the special properties of the CTμL family in its implementation.[4, 5] The bus contains 111 parallel lines. They are distributed as follows:

Data Bus	64
Address Bus	24
Operation Code Bus	6
Terminal Identification Bus	5
Priority Bus	10
System Clock	1
System Clear	1

Four types of bus usage are available. They are:

Processor to MC transfers
MC to Processor transfers
Processor to Processor transfers
Control exchange cycles

The basic information transfers are priority sequenced. The priority bus indicates the desired bus usage for the following cycle; if a unit desires to use the bus it raises its priority line and then checks the priority bus to see if there are any higher priority requests. If not it uses the bus on the following cycle.

Control exchange cycles are used to communicate control information between the SS and the various processors over the data and address buses. See Figure 3. During a control cycle the data and address bus lines have preassigned uses. Certain lines are used to start the CP. Others indicate the completion mode for the TR. During a given cycle any combination of the paths can be used. The SS has autonomous interface control functions that are used to communicate with the processors during control cycles so that more than one control signal can be transmitted during a given cycle.

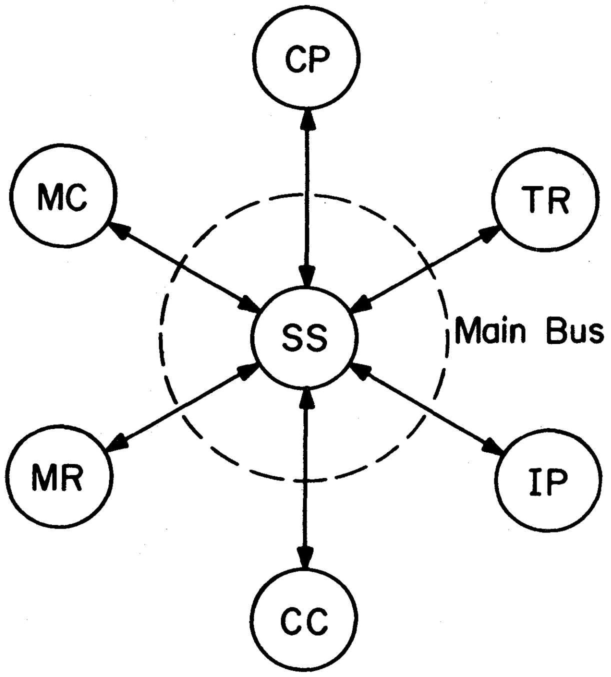


Figure 3—Use of the main bus for control exchange cycles

MEMORY ORGANIZATION

Virtual memory

The SYMBOL memory is organized as a simple two-level, fixed page size virtual memory.[6] The page has 256 words with each word having 64 bits. Virtual memory is accessed by a 24 bit address with 16 bits used to select the page and 8 bits to select a particular word within a page. See Figure 4.

The main memory for the experimental system is logically divided into 32 pages. The relative portion of the address is used directly while the page number accesses an associative memory which in turn supplies the current page address in main memory.

The associative memory has one cell for each page

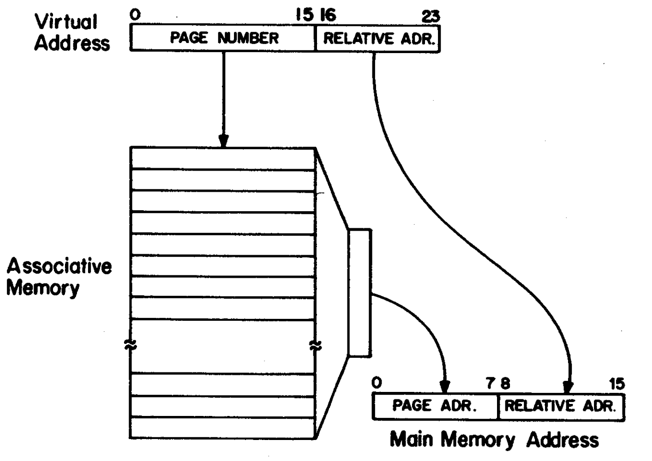


Figure 4—The simple two level addressing structure for the virtual memory

in the main memory. By providing an associative memory tied to the main memory the individual processors need not be concerned with the location association process. This provides a significant reduction in the logical complexity of the processors even though it may lead to slightly more overall electronics.

The paging disk memory has fixed assignment of page locations. See Figure 5. A page is brought into an available location in main memory upon demand. When it is purged back to disk it is transferred back to the same location on disk. (The return transfer is omitted if the page was not changed in main memory.)

The main memory organization is shown in Figure 6. The first page is used for system tables. This includes a reserved word table for the translator, a software call table, and the control words for memory allocation and queuing. The next set of pages are used for storing the control words of the various terminals or users on the system. Each active terminal has 24 words of control information in low core. Much of the control information could have been placed in virtual memory as would certainly be required for a system with a larger channel capacity. As a simplifying restriction for SYMBOL all channel tables were placed in main memory.

The input/output buffers for the various active channels are also held in core. The buffers require 16 words per active channel. Variable buffer sizes although possible were not implemented.

The remainder of main memory is available for virtual memory buffering. Paging is managed by the hardware with the page selection for purging under the control of the system supervisor. The algorithm is a very flexible parameterized process that allows most of the conventional paging algorithms to be executed. The parameters are maintained for each terminal so that the paging dynamics can be tailored on a terminal by terminal basis.

The virtual memory organization is quite simple for SYMBOL in contrast to the more common segmentation schemes.[7,8] The primary difference that allows the simplified approach to be taken in SYMBOL is that contiguous addressing above the page level is not needed. All users and channels share the same virtual memory space. The 24 bit address space is thoroughly used. With space allocated only upon demand and with no restriction on a scrambled assignment of pages to users it is anticipated that 24 bits will be sufficient for many more than the 31 possible terminals. If file space is needed beyond the 24 bits of address space it can be addressed via special block input/output transfers.

Page lists

Pages are associated together with the use of linked page lists. Pages available for assignment are main-

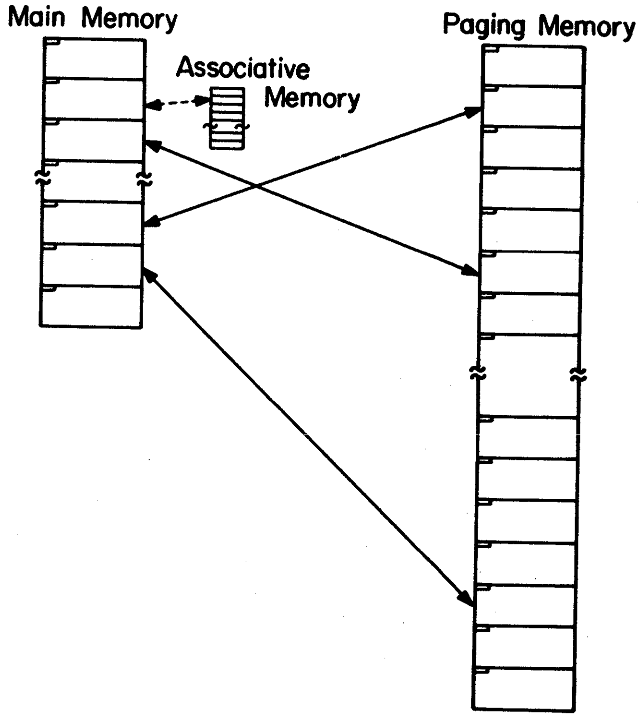


Figure 5—Virtual memory organization showing the fixed location of pages in the paging memory

tained on an available page list. As each user needs space a user page list is started by transferring a page from the available page list to the particular user. A control word is established at this time as a focal point for all future page list management for the user. As more space is needed pages are added to form a variable length storage area for general purpose usage. See Figure 7.

A given user may have more than one page list. Typical page list usage for a terminal would be one page list for program source text, another for the compiled object program, and a third for data variable storage. Other page lists are used for long or short term file storage.

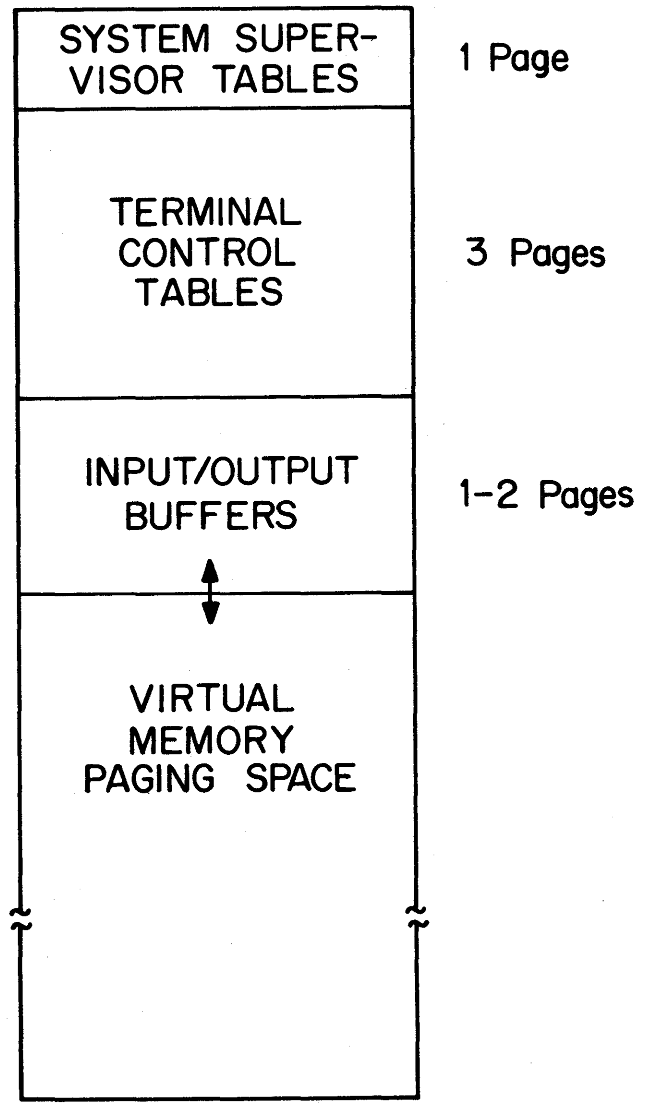


Figure 6—Layout of main memory

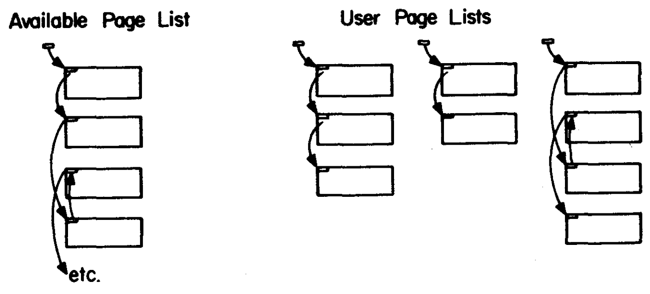


Figure 7—Simplified page list structure within the virtual memory

Page lists grow monotonically as space is needed. When an entire list is no longer needed it is given back to the system by returning it to the available page list.

Page organization

In order to handle non-contiguous address space a certain amount of the storage space must be devoted to linking or association data overhead. In SYMBOL about 11 percent of the storage space is for overhead bookkeeping.

Pages have three distinct information regions as shown in Figure 8. The first region called the page header is used to maintain the page lists and manage the space within the page. The second region is a set of 28 words. The third region is a set of 28 groups with each group containing eight words. Each group has a corresponding group link word associated by a simple

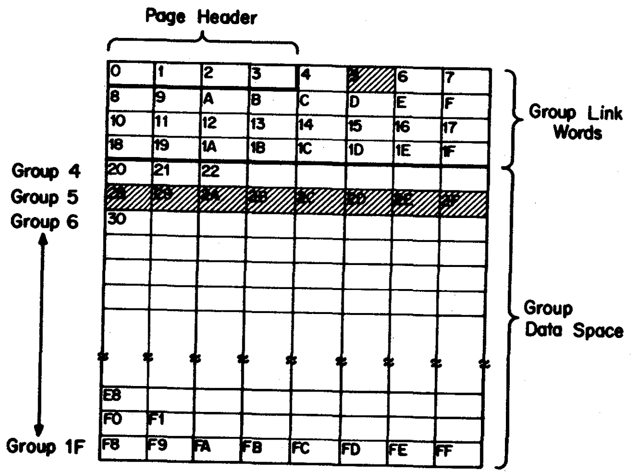


Figure 8—Page organization showing group and link word layout where addresses are given in HEX notation

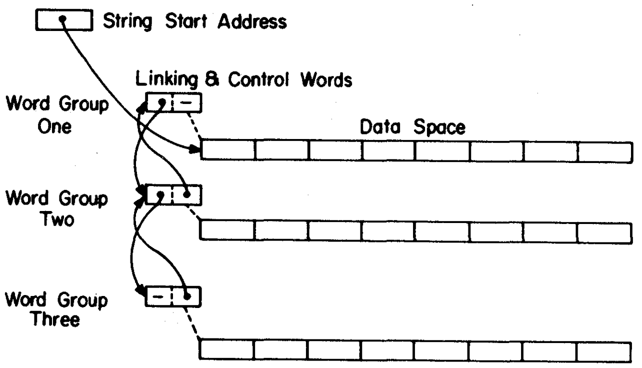


Figure 9—Structure of a variable length string

address mapping. Consider in Figure 8 word 5 and the corresponding group 5. Data is stored in words 28 through 2F. This eight word group is the fundamental quantum of space allocation. It is the smallest amount of memory space that is assignable to a given purpose.

When data is needed for some purpose groups are assigned. For example, if six words were needed to store a data vector one group would be assigned. If space for a vector of 14 one word items were needed two groups would be assigned. Variable length information areas are developed by chaining together these basic units of storage.

Information strings

Variable length lists of storage locations are used for general information storage in SYMBOL. They are logically contiguous memory cells but not necessarily physically contiguous cells.

Consider a typical variable length information string in Figure 9. Data space for 24 words of information is tied together by way of the associated group link words. If access to the start of the string is known it is possible to follow the entire string by accessing the corresponding group link word each time the end of a group is encountered. It is also possible to traverse the string backwards by using the back links also stored in the group link word.

Each processor uses the variable length storage service of the memory controller (MC) without cognizance of the address sequence that is involved. For example, when a processor needs space to store a vector of data fields an Assign Group (AG) command is sent to the MC along with a tag specifying a page list with which the string is to be associated. The MC then selects an available group from the page list and returns the address of the first word of the group to the requesting processor. When the processor is ready to store a word it transmits the data and the address previously assigned to the MC along with the command Store and Assign (SA). The MC stores the word and generates the address of the next available word. When the end of the group or string is encountered the MC assigns another group and links it into the string.

In the string storing process the requesting processor receives addresses from the MC and resubmits them to the MC at a later time for future extension of the string. All address arithmetic is done by the MC. Consider the example in Table 1. The first five commands result in the words A, B, C, and D being stored in a string beginning with word A.

To reaccess the string the original start address A is submitted to the MC with the Fetch and Follow (FF) command. The data in cell A is returned along with the next address in the logical sequence. When the string is no longer needed a Delete String (DS) command along with the string starting address is submitted to the MC. The entire string is then placed on a space reclamation list. The Memory Reclaimer processor scans the space reclamation lists of the various page lists during idle memory time and makes the groups of the deleted strings reassignable.

The basic memory usage process deals with variations of the AG, SA, FF, and DS operations. Eleven other memory commands are available to give a full memory service complement.

Space utilization efficiency was an important aspect of the SYMBOL memory design. Studies have been made into the optimum size of the space allocation group.[9] The trade-offs center on balancing the linking overhead cost and the unused group fragments cost. The overhead cost is compensated by the allocation on

TABLE I—Simplified example of a memory usage sequence

SIMPLIFIED EXAMPLE OF A MEMORY USAGE SEQUENCE

MNEMONIC	OPERATION	ADDRESS TO MC	RETURN ADDRESS	DATA TO MC	RETURN DATA
AG	Assign Group	-	a	-	-
SA	Store & Assign	a	b	A	-
SA	Store & Assign	b	c	B	-
SA	Store & Assign	c	d	C	-
SA	Store & Assign	d	e	D	-
FF	Fetch & Follow	a	b	-	A
FF	Fetch & Follow	b	c	-	B
FF	Fetch & Follow	c	d	-	C
FF	Fetch & Follow	d	-	-	D
DS	Delete String	a	-	-	-

demand approach. In most machines, fixed size data arrays are allocated to their maximum needed size. When the average array usage is considered a substantial amount of demand allocation overhead can be afforded before approaching the normal excess fixed allocation space usage.

INFORMATION FORMS

Data fields

Two basic data types are defined in the system, namely string and numeric fields. The string field is characterized by a special String Start (SS) character followed by a variable length set of ASCII alphanumeric characters terminated by a special String End (SE) character. This illustrates perhaps the most significant aspect of all SYMBOL data representations. The type and length of the datum is carried with the datum. The instruction code is independent of the dynamic attributes of the data.

The second data type is a variable length, packed decimal, floating point number. The numeric form also carries a designator of the class of precision. Numbers may be *exact* with an infinite number of trailing zeros implied or they may be *empirical* implying that all following digits are unknown and cannot be assumed present for calculation purposes. Like the string field all attributes of the datum are carried by the datum itself.

As a simplifying hardware design decision other forms of data fields were not implemented. It is straightforward to extend the SYMBOL concepts to packed variable-length binary strings, fixed length binary numerics, variable length binary numerics, etc. In any of these cases the datum must carry a type designator and an explicit or implicit designation of field length.

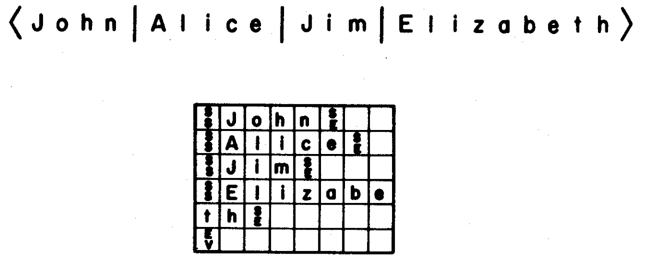


Figure 10—A vector of string fields and the corresponding representation of the data in memory

Source programs

Source programs are special forms of string fields. They are variable length ASCII character strings with delimiters defining length and type. They can be treated as data fields during preparation and then later used as program source for compilation. Source procedures may be assembled into libraries of various forms as long as they retain the string field attributes for compilation purposes.

Data structures

Data structures are defined as a variable length group of items where an item may be a string field, a numeric field, or another group of items. With this recursive definition a structure could be a vector, a matrix, or an irregular array. There is no limit to the depth or size of an array providing a field or a group does not exceed the size of main memory during execution.

Consider the example of a simple vector shown in Figure 10. The special graphics <, |, and > have been introduced for representing field boundaries and groupings of fields. They are used to define the extent of variable length fields and referred to as left group marks, field marks, and right group marks respectively. In memory the string fields are delimited by String Start (SS) and String End (SE) characters. Another special character called the End Vector (EV) code terminates a group of fields. The storage representation in Figure 10 shows a series of string fields followed by a special End Vector (EV) code which again is a length indicator with the data. The string fields are aligned to start on machine word boundaries. In the case of Elizabeth two machine words are needed to store the field.

In Figure 11 the matrix representation is similar to the vector example except that two levels of vectors exist. The definition of a structure could be restated as a variable length group of items where an item may be a string field, a numeric field, or an address link to another group.

Object string and name tables

When a program is compiled the translator creates a reverse Polish string with postfix operator notation and a structured name table. The Polish string, called the object string, and the name table are the basic information forms used during program execution by the central processor. The use of a separate name table during execution is perhaps one of the most distinctive departures from traditional processing forms. Where in most systems, the program string to be executed con-

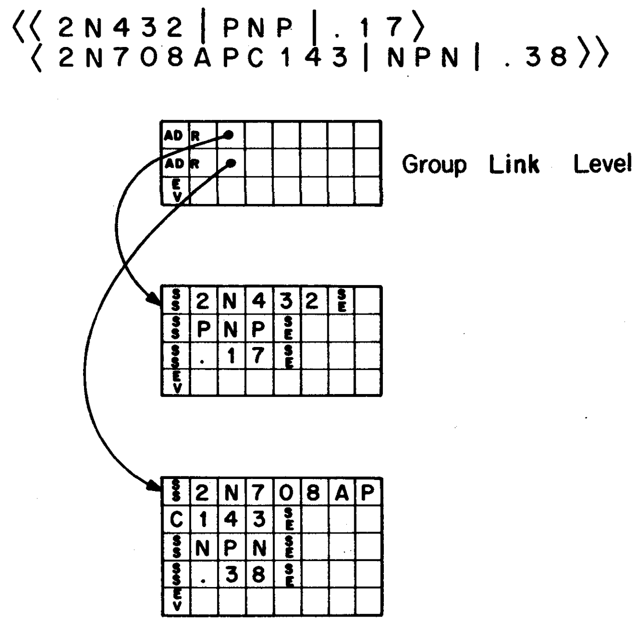


Figure 11—A simple two dimensional array and the corresponding three variable length memory strings that are used

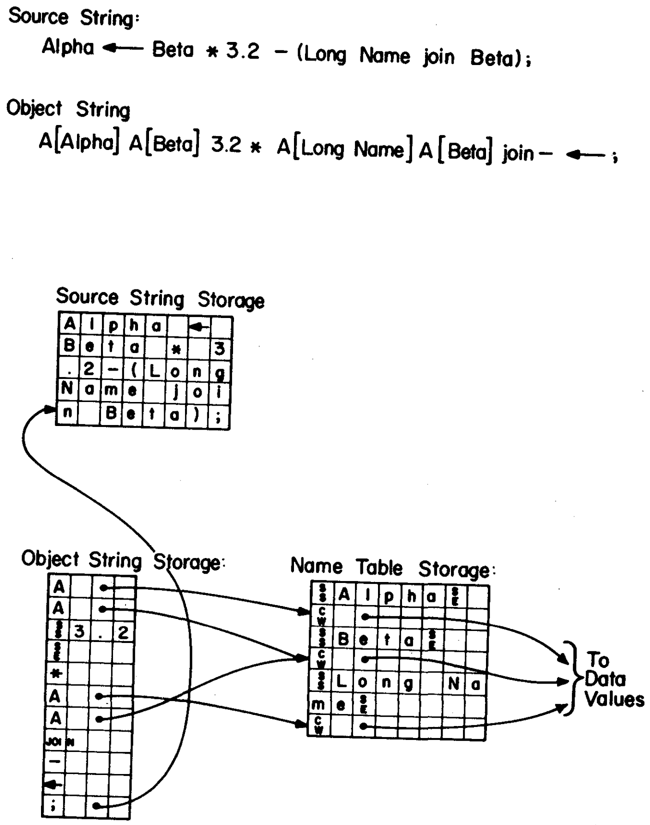


Figure 12—Information structure for a simple assignment statement

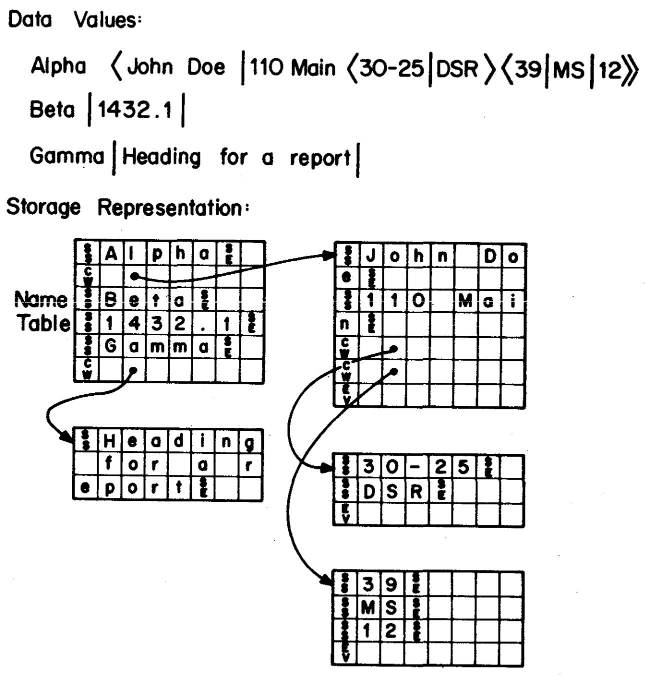

Data Values:
Alpha ⟨John Doe |110 Main ⟨30-25|DSR⟩⟨39|MS|12⟩⟩
Beta |1432.1|
Gamma |Heading for a report|
Storage Representation:
Name Table

Figure 13—Examples of a structure and two fields and how they are stored into memory along with the name table

tains address references to the data space to be utilized, with the SYMBOL system the object string contains references to entries in the name table which act as a centralized point where all information about a given identifier is kept. It is this feature that gives the system its extreme execution time dynamicism. Whenever the nature of an identifier is modified in any way—location, size, type, etc., only the name table entry need be changed since all references in the object string to an identifier must go through this entry.

The source form of a simple assignment statement and the corresponding object string and name table are shown in Figure 12. The identifiers are isolated and added to the name table when not already there. Note that the identifiers can be variable length and have more than one word. Associated with each identifier is a control word. All references in the object string involving the identifier will point to the corresponding control word. The object string is composed of name table addresses, literal data (the value 3.2), operators in postfix representation, and correspondence links back to the source string. The correspondence links are for simple error diagnosis and are therefore ignored during normal execution. The object string and name table are totally independent of the future size and data type of the variable.

Now consider the name table after execution has begun and assume that the data variables have current values. In Figure 13 the variables Beta and Gamma are simple fields. Gamma is a multiword string and therefore it is stored in a memory string with a link address placed in the corresponding control word. Beta is a short field such that it can be stored in one word directly in the name table. Alpha is an irregular structure. The name table for Alpha contains a link to the first group which in turn contains two string fields, two link addresses, and an end vector mark. The link addresses point to two groups, one containing two fields and one containing three fields. As execution progresses the attributes and storage representation of the variables may change. In any event, the name table and the data itself will contain all the attributes of the variables.

BASIC INFORMATION FLOW

In order to observe how the various processors of SYMBOL are used to serve the end users problem temporarily ignore the multiprocessing aspects of the system. A user at a terminal operates in various modes; program loading, program compilation, and program execution are the fundamental usage modes. Consider the state diagram in Figure 14. A user would start in the OFF-LINE mode and by some transitional control means he would initialize his tasks into the ON-LINE IDLE mode. From here he can go into the LOAD mode to develop a program. When he is ready to execute his program and assuming he is a perfect programmer he would have his program compiled and executed. At the end of execution he can restart and rerun his program or he can return to the LOAD mode and modify his program.

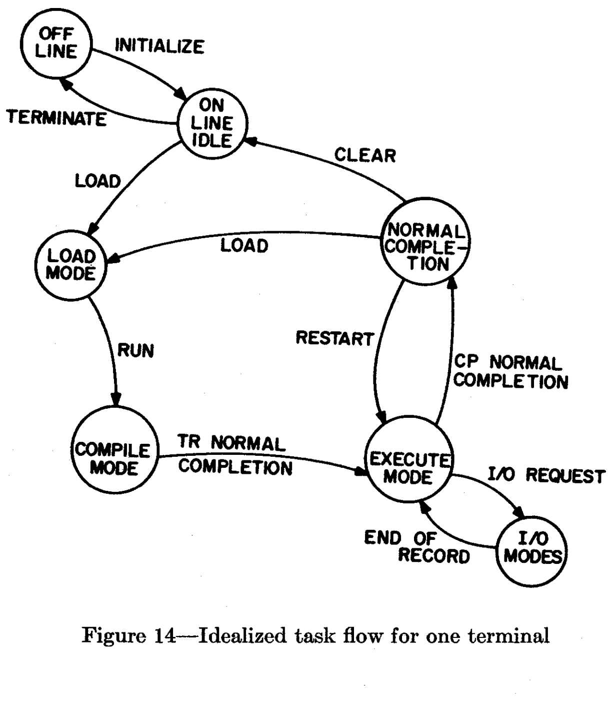


Figure 14—Idealized task flow for one terminal

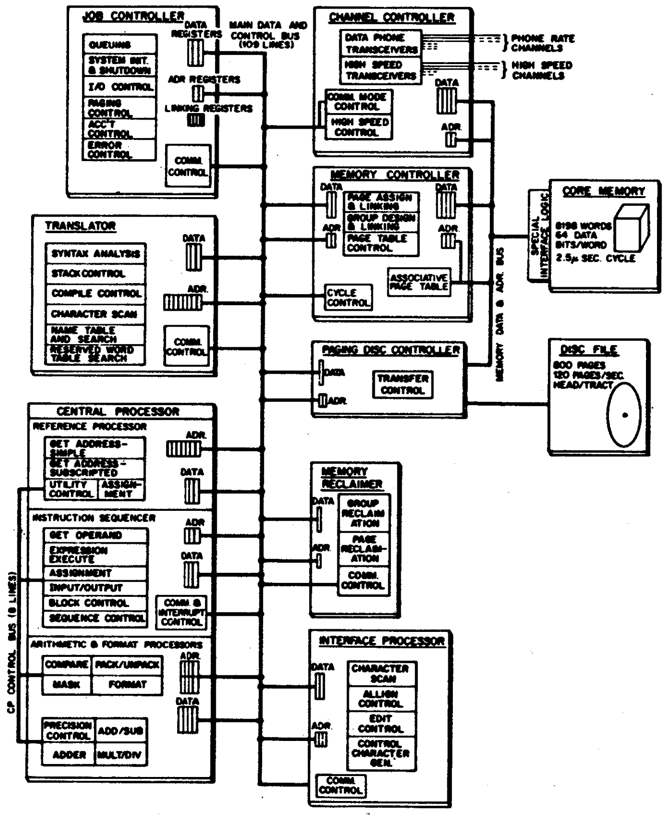


Figure 15—A more detailed block diagram of the SYMBOL system showing register configuration and major functions within each processor

The following sections deal with examples of the information flow for the basic operational modes of a terminal. A more detailed system block diagram in Figure 15 will be used to support the description. Visualize the time sequence of the terminal operational states of Figure 14 in conjunction with the static hardware diagram of Figure 15.

Load mode

The LOAD mode is an input/output text editing mode. Its primary purpose is for program source loading. In the normal case a separate page list is used to store the text string. This area is called the Transient Working Area (TWA).

Three processors work together to perform the text editing tasks. The Channel Controller (CC) transfers data characters to and from I/O devices from and to the I/O Buffers in main memory respectively. When the CC detects control characters in the I/O stream it communicates the control information directly to the

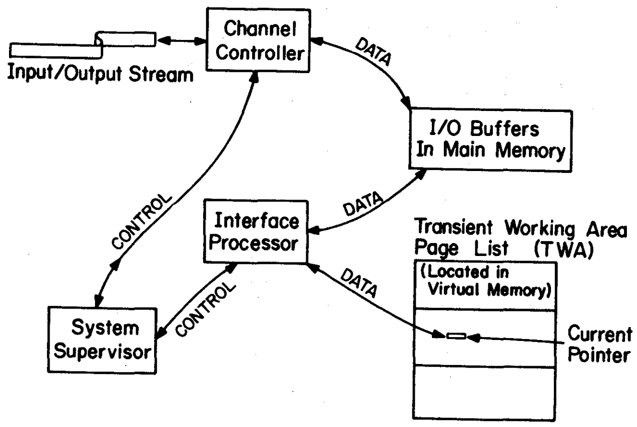


Figure 16—Information flow in the LOAD mode

SS by way of a control exchange cycle. The CC is a character oriented processor which services up to 32 processors in a commutating manner. The CC also has a high speed (block) operating mode which is priority driven to allow servicing of disk and high speed tape devices. The block mode is not used in the LOAD or normal I/O mode.

The Interface Processor (IP) operating on a burst basis empties or fills I/O buffers and transfers appropriate characters to and from the virtual memory. The IP works with a current text pointer while performing its functions. The IP functions include basic text insertion, searching, displaying designated text portions, deletion of designated text portions, and moving the current pointer. In Figure 16 the basic information flow during the LOAD mode is summarized.

Part of the justification for implementing editing functions in hardware came from the desire to eliminate the CP from many of the system overhead tasks. In addition, response times would be unacceptable if the CC were to communicate directly with virtual memory. The IP was developed to make the basic transfers between small buffers and paging memory. Once a special processor was developed it was found that many editing tasks and double buffering could be handled using essentially the same data transfer hardware.

This IP/CC/SS process is available for both LOAD mode data preparation and program execution I/O. The full text editing facilities are available for any program input statement.

Compile mode

Program compilation and address linkage editing functions are performed by the Translator (TR). The TR accepts the language source string from the TWA or some other source text area in virtual memory. The high level language is converted into a reverse Polish string and a structured name (identifier) table. The Polish string, called the Object String, and the Name Table may be stored in Virtual Memory on separate page lists or on a common page list. The gross flow of information in the Translation mode is shown in Figure 17.

The TR performs a one pass compilation generating the object string as it scans the source string. It also builds the name table during this scan on a program block-by-block basis. At the end of the source pass the TR processes the name table and resolves all global references by creating appropriate indirect links. External procedure references are resolved during the name table pass and they are compiled and included with the object string as needed.

The TR includes external procedures by accessing procedure source libraries and compiling needed procedures into the object string. The procedure libraries are organized into two sets, namely privileged and non-privileged procedures. Privileged programs differ from normal programs in that they can contain privileged statements for direct memory manipulation using the MC operations. Storage protection is obtained by controlling the privileged status of user programs and the programs that they can reference. Non-privileged programs have a high degree of storage protection both from other programs and from themselves due to the hardware storage management and central processor algorithms. Programs using privileged statements loose some of the protection. By controlling the access to

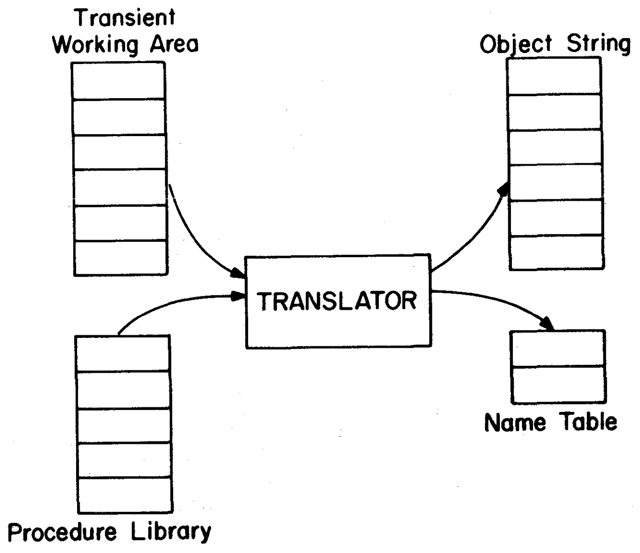


Figure 17—Information flow in the COMPILE mode

privileged programs and the manner in which they are used the overall storage protection in the system is quite satisfactory for multiterminal operation.

Execution mode

The Central Processor (CP) is the execution unit for the translated language receiving the translated source string along with the nested name table blocks as input. Because the CP operates on a high order language—actually a Polish string, postfixed operator object string—the CP uses a push-down stack for its operands. That is, the data reference is generated with all indirections traced out until a memory reference point is reached, and then this reference is pushed into the stack. This process continues until the postfixed operators are encountered in the object string. Each operator causes the top one or two (monadic or dyadic operator) stack entries to be pulled up, processing to take place, and the result to replace the operand(s) on the stack.

Substructure referencing, also known as subscripting, is a much more formidable task in SYMBOL than with conventional systems. This is due to the extremely dynamic flexibility of these structures. With conventional, systems, accessing an element of a vector is a simple matter of assigning a base along with an index register for the subscript variable and at execution time merely doing an address calculation to find the desired element. With SYMBOL there can be no possibility of a base address or an address calculation both because of the dynamic nature of space allocation as well as the fact that logically contiguous data need not be physically contiguous in memory. The Reference Processor (RP) has the charter for finding substructure points, basically through a scanning technique along with several speed-ups.

Another novel aspect of the CP is that all processing operations are done on variable length data. The string operations can be of any length, the only limitation being that they must fit into the main memory. The numeric operations are limited to a 99 digit fractional length (numbers are represented internally as normalized floating-point decimal numbers). Furthermore, the length of numeric processing is controlled by the limit register. Also, a precision mode exists whereby numbers tagged with EM (empirical) will limit processing precision to the number of fractional digits they contain, unless the limit register is set to a smaller value.

The information flow for the CP is summarized in Figure 18. The CP has four distinct sections, namely the Instruction Sequencer (IS), the Reference Processor (RP), the Arithmetic Processor (AP), and the Format and String Processor (FP). As shown in Figure 15 the CP has a common control bus that is used to control the various processors during program execution. The following four sections describe the functions of each of the processors in the CP.

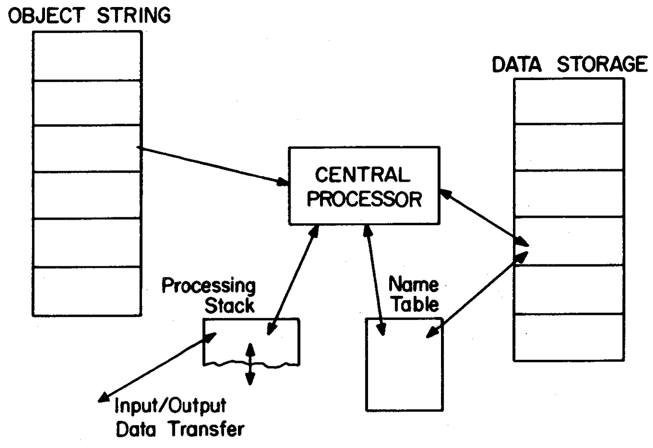


Figure 18—Information flow during program execution

Instruction sequencer

The IS portion of the CP is the master controller and switching unit of the CP. It has the task of scanning the object string, and accumulating items in the stack for the various units it supplies. For example, operands are accumulated for the process units and any type conversion required is sensed and requested of the FP by the IS, as appropriate. Similarly, a structure reference and all of its subscripts are computed and placed into the stack which is then turned over to the RP for access.

The IS also prepares data for assignment by the RP or output by the I/O unit. It does this in the former case by stacking both the assignment reference and the data and in the latter case by stacking the data and turning control back to the system.

Another major task performed by the IS is that of dynamically creating nested language blocks. Reference should be made to the companion paper on the SYMBOL language[3] if the concept is new to the reader. In quick review, blocks are language constructs consisting of program segments contained between the reserved words BLOCK and END (PROCEDURE and ON also establish blocks). Within a block, all uses of an identifier are local to that block, unless contained within a GLOBAL statement, and thus a different name table is constructed for each block. The overall structure of name tables has a static aspect determined by the way the program is written and a dynamic aspect determined by the sequence in which these blocks are executed. It is this latter characteristic that we are concerned with in this discussion. Whenever a new block is encountered by the IS, processing on the old block is suspended by pushing down all information about that block that must be retained (sometimes called the activation record) into the stack, and starting a new stack and activation record for the new block on top of the old stack. Of course, the new record must contain a link to the old record so that when the new block is completed, the old block with its status information can be reestablished.

A further complexity occurs with procedure blocks because of the need to correlate actual and formal parameters (again, see the language paper).[3] The IS transfers the links to the actual parameters from the object string to the stack, accesses the name table for the new block where the formal parameters occur as the first entries of this name table. The actual parameter links are then placed one-by-one into the formal parameter entries of the name table. Parameter linking completed, the remainder of the normal block action for the procedure is accomplished. Whenever the IS encounters a name table entry tagged as a formal parameter, it indirectly accesses the actual parameter in its place, which may not be a statement; but may be a variable, constant, label, literal, procedure, or expression. This indirection mechanism is also handled in the IS stack. A push down of a limited set of status information takes place, mostly consisting of the address where execution of the object string was temporarily discontinued. Then the new object string of the actual parameter is executed, using the stack until the return operator is encountered indicating the end of the actual parameter string. This causes the previous status to be recovered from the stack and execution of the object string recommences with the results of the execution of the actual parameter remaining in the top of the stack.

Reference processor

The basic task of the RP is to deal with structures. As a simple added duty, it accesses the address of an item from the name table for the IS. That is, the IS receives an address from the object string and turns it over to the RP with a request to "get simple address." The RP performs several actions depending on the nature of the identifier. If it is an existing data item it provides the address of the data along with a code indicating its nature. If it is an uninitialized data item, it first assigns space before supplying the data address. In a similar manner it provides links to labels and procedures and if any identifiers are global, it first traces out the global indirection before returning the link. Any anomalies in the name table cause an error return.

The structure handling task may be broken down into two subtasks: creation of structures and substructure and the referencing of substructure points. Recall that structures are dynamically variable in all aspects. Thus, there are two further subsets under the creation of structures: creation of basic structures and the reconfiguration of substructures. As a subset task to the referencing of substructures the language contains a character subscripting capability where the final subscript may be a "bound-pair" of subscripts which refer to the starting point and extent of a character subfield with the previous subscripts pointing to the field.

The RP receives a linear representation of the structure to be created in the IS stack. The RP must store this structure in memory, replacing its linear form with a hierarchical form with links to lower or deeper elements occurring at the next higher level. Refer to Figures 10, 11, and 13. It achieves this by assigning a new memory group each time it encounters a new left group mark, creating a line to the new group in the higher group and filling that group with elements maintaining a link back to the higher group in its own group link stack. Whenever a right group mark is encountered in the IS stack, the current memory group is closed with an "end vector" tag and the next higher memory group continuation point is accessed from the group link stack. This process continues until the structure in the IS stack is exhausted and results in a linked, hierarchical structure.

A similar process takes place when a new structure is assigned to an existing substructure point. The old structure is deleted (for later recovery by the memory reclaimer) and the new linear structure in the stack is structured and linked into the proper substructure point. All combinations of replacement are allowed: structure by a structure, field by a filed, structure by a filed, field by a structure. The second situation of a field replacing a field can be a problem in the case where the new field is larger than the old field because vector expansion must take place (in the opposite situation, nulls are inserted). The simple solution of providing a non-hierarchical link out of a new space is inadequate for the situation where successive words of a large vector are sequentially expanded. The solution is to link in a new memory group only after checking if there is no space remaining in the present group or the next one, and then rewriting the remainder of the present group adjacent to the new field. In this way, expansion of many fileds of a vector makes use of the newly created space.

The general algorithm for structure referencing is for the RP to scan back through the IS stack to find the structured link, and then to proceed upward a subscript at a time, accessing each vector using special speed up techniques as appropriate, until the final subscript is reached. At this point the RP replaces the subscripted reference in the IS stack with a link to a substructure or a link to a field if the data level was reached. At any point in structure referencing, the structure previously stored may not extend to the referenced point (oversubscripting). The language rule in this situation is that new space should be created as required to expand the structure to the subscripted reference point (fields filled with nulls) and the RP is responsible for accomplishing this task.

If after structure referencing to the field level, a bound pair of subscripts appear in the IS stack, the RP scans and counts across the field, selecting the requisite characters and placing the result in the IS stack. An error is called if the bound pair is encountered before the field level is reached.

Arithmetic processor

The AP is a serial process unit operating on variable length data consisting of floating-point, normalized, decimal numbers. These operations are done from high-to-low order to simplify data handling by allowing the register operations for both string and numeric processing to be similar. Also, comparisons are faster because a mis-match is immediately known. Two other important features are included in the processing hardware: a limit register, loaded by the IS under command of the language, which causes processing to terminate at the precision specified, and a precision controlling mode whereby each operand can be specified to be accurate to its existing precision and thus control the precision of the result.

The operations add, subtract, multiply and divide are performed. For add and subtract, one or the other operand is streamed through the unit (high-to-low) until the exponents are aligned, at which time both operands start to stream through. Since the number representation is magnitude plus sign, a positive result is desired so that the signs of the operands and the sign of the operator are combined to control which, if either, of the operands is streamed through in complemented form. High-to-low order arithmetic requires a nine's counter[10] to delay output over an intervening string of nines until a carry/no carry decision is reached. Eventually, either an empirical end of an operand is reached, or the limit counter value is reached, or both exact numbers are ended. At this point, arithmetic is finished and control is turned back to the IS.

Multiply is accomplished by successive additions or subtractions followed by a shift until all of the multiplier digits are exhausted. Only after the full trapazoid of the partial product is produced is a rounding pass applied to achieve the precision requirements. The speed-up of adding one to the previous multiplier digit and subtracting from the partial product if the multiplier digit is larger than four is used. Of course, with multiply (and divide) exponents are added (subtracted) so that no shift of the fractional portions of the operands are required. Division is accomplished by a gradual non-restoring reduction of the partial dividend until the precision of the result is equal to the least precise of the two operands or the limit counter.

Since processing in this system is accomplished serially in a decimal mode with few speed-ups, the speed of processing is sharply dependent on the size of the operands. When the limit counter is set to a small value, say 5, processing can be quite fast but 99 digit divides can be extremely slow. It is therefore important that the user selects only as much precision as he really needs.

The numeric comparisons are performed by the AP as a subtract operation but terminate immediately upon a mismatch and return a zero result rather than a one. The IS has the task of combining the result returned by the AP with the desired comparison operation to generate the overall result in the IS stack.

Format processor

The FP unit performs the string JOIN operation, the binary string operations AND, OR, NOT, the string comparison operations BEFORE, SAME, AFTER, the FORMAT and MASK operations, and the automatic type conversion on operands requested by the IS: numeric to string, string to numeric, and numeric to integer (used primarily for subscripts). These operations are also performed serially.

The JOIN operation is performed in the obvious manner of streaming the second operand onto the tail of the first operand, forming a single result operand.

The binary operations are performed character-by-character, performing the required operation by producing 0/1 result characters, filling in the shorter operand with zeros.

The string comparisons are also performed character-by-character, comparing successive characters until a mismatch is found according to the built-in ASCII collate sequence and returning a 0/1 result.

The FORMAT and MASK operators provide a powerful string manipulation capability for a wide variety of applications from payroll and banking forms

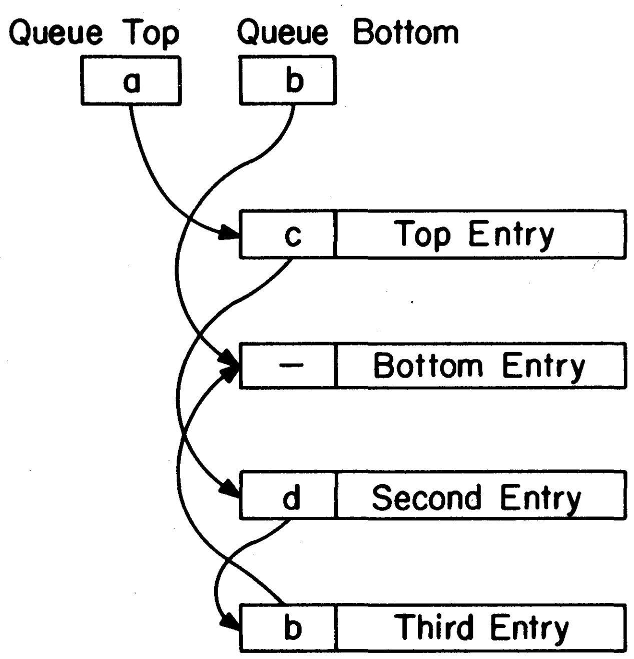


Figure 19—Typical task queue structure

preparation to system software character manipulation. FORMAT is a packed-numeric-to-string operator that allows the user to describe the format of the result with a pictorial like character string. The operation is performed in a serial manner as dictated by the operands. The standard default conversion from packed numeric form to string is a subset of the FORMAT operation. MASK is a string-to-string operator similar to FORMAT. MASK can be used for character insertion, deletion, and spacing control. It is often used to control or measure the length of the fields. MASK is also processed in a serial-by-character manner.

SYSTEM SUPERVISION

The Load, Compile, Execution, and I/O comprise the basic processing modes for the system. Three additional modes are defined for a terminal, off-line, on-line idle, and normal completion. They are all passive modes and differ only in the allowed transitions that can take place upon an interrupt stimulus. For example, the normal completion state is the only state from which the RESTART execution command can be honored. RESTART is only allowed if the object string were left in a reusable state.

The diagram in Figure 14 shows a few of the terminal state transitions. These transitions are significant in that they are all supported by hardware algorithms. When the control code corresponding to RUN is received by the SS the transition from the Load mode to the compile mode can be processed without software intervention. Many other transitions can occur but they generally require some system software assistance. The transition from the Load mode to the Compile mode involves the following steps. If the IP is active it must be allowed to complete in such a way that the source string is intact. The task is then removed from the queue for the IP and added to the queue for the TR. In addition the control tables in main memory are initialized for the TR making available the address of the start of the source string and the address of the procedure libraries to be used.

A typical task queue is illustrated in Figure 19. It is comprised of a linked list of entries (control words). The queue has a pointer to the top entry and another pointer to the bottom entry. By maintaining both the top and bottom pointers it is easy to add an entry to either the top or the bottom of the queue.

Each time a control transition occurs the SS updates the queues by performing appropriate add or delete actions to each of the processor queues involved. This is part of first phase of any SS task processing. The second phase of SS processing involves assigning work to free processors that have assignable tasks on their queues.

The multiprocessing algorithm is centered around manipulation and use of queues for the CP, TR, IP, MR, and DC. The SS has a general purpose queue processor that allows an item to be added to the top, added to the bottom, or deleted from any queue. The algorithm has a default mode which is completely hardware controlled. Various parameters can be set by software that bias the operating dynamics. For example, two time values are maintained for each entry in the CP queue. One measures the accumulated processing time and the other measures the actual time that the task is on the top of a queue. The values are preset to parameter values when a task enters the queue. When the values have been counted down to

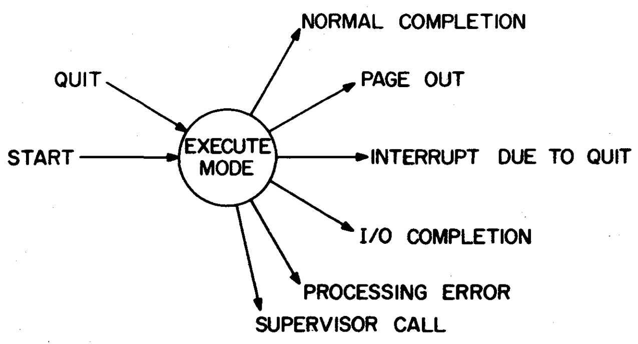


Figure 20—Mode transitions affecting the central processor

zero an SS task is generated to modify the queues. In most cases this is used to move the task from a high priority position near the top of a queue to a low priority position near the bottom of a queue.

The processing flow in Figure 14 is greatly oversimplified for general purpose system supervision. In Figure 20, the control commands to and from the central processor are illustrated. The SS can command the CP to start on a task or to quit working on a task. The CP can terminate processing on a given task for one of six basic reasons. Consider the I/O completion. In most cases for most terminals the hardware algorithm for controlling I/O would be sufficient. If on the other hand, a batch processing terminal with spooled I/O were desired it would be necessary to alter the control process for I/O with a system software procedure. To cause software to be called for a specific terminal upon an I/O service request, a specific control bit must be set in the terminal control word for that channel. This causes an automatic software call to be generated by the SS.

The software call is handled in SYMBOL by starting a pseudo terminal operating with the requesting channel number as a parameter. In this manner the control header tables for the requesting channel can be operated upon as data. This is illustrated in Figure 21 where an interrupt of a specific class causes the corresponding program specified in a software call table to be selected and control transferred to the pseudo terminal with the parameter TN. Each different class of interrupt maps into a different control word in the software control table. In this manner only the software procedure desired will be accessed in virtual memory. In SYMBOL over 80 different software interrupts are controlled via the software control table located in the lower part of main memory. This represents the principle interface between hardware and system software.

CONCLUSION

The traditional boundary between hardware and software has been weakened during the past ten years and is due for a significant shift beyond the token improvements. It is believed that in SYMBOL a major step towards significantly more capable hardware has been attained.

The SYMBOL system is now entering an extensive evaluation phase where the system's strengths and weaknesses will become more apparent through actual day to day usage. The developers of the system have gained much insight into the merits of each of the approaches taken. The overall approach to memory management is considered a breakthrough. The moving of

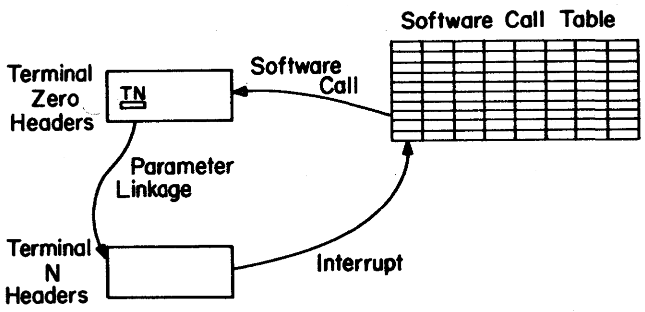


Figure 21—Mechanism for handling a software call caused by a transition interrupt

data attributes from instructions to the data is considered fundamental.

Nó clain is made that the SYMBOL system has been balanced for optimum performance and use of hardware. Certain critical areas of memory management and system supervision are felt to be 10 to 100 times more efficient than conventional means. Certain aspects of structure referencing are a major advance over software list processors but fall short of being competitive for some types of large array referencing. Many of the weaknesses in this first SYMBOL model were solved by the designers too late to be factored into the actual hardware. Many other aspects of the system such as the paging and system supervisor algorithms can be evaluated after significant usage experience.

The computing professionals have debated for many years the questions: Can a compiler be developed in hardware? Can the heart of system supervision be committed to hardware? Can data space management be taken over by hardware? Can hardware be designed to take over major software functions? Can complex hardware be debugged? These and many other questions have been positively answered with the running SYMBOL system. The most significant part of the entire project is that the concepts were reduced to full scale, operating hardware.

ACKNOWLEDGMENTS

In addition to the authors of this paper, many others contributed to some aspect of the SYMBOL project. Critical to the completion of the system was the team spirit and devotion displayed throughout the project by all members of the staff.

The authors wish to acknowledge George Powers, Stanley Mazor, and Russel Briggs for their contributions to the hardware development and Hamilton

Richards and Mrs. Hilma Mortell for their contributions to the early software development.

REFERENCES

1 R RICE W R SMITH
SYMBOL: A major departure from classic software dominated von Neumann computing systems
This volume

2 S MAZOR
Programming and/or logic design
Digest of the 1968 Computer Group Conference June 1968 Los Angeles

3 G D CHESLEY W R SMITH
The hardware-implemented high-level machine language for SYMBOL
This volume

4 B E COWART R RICE S F LUNDSTROM
The physical attributes and testing aspects of the SYMBOL system
This volume

5 W R SMITH
System design based on LSI constraints: A case history
Digest of the 1968 Computer Group Conference June 1968 Los Angeles

6 T KILBURN
One-level storage system
IRE Transactions on Electronic Computers Vol EC-11 Number 2 April 1962

7 E L GLASER J F COULEUR G A OLIVER
System design of a computer for time sharing applications
1965 FJCC Vol 27 Part 1

8 F J CORBATO V A VYSSOTSKY
Introduction and overview of the multics system
1965 FJCC Vol 27 Part 1

9 W R SMITH
Associative memory techniques for large data processors
PhD Dissertation Iowa State University 1963

10 A P MULLERY R F SCHAUER R RICE
Adam: A problem-oriented symbol processor
1963 SJCC Vol 23

A semi-automatic relevancy generation technique for data processing, education and career development

by J. DAVID BENENATI

Xerox Corporation
Rochester, New York

TECHNICAL OBSOLESCENCE: CURE AND PREVENTION

"In a rapidly changing technology such as ADP, personnel resources, in the absence of intensive training, tend to become obsolescent at the same rate as hardware resources, and a major effort is required to keep a staff current and competent." This conclusion was recorded by a 14 member panel appointed by President Nixon to recommend improvements in the Data Processing activities of the Department of Defense. It is particularly interesting to note that this statement was one of the panel's main recommendations as cited by Information Week (8/3/70), and COMPUTERWORLD (8/19/70). The significance, of course, is that training costs are finally being classified along with hardware costs and we may at last be somewhere near the threshold of putting some order to the existing chaos of Data Processing education.

Everyone has always agreed in principle to the idea of continuing education for data processing professionals. The problems connected with providing quality DP education are not in selling the idea that it is required; but, in getting people to come to an operational agreement as to what it is. Or more specifically, there has not been a system available to DP professionals and management that enabled them to identify what training had the greatest relevance for any particular individual. If a method was established whereby an individual educational plan could be developed for each person in a given organization, it would then be a fairly simple matter to summarize these plans and ultimately set organizational priorities.

However, even though everyone concurs that continuing education is a must, it is important that a more specific statement of need is made to provide clues as to who should develop what skills. For example:

a. DP Managers and Project Leaders must be trained lest they lose touch with the rapidly changing technology. A manager who developed his programming skill with the autocoder language will not necessarily appreciate the significance of modular programming in COBOL if he has not previously been involved in COBOL coding.
b. A programmer who could improve his diagnostic techniques through a greater understanding of machine instructions cannot always pull an assignment that will expose him to BAL.
c. New techniques and concepts that have a reasonably high probability of usefulness to all DP professionals on subjects such as teleprocessing and data base organization are constantly arising but one cannot take the time to study them if he is chained to the oars of the maintenance problems associated with a second generation system.

Data Processing organizations need to develop educational guidelines that can be translated into a reasonable plan for individual skills enhancement. For only in this way can the resources be mustered that are necessary to identify and implement the monumental training task that is required everywhere in the DP industry. Most data processing professionals would agree that at least 20 percent of their time should be spent in some form of self development. If you consider a 300 man DP shop with an average per capita earnings of $10,000 per year, just the time allocated for this development would cost $600,000 per year. If you consider that most companies have in the past spent something like $2,000 per man on travel, per diem and course costs, this would add another $600,000 to the pile. The implication becomes obvious that a reasonable amount of planning and control must be exerted over a $1.2 million expenditure.

In general, the situation is even worse. Industry pays for education and training that is poorly planned, ill timed and seldom uniformly agreed as even being

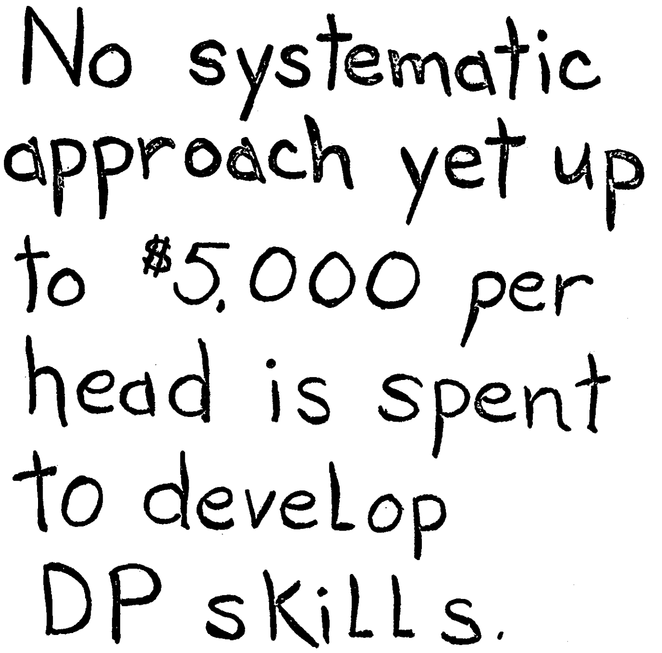

No systematic approach yet up to $5,000 per head is spent to develop DP skills.

Figure 1

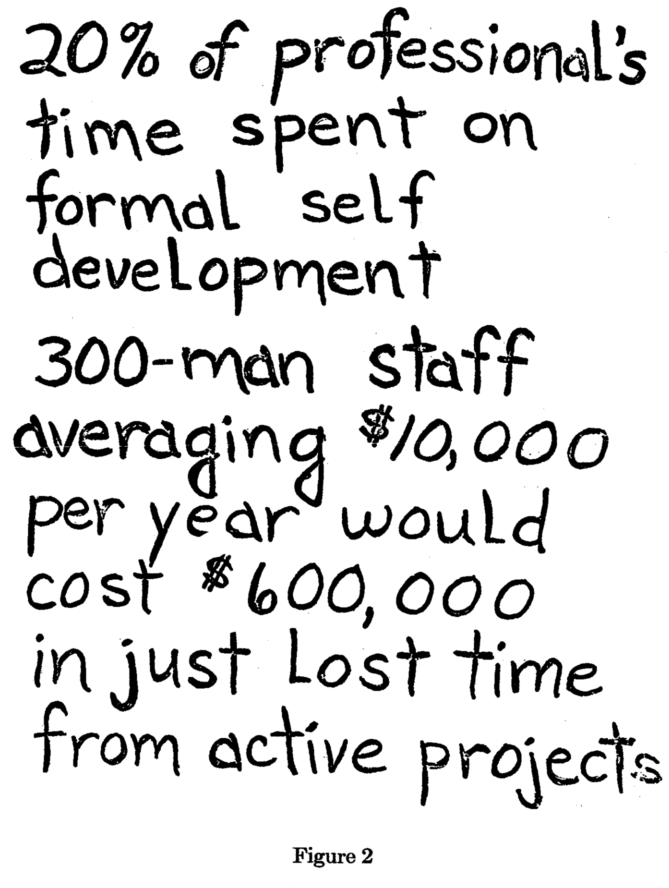
relevant to the business goals of the particular company in question; and, even more ironic, they pay for the same training twice. For if Ajax Electronics hires a programmer trainee for $8,000/year and spends another $3,000 on his training, you can typically expect that this particular programmer will be hired by Bachalah Industries for $12,500/year in two years. Thus, both Ajax Electronics and Bachalah Industries pay for the same training. Superficially, this might look great to the programmer for it appears that he really makes out—but let's face it, you cannot really get something for nothing. Sooner or later, the piper must be paid and if you really believe that your profession has substance, you should not have to resort to artificial means to inflate your salary.

How do you correct the current situation? How do you prevent it from recurring? How can management be assured that the maximum resources are directed to the most appropriate technical development of the DP staff? One approach is to systematize a great deal of the decision-making process associated with educational planning. This has been done for one Xerox Systems & Programming Group through the Data Processing Manpower Planning System. This system enables a detailed description of each staff member's personal skills history to be compared to an idealized educational model for each particular job code in order to produce a generalized training plan for the total environment. Sufficient data is now available to analyze the gross requirements of the environment and establish priorities. Once the resources have been identified and appropriated to resolve the gross requirements, a second pass can be made at individual training plans. This time, each individual's personal plan can be redeveloped in conjunction with his immediate supervisor and an education staff member. In this way, we will be in a position to realistically allocate educational resources to the outstanding requirements of the environment.

OUTLINE OF SYSTEM

It is difficult to understand the underlying rationale of man that stimulates him to account for every nickel spent in the pursuit of business goals, wherein little control whatsoever is exerted in the attainment of the human skills required to achieve those goals. At Xerox we have developed a semi-automatic mechanism for the identification, control, and attainment of the skills associated with the data processing activity.

It is fairly obvious to most data processing professionals that some sort of planning and control procedure is required to administer the resources associated with the development and maintenance of technical skills. Some of the economic relationships of data processing skills development have already been described. However, few people fully appreciate the more subtle


20% of professional's time spent on formal self development

300-man staff averaging $10,000 per year would cost $600,000 in just lost time from active projects

Figure 2

aspects of the education dilemma such as: How does a professional determine what skills will be most useful to him or his company? How does one determine how his particular skill pattern relates to the corporate goals? How does he prioritize his own educational requirements as there just is not time to learn everything?

Obviously, most data processing professionals do not have access to the kind of information that would enable them to formulate reasonable conclusions with respect to such questions. As a matter of fact, this sort of data is generally not available to anyone. The data processing manpower planning system is designed to provide the data base and the feedback mechanism to produce the information required to more intelligently address these questions. The output of the system will be available to all effected individuals such that each can contribute to the decision-making processes regarding his own professional career development.

The major elements of the system can be summarized as follows:

1. Analysis of individual work experience and educational background through the use of a Skills Inventory questionnaire.
2. Development of individualized educational plans for each member of the group.
3. Establishments of educational priorities.

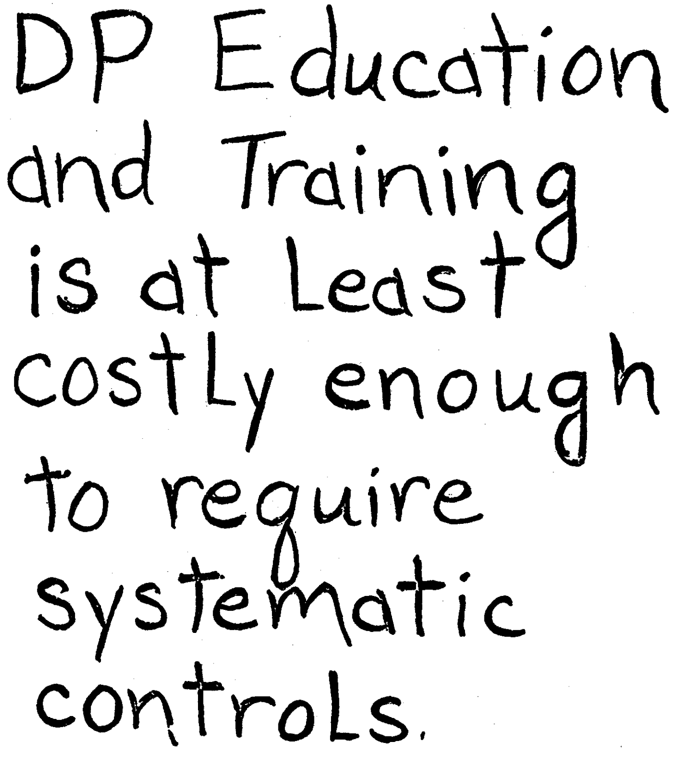

DP Education and Training is at Least costLy enough to require systematic controLs.

Figure 3

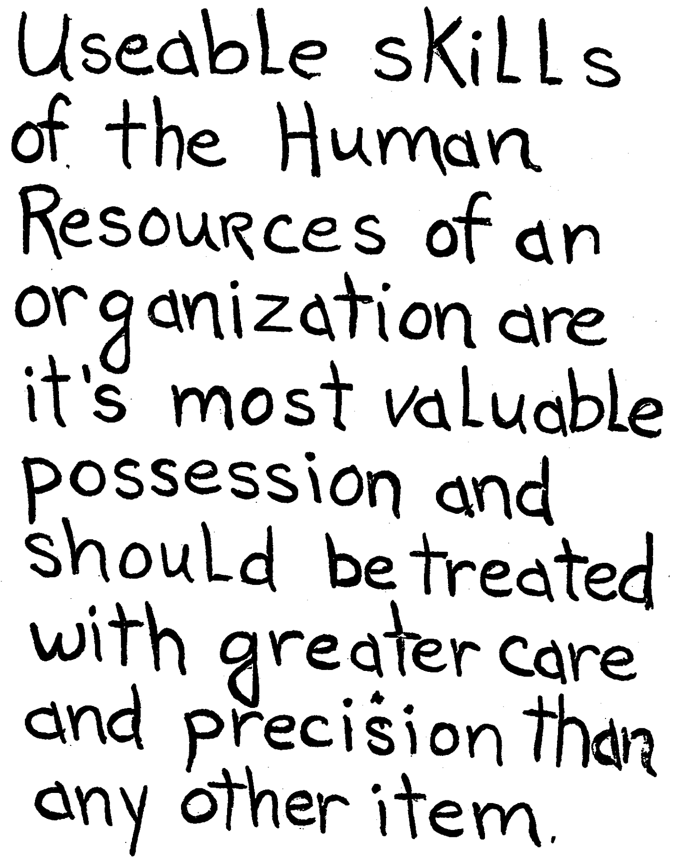

Useable sKiLLs of the Human Resources of an organization are it's most valuable possession and shouLd be treated with greater care and precision than any other item.

Figure 4

4. Determination of best implementation approach.
5. Implement required training.
6. Measure training effectiveness.
7. Reiterate entire procedure.

SKILLS INVENTORY QUESTIONNAIRE

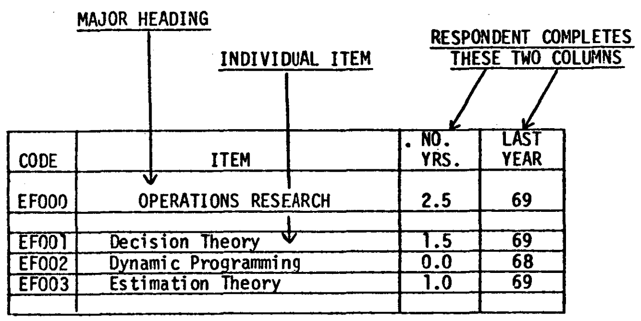
A thirty-five page questionnaire was designed to collect the work history and relevant technical education background of the members of the data processing staff. Analysis of the data contained in all of the completed questionnaires is expected to improve existing methods of:

- Educational Planning and Career Development
- Promoting and Transferring of Personnel
- Optimal allocation of Manpower to on-going and planned projects

The scope of this article will permit treatment of only the educational planning and career developmental aspects of the system.

Analysis of existing skills is an obvious point of departure in the development of any individualized manpower planning system. The questionnaire itself is divided into the following seven sections: (1) Management Experience; (2) Supplementary Personnel Data (Professional Affiliations, Memberships, Patents, Licenses, Publications, etc., and Previous Employers); (3) History of Work Experience (Applications, Data Processing Skills History, Additional Skill Fields); (4) Specialties and Preferences; (5) Hardware Experience; (6) Technical Training; and (7) Survey of Project Assignments.

For most of the items on the questionnaire, the respondent is asked to indicate the number of years associated with a particular skill and the year in which this skill was last used.


MAJOR HEADING
INDIVIDUAL ITEM
RESPONDENT COMPLETES THESE TWO COLUMNS

CODE	ITEM	NO. YRS.	LAST YEAR
EF000	OPERATIONS RESEARCH	2.5	69
EF001	Decision Theory	1.5	69
EF002	Dynamic Programming	0.0	68
EF003	Estimation Theory	1.0	69

There are over twelve hundred and fifty individual items and over one hundred individual categories. However, the form requires, on the average, only about two hours to complete because it is organized in such a manner that the respondent will skip the areas in which he personally does not possess previous training or work experience. In addition, the instrument is administered in groups of up to twenty people under highly controlled conditions with several proctors to assure that the respondents have properly interpreted the instructions. A manpower planning staff specialist describes the questionnaire and explains how to complete the form. In addition, each section is preceded by written instructions. To further reduce the communication problem, the proctors assist by providing immediate access to common definitions such as the distinction between the various levels of management and how to respond to certain areas of the questionnaire if you have never been employed as a computer operator—yet, as a programmer you acquired considerable hands-on operations experience.

When the form is completed, it is reduced to an individual profile containing only those items marked by that particular respondent. Selected key data is then summarized to formulate individual profiles to be used for analysis.

In order to assure that each individual best depicts his skill pattern in the most representative way, the profile is returned to both the individual and his immediate supervisor for comments and modification. In addition, we wish to assure that every individual knows exactly what information is retained on him in the data bank.

The inventory we have developed concentrates on strictly those skills that can be interpreted to have a direct relationship to currently envisioned projects and data processing applications. This excludes such items as chemistry, physics, etc. The decision to exclude general skills was one of expediency and it is tentatively planned that our experience might generally lead the way to the development of similar inventories for all areas of the company, not just the Rochester-based data processing organizations.

EDUCATIONAL PREREQUISITES AND EXPERIENCE EQUIVALENCIES

In order to determine what skills would most probably be needed in the Xerox-Rochester environment in the next six to eighteen months, we established a Blue Ribbon Committee consisting of our most experienced Data Processing Management and Technical Specialists. The committee made formal recommendations as to exactly what courses would be required to assure the desired minimal skills for each of our data processing related job codes. These recommendations ranged from operations personnel to functional area training and general management. The idea was to describe in explicit detail the subject matter and skills required by an individual to be considered properly qualified for any given job. Most of the jobs have various levels and corresponding pay ranges such as associate programmer, programmer and programmer/analyst. As the business programmer/analyst makes more money than the associate programmer, it can logically be argued that he should possess all of the skills and/or formal training of the next two lower positions. Thus, it frequently turns out that the senior specialist requires more training than the junior specialist because he has not had the opportunity to update his skills through the years. The committee's recommendations as they pertain to these three positions are indicated in Figures 5, 6, 7, and 8. Each of the indicated courses has an explicit detailed course description that identifies the precise skills and the proficiencies that can be expected of its graduates. Thus, the minimal skills required for each of the indicated job codes are translated into course descriptions.

The committee also meets on a periodic basis to assure that the skills and/or training activities that have been identified as significant are also the ones most likely to be required by the environment during the next planning cycle.

We are now ready to compare the mandatory education recommendations of the committee against the individual skills profiles to develop a specific educational plan for each of our people.

DEVELOPMENT OF INDIVIDUALIZED EDUCATIONAL PLANS

Now that we have a detailed an accurate record of each individual's background and a series of formal recommendations for each job code, it should be a fairly simple matter to bring the two together in a detailed educational guide for each programmer (see Figure 9). It now becomes a mechanical task to determine whether or not an associate programmer has had experience with one of the report writer type languages such as RPG, MARK IV, or MANAGE. The specific mandatory requirements of the associate programmer dictate (see Figures 5 and 6) that he either has experience in one of these three languages or that he be automatically scheduled for training in MANAGE. It is also possible to evaluate a business programmer's Systems and Procedures background by comparing the total years and number of relevant items indicated in his response to predetermined parameters in order to evaluate whether he should be recommended for the Systems Analysis II course or has already obtained equivalent experience. The general problem of course scheduling is thus reduced to determining what experience background is equivalent to what recommended training activity. It can be readily seen that if we establish rough numeric parameters for the equivalencies it becomes a simple task to automate the entire process. For example, we can establish that a total of ten years cumulative experience or four years experience in any one of the following categories can be regarded as equivalent to the Systems Analysis II course. The categories are as follows:

CODE	ITEM	NO. YRS.	LAST YEAR
EI000	TOTAL SYSTEMS AND PROCEDURES EXPERIENCE		
EI001	Card Design		
EI002	Data File Design		
EI003	Flowcharting		
EI004	Forms Design		
EI005	Func Anal Sys Model		
EI006	Hardware Evaluation		
EI007	Mgmt Exper Sys & Proc		
EI008	Mgmt Info Sys Design		
EI009	Mission Concept-Scope Determination		
EI010	Operational Analysis		
EI011	Organizational Analysis		
EI012	PERT-CPM		
EI013	Procedure Writing		
EI014	Project Control		
EI015	Records Mgmt, Reports		
EI016	Specification Writing		
EI017	Systems Analysis		
EI018	Systems and Proc, General		
EI019	Systems Planning		
EI020	Systems Presentation		
EI021	Test Procedures		
EI022	Work Measurement		
EI023	Work Sampling		
EI024	Work Simplification		
EI025	Other (Please Specify)		

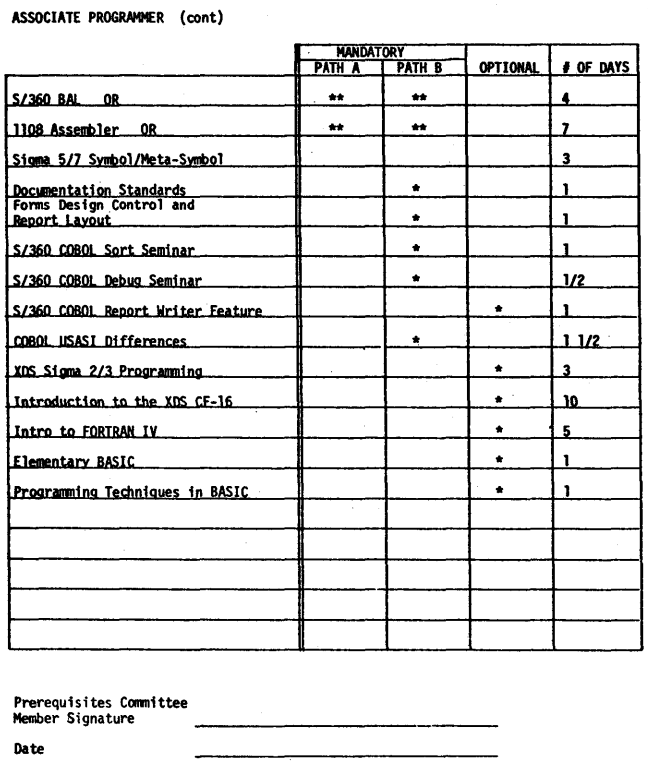
Obviously, any combination of these rules can also work such that four years indicated experience in the EI017 Systems Analysis category would be considered equivalent to the recommended training but we will not accept four years experience in EI001 Card Design

ASSOCIATE PROGRAMMER

NOTE: Associate Programmers can satisfy manditory requirements through either Path A or Path B. The principal difference being that if the Initial Data Processing Skills Course is taken, many of the other course requirements are also satisfied.
**Indicates only one course or its equivalent is required of the associated group.

	MANDITORY		OPTIONAL	# OF DAYS
	PATH A	PATH B		
Introduction to Programming Techniques			*	6
Initial Data Processing Skills Training Program	*			65
Intro to Xerox Data Processing Environment		*		1 1/2
New Employee Orientation	*	*		1/2
Xerox DP Orientation		*		2
Computer Systems Fundamentals OR		**		2 1/2
Basic Computer Concepts		**		1 1/2
Basic Data Communications	*	*		1 1/2
Intro to System 360		*		1
Fundamentals of Programming (PI)		*		2 1/2
Fundamentals of Programming Workshop		*		1
Introduction to COBOL		*		6
Introduction to JCL		*		1 1/2
IBM S/360 Utilities at Xerox	*			2
Introduction to AUTOFLOW			*	1/2
UNIVAC 1108 Intro and Control Cards			*	1
Sigma 7 Intro and Control Cards			*	2 1/2
RPG OR	**	**		1 1/2
MARK IV Advanced OR	**	**		3
Utilization of the Manage Processor	**	**		5
Systems Design and Analysis I	*	*		8

Figure 5



ASSOCIATE PROGRAMMER (cont)

	MANDATORY			
	PATH A	PATH B	OPTIONAL	# OF DAYS
S/360 BAL OR	**	**		4
1108 Assembler OR	**	**		7
Sigma 5/7 Symbol/Meta-Symbol				3
Documentation Standards		*		1
Forms Design Control and Report Layout		*		1
S/360 COBOL Sort Seminar		*		1
S/360 COBOL Debug Seminar		*		1/2
S/360 COBOL Report Writer Feature			*	1
COBOL USASI Differences		*		1 1/2
XDS Sigma 2/3 Programming			*	3
Introduction to the XDS CF-16			*	10
Intro to FORTRAN IV			*	5
Elementary BASIC			*	1
Programming Techniques in BASIC			*	1

Prerequisites Committee
Member Signature ______________________

Date ______________________

Figure 6

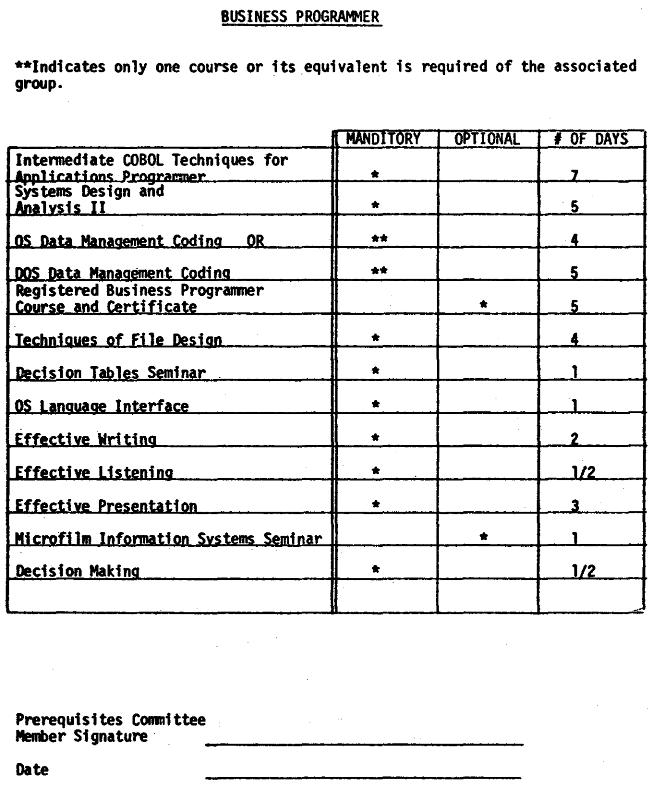
as equivalent. It is just a matter of establishing the ground rules and the basic logic associated with each recommendation.

We are now ready to review the individualized educational planning guides with the effected staff personnel. At this point, an attempt is made to smooth out anomalies caused by the ever present exceptions to the rule. A staff member of the Education & Personnel Development Group reviews the educational plans with each respondant for consistency. Common sense seems to be all that is required to extricate ourselves from embarrassing situations such as scheduling an associate programmer for a one and one-half day survey seminar on data communications because he has no previous experience in the area and has not taken formal training in the subject, when in reality, data communications is his hobby and he has a teletypewriter in his basement that is connected to the civil defense network through a modem he designed and built himself. A second confidence check of the system is made when we review each of the educational plans with the appropriate manager who finalizes the plan with an official sign-off. The manager's sign-off can be also used in extremely austere times to establish worst case priorities. The subject of general educational priorities will be treated later in this paper. As a spin-off benefit of this phase of the planning cycle, we have an opportunity to review with each area manager the overall strengths and weaknesses of his group in terms of human skills as they compare to his project assignments and also to the rest of the data processing staff. In this way, we hope to stimulate the use of the skills profiles and summary data for manpower allocation and planning purposes.

DEVELOPMENT OF GROSS BUDGET ESTIMATES

We have employed this system to collect data and develop educational plans for about three hundred people. One surprising feature of the system has been that remarkably little modifications are required after our first cut at the educational plans. That is, in reviewing our plan with the individual and his manager, less than ten percent alterations have been required from our original estimates.

All data is then summarized and we are thus in a position to identify the outstanding educational requirements of the environment. It becomes a fairly simple matter to determine what courses should be taught in house and what courses should be farmed out to an outside vendor based on an analysis of the total



BUSINESS PROGRAMMER

**Indicates only one course or its equivalent is required of the associated group.

	MANDITORY	OPTIONAL	# OF DAYS
Intermediate COBOL Techniques for Applications Programmer	*		7
Systems Design and Analysis II	*		5
OS Data Management Coding OR	**		4
DOS Data Management Coding	**		5
Registered Business Programmer Course and Certificate		*	5
Techniques of File Design	*		4
Decision Tables Seminar	*		1
OS Language Interface	*		1
Effective Writing	*		2
Effective Listening	*		1/2
Effective Presentation	*		3
Microfilm Information Systems Seminar		*	1
Decision Making	*		1/2

Prerequisites Committee
Member Signature ______________________

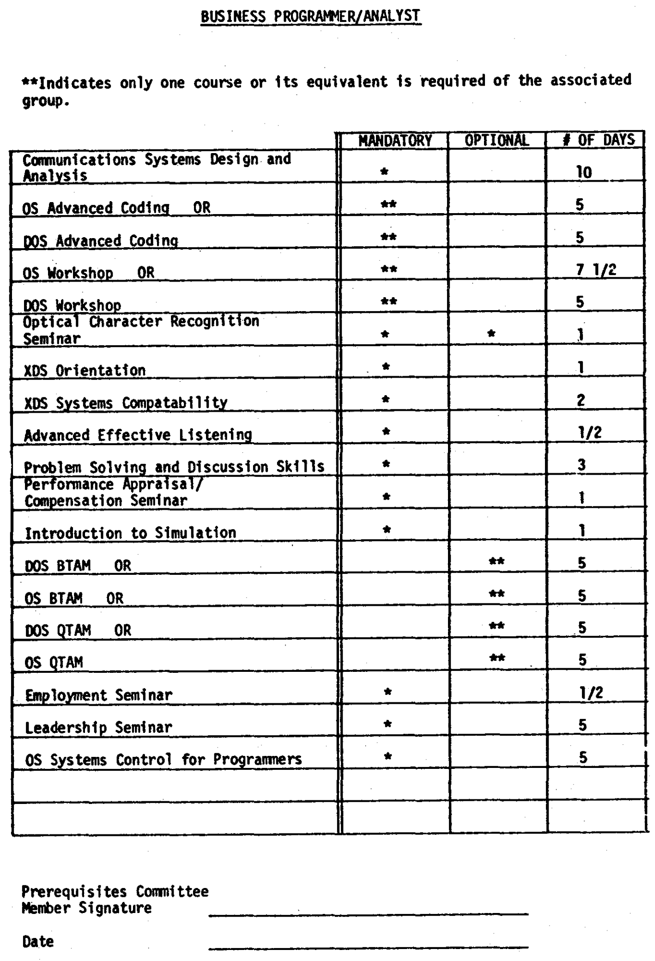
Date ______________________

Figure 7

impact on our environment.[1] In addition, now that gross data are available, the costs versus use considerations for such educational support systems as the Edutronics International films, Advanced Systems Videotapes, Centracom student carrels, etc., can be weighed against projected activity. Such factors as potential number of attendees, internal staff strengths and weaknesses, costs of comparable outside offerings, availability, etc., can now be easily orchestrated to our maximum advantage. Once the big picture is captured, the detailed considerations become almost trivial. We presently have about 100 different course offerings (or equivalent experience descriptors) that are required of various members of our data processing staff.

PRIORITIZATION OF TRAINING ACTIVITIES

The priority system developed provides management with the ability to directly control the principle educational variables in the following ways:



BUSINESS PROGRAMMER/ANALYST

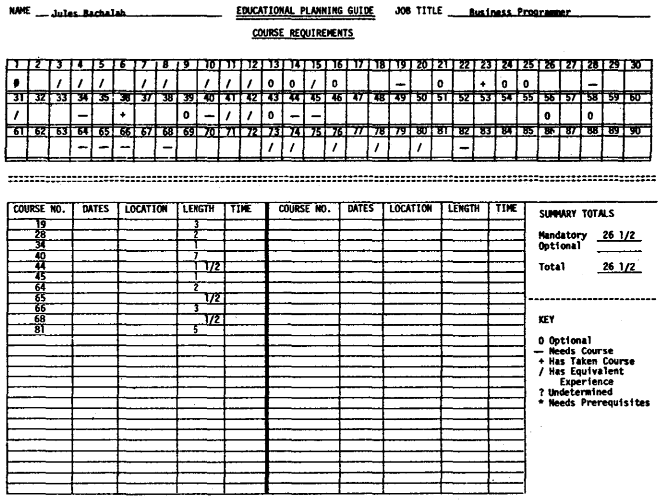
**Indicates only one course or its equivalent is required of the associated group.

	MANDATORY	OPTIONAL	# OF DAYS
Communications Systems Design and Analysis	*		10
OS Advanced Coding OR	**		5
DOS Advanced Coding	**		5
OS Workshop OR	**		7 1/2
DOS Workshop	**		5
Optical Character Recognition Seminar	*	*	1
XDS Orientation	*		1
XDS Systems Compatability	*		2
Advanced Effective Listening	*		1/2
Problem Solving and Discussion Skills	*		3
Performance Appraisal/ Compensation Seminar	*		1
Introduction to Simulation	*		1
DOS BTAM OR		**	5
OS BTAM OR		**	5
DOS QTAM OR		**	5
OS QTAM		**	5
Employment Seminar	*		1/2
Leadership Seminar	*		5
OS Systems Control for Programmers	*		5

Prerequisites Committee Member Signature ______________________

Date ______________________

Figure 8



NAME Jules Bachalah EDUCATIONAL PLANNING GUIDE JOB TITLE Business Programmer

COURSE REQUIREMENTS

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
*		/	/	/		/	/		/	/	/	0	0	/	0			—		0		+	0	0			—		
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
/			—		+			0	—	/	/	0	—	—											0		0		
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
			—	—	—		—					/	/		/		/		/		—								

COURSE NO.	DATES	LOCATION	LENGTH	TIME	COURSE NO.	DATES	LOCATION	LENGTH	TIME
19			3						
28			2						
34			1						
40			7						
44			1 1/2						
45			1						
64			2						
65			1/2						
66			3						
68			1/2						
81			5						

SUMMARY TOTALS

Mandatory 26 1/2
Optional ______

Total 26 1/2

KEY

0 Optional
— Needs Course
+ Has Taken Course
/ Has Equivalent Experience
? Undetermined
* Needs Prerequisites

Figure 9

1. Determine which of the many possible training activities are most significant for each position description in terms immediacy of need, relationship to job category, importance to the individual and relationship to project assignment.
2. Prioritize each of the training activities by position description into three levels or groupings of recommended courses.
3. Rank courses for each individual according to their specific needs as a function of their skills history, course priority and individual assignments.
4. Establish thresholds or cut-off levels at any point within the system to guarantee consistency with existing budgets or simply to determine the effects of future budget supplements or cuts.

The procedure employed is fully compatible with the existing data bank (Individual Skills Profile and Gross Educational Recommendations).

a. *Underlying Rationale*

The underlying rationale assumes that there is a basic set of technical para-technical and nontechnical skills needed by each member of the Xerox Systems & Data Processing Services Organization to properly discharge his responsibilities to the company. Furthermore, the particular set of skills required by any individual should be a function of his position description and his project assignments.

We also make the assumption that human skills can be acquired through either appropriate job experience, formal training or more ideally, a mixture of both. The procedure suggests that the surest, quickest and cheapest way to the attainment of many skills, particularly at the lower levels, is through planned formal training activities synchronized with relevant job assignments. As we have least control over job assignments we have reserved consideration of its effects until the last step in the system. In this way, we should not only have the most up-to-date information on job assignments but lack of definition or stability in this area will also have minimum debilitating effects on the system.

b. *Basic Procedures*

A series of minimal training activities have already been recommended by the Education Prerequisites committee for each professional job code. As expected, the indicated requirements far outstretched any reasonable estimates of the expected 1971-72 educational budgets. Therefore, our procedure is simply a systematic approach to narrowing down the existing recommendations of the Educational Prerequisites Committee to conform to any budget level authorized by management. Furthermore, we have designed the system to speak to the most urgent needs first regardless of the authorized budget level.

Each position description is assigned three levels of course priorities as follows:

Priority 1: The key to a training activity being classified under the number one or highest priority level is "Immediacy of Need." Does a professional with this particular job classification actually need this skill right now to properly discharge his responsibilities to Xerox? Next, is this course technical such as the COBOL Debugging Seminar, or para-technical such as Documentation Standards or Systems Analysis I. Thus, to qualify for priority 1, the training activity must be classified as (1) a technical course with an immediate need (TI), or (2) a para-technical course with an immediate need (PI). No other classifications are allowed.

Priority 2: To qualify for a priority 2 classification, the course must be either (1) a non-technical course with an immediate need (NTI), or (2) a technical course with a non-immediate need (TNI). An example of a non-technical course with an immediate need would be the New Employee Orientation Seminar offered by personnel (for employees with less than three months with Xerox). An example of a technical course that is needed by an associate programmer but not immediately would be Basic Data Communications as we currently do not have any associate programmers assigned to teleprocessing oriented projects.

Priority 3: In priority 3, we again find a common factor and that is non-immediacy of need. Specifically, (1) the para-technical training activities for which there is no immediate need (PNI), and, (2) non-technical training activities for which there is no immediate need (NTI). An example of a non-technical training activity, is the Leadership Seminar. Although it is not required by an associate programmer, the Leadership Seminar or its equivalency should be required of management above a certain level.

Each position description is assigned three levels of course priorities. Within each priority level are ranked individual courses or training activities according to order of significance to Xerox. The priority levels and individual course ranks can be evaluated for each member of the organization twice. First, as a part of the group requirements for all members of any given job category and second, upon request, on an individualized basis as a function of job assignment. In both cases, personal skills history is taken into account. To illustrate how the system would work, consider the following examples taken from the existing levels of Xerox Programmers:

Associate Programmer	Category	Rank
Priority 1		
Intro to Programming Techniques	TI	1
Initial DP Skills Training Program	TI	2
Computer Systems Fundamentals	TI	3
Basic Computer Concepts	TI	4
Fundamentals of Programming	TI	5
Fundamentals of Programming Workshop	TI	6
Xerox Data Processing Orientation	TI	7
S/360 Intro	TI	8
Intro to Sigma 5/7	TI	9
Intro to Sigma 3	TI	10
Intro to CF-16	TI	11
Intro to 1108	TI	12
Intro to COBOL	TI	13
Intro to 360 JCL	TI	14
Intro to Sigma 5/7 Control Cards	TI	15
Intro to Sigma 3 Control Cards	TI	16

Priority 1 (cont.)	Category	Rank
Intro to 1108 Control Cards	TI	17
S/360 Utilities	TI	18
Systems Analysis I	PI	19
Priority 2		
New Employee Orientation	NTI	1
Basic Data Communications	TNI	2
MARK IV	TNI	3
MANAGE	TNI	4
RPG	TNI	5
360 COBOL Sort	TNI	6
360 BAL	TNI	7
Sigma 5/7 Symbol Meta/Symbol	TNI	8
1108 Assembler	TNI	9
Intro to BASIC	TNI	10
Priority 3		
Intro to Autoflow	PNI	1
Intro to Xerox DP Environment	PNI	2
Documentation Standards	PNI	3
Forms Design Control and Report Layout	PNI	4

Business Programmer

Priority 1		
Intermediate Course for Application Programmers	TI	1
OS Language Interface	TI	2
Techniques of File Design	TI	3
Decision Tables	TI	4
Systems Analysis II	PI	5
Priority 2		
OS Data Management Coding	TNI	1
Priority 3		
Microfilm Info Systems	PNI	1
Effective Presentation	NTNI	2
Effective Listening	NTNI	3
Effective Writing	NTNI	4
Decision Making	NTNI	5

Programmer Analyst

Priority 1		
OS Systems Control for Programmers	TI	1
OS Advanced Coding	TI	2
DOS Advanced Coding	TI	3

Priority 2	Category	Rank
OS Workshop	TNI	1
Intro to Simulation	TNI	2
OS BTAM	TNI	3
OS QTAM	TNI	4
DOS Workshop	TNI	5
DOS BTAM	TNI	6
OS QTAM	TNI	7
Priority 3		
Communications Systems Design & Analysis	PNI	1
Leadership Seminar	NTNI	2

At first inspection, all of these recommended training activities look no more abbreviated than our initial gross plans. However, this is not the case for definite priorities have now been systematically established and most of these courses will fall through the cracks when played against the existing individual course recommendations. As an example, take the case of Roger Smedley. The initial gross recommendations for Roger are as follows:

Basic Data Communications
Documentation Standards
Forms Design, Control and Layout

If orders are issued (probably based on overall budget considerations) to satisfy only the immediate priority 1 requirements of the environment, Roger would now be scheduled for absolutely no training because none of these activities are included in this classification. However, if the priority 2 option were exercised, he would receive training in Basic Data Communications. Obviously, the other two courses fall into the priority 3 classification. Should Roger be assigned a project that required an immediate knowledge of teleprocessing, it would be a simple matter to personalize his requirements by getting appropriate management to indicate that he absolutely needs such training as a function of his assignment. The system would then recategorize the Basic Data Communications course, in this case, to be a technical course with an immediate need and attach a personalized annotation to its course descriptor (TI(P)). Thus, the course automatically moves to priority 1 for Roger, but not for all other Associate Programmers. Therefore when this system is played against initial gross recommendations, most of the identified training activities fall into the lower two priority classifications.

c. *Major Benefits*

The real power of the system does not begin to unfold until you consider how it can be used to directly superimpose management control on a highly detailed educational planning system. First of all, management at any time can recategorize a particular course into a priority 1 classification simply by defining that there is an immediate need for this particular training. The system can then report back to management just exactly how much this immediacy costs. This need can be expressed on a group or individual basis depending upon the situation. A second significant benefit to management would be the use of the system to establish overall levels of education support. Management would have the option of establishing thresholds of training at each level for all job categories. For example, in austere times we might classify only three courses as priority 1 courses for the associate programmer level; in better times, we might include all fifteen. At all times, management will have available current detailed and approved training plans for all members of its staff. All general or individual training plan changes can be quickly reflected in terms of expenditures against plan.

Probably the most exciting aspect of the entire system is its ability to help establish an equitable educational priority system. As anyone who has been involved in any form of education knows, the acquisition and dissemination of knowledge is an endless process. One does not have to be too familiar with data processing training to speculate that in a recession year there is a reasonable probability that a good training analyst could identify more educational needs than are currently feasible for any particular organization. If you can only command the resources to accomplish 50 percent of the job, how do you determine which half to implement? Again, the procedure is simple. If, for example, the committee has recommended a total of twenty courses or their equivalents as mandatory at the associate programmer level, an analysis of our staff requirements will indicate that we have people who exist at any point along the scale from one to twenty. As we have already summarized the gross educational requirements of all of the associate programmers and calculated their total educational costs in terms of dollars and man days of effort, we simply have to determine how much of that training we can presently afford. If we can only acquire enough resources to do 25 percent of the total job, our recommendations as to who should take what courses are fairly straightforward from this point onward. If associate programmer A has taken five mandatory courses or has their equivalent, we schedule him for the next five courses (25 percent) or the rough equivalent of his fair share of the total budget derated according to the exigencies of the times and his personalized requirements. If associate programmer B already has taken 15 courses or has equivalent experience, he will also be scheduled for five courses or his 25 percent of the action. Man A will now have completed 50 percent of the mandatory requirements of an associate programmer while man B will have completed 100 percent of the requirements. This approach has the added advantage that it does not penalize an individual because he was unable to receive proper training or acquire appropriate experience because he might have been chained to the oars of maintaining a second generation system. It also does not overlook the possibility that the individual with more relevant skills might have been in large part responsible for their acquisition.

FOLLOW UP AND CONTROL SYSTEM

From here on, all that remains to be done is the following: (1) Maintenance of records such as course enrollments, attendance, grades, etc.; (2) Analyze course critiques and trip reports and plan the categorization and implementation of exception training (vyz. educational activities that are not specifically handled by the system).

SUMMARY AND CONCLUSIONS

Although the system has not been in operation long enough to provide conclusive evidence of its success and we have not yet completed the automation of many of its elements that have been planned with mechanization in mind, we feel very confident that the system has to date provided us with the following benefits:

- A more precise method for the identification of gross educational requirements.
- A technique for stimulating the use of Skills Inventory type information for manpower allocation purposes.
- A tool for the development of training/educational plans in sufficient detail that measurement is meaningful and accurate budgeting is possible.
- A mechanism that enables the equitable prioritization of educational requirements.
- A system that provides summary data that allow the analysis of the economic trade offs of make or buy decisions based on projected usage.

But the most important aspect of the system is its potential to provide information to each and every data processing professional that can be used in guiding his personal career development decisions.

The basic concepts associated with the entire system are amazingly simple. That is, to prepare educational plans based on the difference between the existing skills of our staff members and the technical and business requirements of the environment. The only new twist is that the power of the computer is invoked to enable us to analyze the problems in massive detail in a very timely manner. It is interesting to consider the fact that the area Financial Accounting was one of the first activities to become almost universally automated wherein human skills accounting seems to be only slightly closer to automation today than it was ten years ago. The next step for us will be the automatic generation of daily instructor guides recommending curriculum emphasis for each of our in-house programs based on the actual skills history of the students assigned to that particular class.

REFERENCE

1 J D BENENATI
Building an in-house training program—The price of unbundling
Paper presented at the American Banking Association, National Automation Conference San Francisco California April 1970

An architectural framework for system analysis and evaluation*

by PETER FREEMAN

Carnegie-Mellon University
Pittsburgh, Pennsylvania

THE PROBLEM

In any situation where a large amount of information must be handled, the need for structure soon becomes evident. When one must process a quantity of information and reduce it to a small amount (for example, from a large set of evaluations decide whether to buy system A or system B) the need for hierarchical structure is especially evident because of the inability of the human mind to consider more than a very small number of pieces of information simultaneously. If one is presented with 100 facts and asked to make a single overall judgment in terms of three or four possibilities, he must have some means of aggregation.

A second aspect of the problem is the diversity of the information involved in the evaluation of a complex system. For example, evaluating a political information retrieval system might involve consideration of technical facts, projected usage patterns, the attitudes of politicians toward mechanization of previously intuitive processes, etc. What is needed is not only an aggregation scheme, but (hopefully) a small set of common factors that underlie the larger number of measurements. Apples and oranges *can* be compared if you know that each provides nutritional value.

The evaluation of an operating system is a good example (and, in fact, is the one that prompted the development of the approach presented here). Suppose you wish to make an evaluation that judges the system's performance, suitability to a market, use of hardware, and suitability for further development.

What you are given (or, rather, what can usually be obtained) are external performance measurements, sketchy overall descriptions of the system, some (usually incorrect) internal documentation, listings, and perhaps some internal performance measurements. You also know the hardware it uses, the general state of operating system technology in terms of the mechanisms used in other systems, the kinds of people using the system, and some rough (almost never quantitative) idea of the computational needs of those users. From such a polyglot of data you must develop succinct judgments and explanations that accurately characterize the system.

Examples of other complex evaluation and analysis tasks are:

—determine where a set of design goals is in relation to the state of the art;
—determine if a system meets its design goals;
—propose the logical successor to a given system;
—collect and evaluate all known data on a system.

These and similar tasks require one to build up a coherent view of the object being studied. This in turn requires the hierarchical structuring of information into a small number of categories. The framework presented here is a way of achieving this.

THE FRAMEWORK

Some day we may understand complex systems well enough to permit the definition and use of factors common to all systems (just as we now evaluate diverse electronic components in common terms of power drain, number of circuits used, operating temperature, etc.). Currently, however, the best we are able to do is to find a set of interesting dimensions along which to evaluate some small set of systems. The trouble is that these dimensions are usually chosen on the basis of readily available data and not on the basis of their

* This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research.

ability to cover a diverse range of information or provide a basis for total evaluation.

The framework presented here was developed in analogy to the problem of developing a thorough and, at the same time, an overall analysis of a building. It consists of six dimensions that encompass most of what one would want to know in such an evaluation. The dimensions are not necessarily independent and may be thought of as representing six ways of looking at an object.

Imagine that you are an architect asked to evaluate a large building. The data points that you can collect may be as diverse as those in our illustration above. There are, however, six categories which will serve to group information of interest about the building: its location, the foundation it is built upon, its structure, the functions it is meant to provide and/or that it actually provides, its finish, and its adaptability to new purposes. If you make a judgment about the building along each of these dimensions and combine them in a manner that agrees with the importance assigned to each (which may change from building to building), you can obtain a coherent evaluation.

Further, preparing the judgment on each dimension will require a number of subsidiary decisions that may be of final interest themselves (for example, how strong is the foundation, does it take account of special local soil conditions, etc.). More importantly, the categories will force you not only to sort out data into convenient equivalence classes but will permit the use of a single piece of data in several ways (for example, the observation that concrete block is used throughout is pertinent to an evaluation of the foundation, the finish and the adaptability as well as the structure).

Our framework is based on the following definitions of the six dimensions:

Location: The system's position with respect to another object, set of concepts, set of mechanisms or techniques.

Foundations: Objects, concepts, techniques, practices, etc., whose effect is felt throughout the system.

Structure: The way the system is physically or logically built and put together; the basic mechanisms that provide the functions of the system.

Functions: Services or actions provided by the system.

Finish: Polish, absence of "rough edges", smoothness, external appearance.

Adaptability: The ability to be changed to provide new functions or reside on new foundations.

These definitions are not very precise, nor do we want them to be. They are meant to indicate six major views that should be considered; their exact definitions may vary somewhat from situation to situation.

As illustration, the following definitions-by-example show the categories of information, or to put it another way, some of the questions to be asked along each of the six factor dimensions for the evaluation of an operating system.

Location:
1. with respect to competing systems;
2. with respect to hardware technology;
3. with respect to software technology;
4. with respect to design goals;
5. with respect to conceivable systems.

Foundations:
1. functional concepts used;
2. implementation concepts used;
3. basic resource sharing algorithms;
4. coding quality;
5. internal documentation quality.

Structure:
1. physical layout of code and tables;
2. subpart interconnections;
3. data flow;
4. control linkage mechanisms;
5. measurement sub-system;
6. test sub-system.

Functions:
1. functions provided;
2. evaluation of performance;
3. additional needed functions;
4. completeness of functions provided (internal consistency).

Finish:
1. user interface;
2. user documentation;
3. ease of doing simple tasks;
4. error handling;
5. crash rate;
6. crash recovery.

Adaptability:
1. growth possibilities;
2. ability to be tailored;
3. extensions possible.

USE OF THE FRAMEWORK

The purpose of presenting this framework is primarily to provide a starting place for the development of more coherent system evaluation methods. Nevertheless, at least three different uses can be made of it in its present form:

Generation of analysis questions

When evaluating a complex system, one is often hard-pressed to know what questions to ask. Simply measuring the performance of various components or analyzing their logic may be insufficient. By taking this framework of six dimensions, defining appropriate sub-dimensions, and then asking what must be known in order to evaluate the system along each such direction, one can arrive at a more complete set of measurements to be taken and analyses to be made.

Data organization

Given a large amount of data about a system, one must structure it as we pointed out above. Normally one wishes to organize it in an hierarchical fashion that groups like data or like subcomponents of diverse data together. This framework provides such an organization.

Guide to evaluation

If one is evaluating several systems for purposes of comparison or wishes to arrive at an overall judgment of a single system, this framework can provide the nucleus of an approach. After the framework is fully defined for the situation at hand an attempt can be made to assign relative weights to the different dimensions and sub-dimensions. Although one may not wish to rely totally on such a procedure (although it should be seriously considered), attempting to determine quantitatively the relative importance of different factors that enter into the final decision will prove to be of great help in arriving at a rationalized evaluation.

SUMMARY

We have presented a framework for the analysis and evaluation of complex systems that can serve both as an organizer of existing data and a generator of measurements to be made. Its primary feature is the grouping of diverse pieces of data into a small number of factors that have common and intuitive meanings. Its hierarchical nature allows one to gain an overview of a large amount of dissimilar data and to aggregate individual judgments from a wide variety of evaluations.

The ability of the approach to reduce the amount of information that must be considered at each level, its adaptability to differing evaluation problems, and its guidance to what questions to ask recommend its use. Further, its (albeit slight) quantitative character puts it a step ahead of the completely informal and unstructured methods commonly used. Use of the approach by the author in the total evaluation of a medium scale time-sharing system indicates that it does indeed provide a good evaluation framework.

Clearly, there is much work left to be done here. We must further refine and formalize system analysis and evaluation techniques. We must try to get a more quantitative understanding of the relationships between various dimensions and factors. We must evaluate in as rigorous a fashion as possible many systems.

Our intent has been to define a new framework for the analysis of complex systems. If its grouping of concepts proves to be useful in practice, fine. If not, perhaps its insufficiencies will spur others to develop better frameworks.

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

1971 SJCC STEERING COMMITTEE

SESSION CHAIRMEN, REVIEWERS, AND PANELISTS

SESSION CHAIRMEN

Artope, George R.
Bergman, Jules
Birnbaum, Irving
Brown, Kathryn, M.
Cox, J. Grady
Edwards, N. P.
Estrin, Gerald
Fain, William W.
Fogel, Lawrence, J.
Fox, Raymond G.
Gass, Saul I.
Goldstein, Gordon
Gotlieb, Calvin C.
Green, Paul
Grimm, E. H., III
Halbrecht, Herbert Z.
Hamblen, John
Hammer, Carl
Herzberg, Donald G.
Howerton, Paul W.
Israel, Fred
Jacobs, Walter W.
Johnston, Robert F.
McDonald, Bruce J.
Nagel, Roger N.
Oliver, Paul
Osborne, Thomas E.
Pasta, John R.
Rice, Rex
Rosin, Robert F.
Saunders, William B.
Simonson, Walter E.
Turoff, Murray
Vichnevetsky, R.
Wheeler, Gilmore S.
Wilcox, Richard H.
Winkler, Stanley

REVIEWERS

Anzelmo, Frank
Aron, Joel D.
Badger, George F., Jr.
Ball, N. Addison
Ballot, Michael
Basili, Victor R.
Bayles, Richard
Boehm, John G.
Bryan, G. Edward
Campi, Anthony V.
Canaday, R. H.
Casey, Jay
Cashman, Eugene K., Jr.
Chow, W. F.
Cohen, Gerald D.
Curtis, Kent
Denes, John E.
Dumey, Arnold I.
Earnest, C. P.
Enslow, Philip
Farmer, Nick A.
Fuelling, Clinton P.
Gass, Saul
Gilstrap, Lewey O.
Hamblen, John
Hammer, Carl
Hollander, Gerhard L.
Hsiao, David K.
Jacobs, Walter
Kain, Richard Y.
Kaltman, Alvin
Klir, George J.
Koller, Herbert
Landoll, James R.
Larsen, Norman
Lindsay, Robert K.
Lowe, Thomas C.
Machover, Carl
Macon, Nathaniel
Mawson, Bruce
McFarland, Clay
McDonald, Bruce
Meadows, Charles
Meers, Dale
Migliorisi, E. M.
Miles, E. P., Jr.
Mills, H. D.
Mitchell, Baker A.
Moshman, Jack
Nielsen, Norman R.
Oliver, Paul
Pasta, John R.
Peters, Bernard
Ramamoorthy, C. V.
Rosin, Robert
Ross, Dann C.
Saunders, William B.
Schneider, Arthur J.
Shirely, D. Lynn
Skelly, Patrick G.
Swift, Wayne
Tauber, Richard
Thomas, Lou
Turoff, Murray
Wallace, John B., Jr.
Weizenbaum, Joseph
Whiteman, John R.
Winkler, Stanley
Wolf, Eric W.
Zinn, Karl L.

PANELISTS

Abraham, David G.
Adams, Edward
Arnold, R. F.
Avram, Herbert M.
Barthel, Daniel
Belkin, Jack
Bitzer, Donald
Blumstein, Alfred
Bright, Herbert S.
Bromberg, Howard
Casey, Jay
Cawley, Daniel M.
Chartrand, Robert L.
Chevion, Dov
Chou, W.
Collins, Timothy
Conway, Richard W.
Criswell, James

Daunt, Jerome J.
Davis, Robert N.
Davis, Ruth N.
Dixon, Robert G., Jr.
Doede, John
Donahue, George R.
Dorn, Philip H.
Douglas, Alexander S.
Dummermuth, Ernst
Dunn, Robert
Ellis, Robert H.
Favret, Andrew
Felling, William
Flynn, Tom
Frank, H.
Glass, William
Gourley, D. E.
Gracer, Franklin
Grosch, H. R. J.
Hess, Sidney
Hickey, Albert E.
Higgins, Frank
Hoppe, Charles W.
Hoover, Charles
House, Peter
Hughes, L.
Ishizaki, Sumio
Israel, Fred
Jefferies, Robert S., Jr.
Johnson, Philip
Jones, Richard C.
Kelly, Joseph C.
Klein, Herbert
Knowlton, Kenneth
Lehner, Ralph
Lehmer, Derck H.
Liberatori, Robert
Longenecker, Al
Lower, Elmer
Ludwig, George H.
MacDonald, Robert B.
Macon, Nathaniel
Melody, W. H.
Mezei, L.
Mills, Richard C.
Mitzel, Harold E.
Nagel, Stuart, S.
Negroponte, Nicholas
Oestreicher, Hans
Oliver, G. A.
Olson, Jerry
Paller, Alan T.
Pimental, David
Plummer, Dave
Prokop, J. S.
Rappsilber, Thomas W.
Rea, D. E.
Rothenberg, David
Samoylenko, S. I.
Scannon, Richard, M.
Schiller, Jeffrey, S.
Schulz, Harold A.
Schwartz, Jacob T.
Smagorinsky, Joseph
Small, Robert
Smith, Fred
Snow, Joel
Solomon, Martin B.
Stewart, Robert M., Jr.
Strassmann, Paul A.
Svenson, R. A.
Talbot, Peggy Anne
Teicher, S.
Van Dyne, George M.
Vichnevetsky, Robert
Weeg, Gerard P.
Winograd, Shmuel
Yencha, Martin A.
Zingy, Roy J.

SJCC PRELMINARY LIST OF EXHIBITORS

ACM
Addison-Wesley Publishing Co., Inc.
Addressograph Multigraph Corporation
AFIPS Press
Airoyal Manufacturing Company
American Elsevier Publishing Company
AMP Incorporated
Ampex Corporation
Anderson Jacobson, Inc.
Applied Magnetics Corporation
AT&T
Atlantic Technology
Atron Corporation
Auerbach Info., Inc.
Auricord Div. Scovill Mfg. Company
Auto-Trol Corporation
Barnes & Noble
Beehive Medical Electronics, Inc.
The Bendix Corporation
Biomation
Boeing Computer Services, Inc.
Boole & Babbage, Inc.
Bridge Data Products
Bucode, Inc.
Bunker-Ramo Corp.
Burroughs Corporation, ECD
Caelus Memories, Inc.
California Computer Products, Inc.
Calma Company
Cambridge Memories, Inc.
Centronics Data Computer Corporation
Century Data Systems, Inc.
Certex, Inc.
Certron Corporation
Cincinnati Milacron, Inc.
Cipher Data Products
Codex Corporation
Collins Radio Company
ComData Corporation
Computek, Inc.
Computer Communications, Inc.
Computer Design Publishing Corporation
Computer Sciences Corporation
Computer Terminal Corporation
Computerworld
Congraf Corporation
Conrac Div. Conrac Corporation
Consolidated Computer Limited
Customized Data Systems, Inc.
Data 100 Corporation
Data General Corporation
Datamation
Datapac, Inc.
Data Printer Corporation
DataPro Research Corporation
Data Processing Magazine
Data Products Corporation
Data Product News
Datascan, Inc.
Decision Data Corporation
Delta Data Systems Corporation
Diablo Systems, Inc.
A. B. Dick Company
Dicom Industries, Inc.
Digi-Data Corporation
Digital Computer Controls
Digital Equipment Corporation
Digitronics Corporation
Documation, Inc.
Eastman Kodak Company
Electronic Arrays, Components Div.
Electronic Associates, Inc.
Electronic News
Electronic Processors, Inc.
ESE Limited
Facit-Odhner
Ford Industries, Inc.
Four-Phase Systems, Inc.
Fujitsu Limited
General Computer Service Inc.
General Electric, Communications Systems Div.
Genisco Technology Corporation
The Gerber Scientific Instrument Company
Gould Inc., Graphics Div.
GTE Lenkurt Inc.
Hayden Publishing Co., Inc.
Hazeltine Corporation
Hewlett-Packard
Hitchcock Publishing Company
Houston Instrument
IEEE
Incoterm Corporation
Inforex, Inc.
Information Control Corporation
International Data Corporation
I/Onex Div. of Sonex, Inc.
Iron Mountain Security Storage Corporation
ISS (Information Storage Systems, Inc.)
Kennedy Company
Kybe Corporation
Licon Div. of Illinois Tool Works, Inc.
Litton ABS OEM Products
Litton DATALOG Division
Lundy Electronics & Systems, Inc.
McGraw-Hill
Marshall Data Systems
Maverick Computer Systems, Inc.
Memorex

Memory Technology
Micro Switch, A Div. of Honeywell
Milgo Electronic Corporation
Modern Data
Mohawk Industrial Labs, Inc.
Monolithic Memories, Inc.
NCR Special Products Div.
Nemonic Data Systems, Inc.
Nortronics Company, Inc.
Nuclear Data Inc.
Numeridex Tape Systems, Inc. (In/Opac Div)
Odec Computer Systems, Inc.
On Line Computer Corporation
Optical Memory Systems
Optical Scanning Corporation
Pace Computing Corporation
Panasonic
Penril Data Communications, Inc.
Per Data, Inc.
Peripheral Equipment Corporation
Photophysics, Inc.
Plessey Memory Products Corporation
Potter Instrument Company, Inc.
Precision Instrument Company
Prentice Hall, Inc.
Princeton Electronic Products, Inc.
Quadri Corporation
Quindata, Inc.
RCA–CSD
Raymond Precision Industries, Inc.
Raytheon Company (Computer Operation)
Recortec, Inc.
Redcor Corporation
Remex
Research & Development
RFL Industries
Sangamo Electric Company
Science Accessories Corporation
Singer-General Precision, Inc.
Singer-Link Div.
Sorbus Incorporated
Spartan Books
Storage Technology Corporation
Sugarman Laboratories, Inc.
Sycor, Inc.
Sykes Datatronics Inc.
Syner-Data, Inc.
TEAC Corporation of America
Techtran Industries, Inc.
Tektronix, Inc.
Tele-Signal Corporation
Teletype Corporation
Tempo Computers, Inc.
Texas Instruments Incorporated
Timeplex, Inc.
Tracor Data Systems
Trio Labs
Trivex, Inc.
Ultronic Systems Corporation
United Telecontrol Electronics, Inc.
Van San Associates
Varian Data Machines
Versatec, Inc.
Video Systems Corporation
Visicon, Inc.
Vogue Instrument Corp., Shepard Div.
Wang Computer Products, Inc.
Western Union Data Services Co., Inc.
Western Union Telegraph Company
Westinghouse Electric
John Wiley & Sons, Inc.
Wiltek, Inc.
Xerox

AUTHOR INDEX

Alkus, M. R., 143
Allred, G. R., 163
Arora, S. R., 316, 337
Ayres, R. B., 497
Balzer, R. M., 485
Barsamian, H., 151
Beckett, W., 39
Benenati, J. D., 617
Berra, P. B., 557
Biss, K., 303
Blevins, P. R., 545
Boehm, B. W., 205
Borch, D. T., 93
Botterill, J. H., 103
Brame, J. L., 425
Bright, H. S., 387
Bush, T. C., 451
Byram, G. W., 401
Calhoun, M. A., 601
Cantrill, S. V., 239
Chalmers, L., 265
Chang, S. K., 393
Chen, T. C., 365
Chesley, G. D., 363, 601
Chien, R., 303
Cockrum, J. S., 23
Cohen, D., 77
Colhoun, B. A., 387
Conn, R. W., 1
Cook, T. G., 601
Cowart, B. E., 589
Crockett, E. D., 23
DeCegama, A., 151
DeCuir, L. E., 271
Defiore, C. R., 557
Derrenbacher, R. L., 497
Dickman, B. N., 529
Diehr, G., 231
Droulette, D. L., 467
Eckhouse, R. H., Jr., 169
Flynn, M. J., 459
Freeman, P., 629
Gallo, A., 337
Garnatz, D., 231
Garrett, R. W., 271
Gentile, R. B., 345
Gerould, L. D., 601
Gould, J. S., 477
Hall, T. W., 217
Hamada, N., 123
Heyne, G. F., 103
Hibbs, R. B.,
Hunt, E., 15
Ishizaki, S., 231
Jacobs, W. W., 357
Jain, G. P., 319
Jensen, D. D., 295
Katzan, H., Jr., 325
Kesel, P. G., 39
Kiefer, M., 407
Konishi, T., 123
Kuroda, T., 451
Laliotis, T. A., 601
LaLumiere, L. P., 401
Lessieu, E. J., 61
Ling, H., 373
Loo, J., 83
Lucas, J. R., Jr., 345
Lundstrom, S. F., 589, 601
Macon, N., 407
Mallory, F. B., 387
Manocha, T., 309
Martin, W. L., 309
McQuillan, J. M., 77
Moore, J., 179
Morenoff, E., 39
Morgan, H. L., 503
Morgan, K. G., 239
Nagel, S. S., 137
Noetzel, A. S., 415
Nunamaker, J. F., Jr., 283
O'Connell, M. L., 539
O'Donald, B. T., 83
Olds, G. V., 401
Patel, A. M., 11
Price, G. W., 131
Prosser, F., 295
Ramamoorthy, C. V., 425, 545
Raymond, S., 265
Rice, R., 575, 589, 601
Richter-Nielsen, J. A., 199
Schultz, J. R., 239
Seven, M. J., 205
Smith, D. N., 511
Smith, W. R., 563, 577, 601
Springer, C. H., 143
Stack, T. R., 113
Stahl, F., 303
Steingart, D., 179
Steuber, W., 265
Stevens, K. W., 309
Stillman, N. J., 557
Szygenda, S. A., 459

Thomas, D. E., Jr., 277
Thurber, K. J., 49
Tymes, L. R., 211
Ver Hoef, E. W., 491
Wagner, R. A., 503
Walker, S. T., 113
Walther, J. S., 379
Watson, R. A., 205
Winkler, S., 131
Winninghoff, F. J., 39
Whiteman, I. R., 83
Wolff, P. M., 39
Yasuda, I., 123
Zaks, R., 179