Z77-9096

TECHNICAL INFORMATION EXCHANGE

IBM®

February 25, 1969

PRINCIPLES OF TIME-SHARING

This paper is intended to introduce IBM marketing personnel to the fundamentals of terminal based computing. The nature of time sharing and its place in the spectrum of computing is discussed. In addition, the monograph covers basic implementation techniques and surveys topics related to time sharing systems, such as multi-processing and virtual memory system structures. An appendix includes a time sharing acronym and reference list, reading and a glossary of time sharing terms.

Mr. Milon H. Cluff
Mr. David Thompson
IBM Corporation
Building 005-2
South Road
Poughkeepsie, New York 12602

For IBM Internal Use Only

IBM Corporation, Technical Publications Dept., 112 E. Post Road, White Plains, N.Y. 10601

Z77-9096

CONTENTS

PREFACE

These notes have evolved as a result of teaching a one-week course entitled Principles of Time Sharing. This course is designed to introduce IBM Marketing people to the fundamentals of terminal oriented computing and to acquaint them with the support offered by IBM in this area. Much of the value of the course is derived from the actual hands-on terminal sessions during the class. These notes are intended to document the classroom lectures and provide background for analysis and evaluation of the terminal sessions.

It is said that in the course of a lifetime one should plant a tree, sire a child, and write a book. The authors have accomplished the first two of these life goals and now submit this monograph in partial fulfillment of the third.

I. What is "time-sharing"?

A. Introduction

1. The term "time-sharing" was introduced in 1959 in a paper delivered by a British mathematician, Christopher Strachey. Almost at the same time, Professor John McCarthy, then of MIT, produced independently an unpublished paper on the same subject. In the intervening years, much discussion, development, publicity, time, and money have been devoted to "time-sharing." The term has been applied broadly to include any system where components are used in an interleaved manner. The term has also been applied in a narrower sense to mean specific implementations of interactive data processing systems. As a result, "time-sharing" has no universally accepted definition. Today there are many and varied definitions in the data processing industry. Like much of the jargon in our marketplace, "time-sharing" has been overused and misunderstood. It is frequently equated to multiprogramming, or to remote job entry, or to "on-line" systems, or even to "real-time" systems. In IBM particularly, "time-sharing" has meant TSS - the S/360 Model 67 hardware and software system. It is important to recognize that there is no rigorous and widely accepted definition of "time-sharing." When you hear or speak of "time-sharing", remember the scornful but dangerous counsel of Humpty Dumpty who said, "When I use a word, it means just what I choose it to mean - neither more nor less."

2. Time-sharing in the literal sense of the term refers to the allocation of computer resources in a time dependent manner to the several programs simultaneously core resident. In this general context time-sharing is merely synonymous with multiprogramming. It represents an attempt to maximize utilization of the collective resources of a computing system. It was this concept of time-sharing that was presented by Christopher Strachey at UNESCO's International Conference on Information Processing in 1959. Note that in this use of the term, "time-sharing" is not concerned, at least not directly, with what an individual computer user wants to do.

3. In the late 1950's there existed a number of people with certain computing requirements - requirements that were not being met satisfactorily by the existing batch systems. For the most part these people were located in the M.I.T. - Lincoln Laboratory complex and the SAGE related activity at the RAND Corporation and System Development Corporation. Their concern was the efficiency of persons trying to use a computer facility. Their objective was to provide a man-computer relationship to enhance

man's problem-solving ability. It is from this group that a second "time-sharing" concept has evolved. It is this concept of "time-sharing" that we are concerned with in these notes.

4. "Time-sharing" then is merely one way of using a computing facility. It is "user oriented" in that it is intended to provide convenient access to data processing capability. It is a technique that accentuates the man-machine relationship in the problem-solving environment. This technique permits the user to deal "directly" with the system by means of a terminal device. The user is on-line and interacts or engages in a conversation with the computer system. Hence, conversational and interactive computing are synonymous terms with "time-sharing."

Before giving the definition of "time-sharing" to be used in this class, let's look briefly at other techniques of computer usage.

B. Comparision of techniques of computer usage

1. Methods of computer use:

a. User at the console
b. Traditional Batch Computing System
c. Remote Computing
   1) Remote Batch Processing
      a) Remote Job Entry
      b) Remote Job Output
   2) "Time-Sharing"
d. Real Time Computing

2. Characteristics of "User at the console"

a. The user is in direct contact with a system via its console.
b. An interactive situation exists with minimum turnaround. The degree of interaction depends on software.
c. Thruput is not a serious factor with the relatively low-cost desk-type computer. However, as system size and cost increase, direct access by single user at the console cannot be justified.
d. The size and characteristics of problems are limited by the following in desk size computers:
   1) Core size
   2) Programming Systems/Languages
   3) I/O Capability

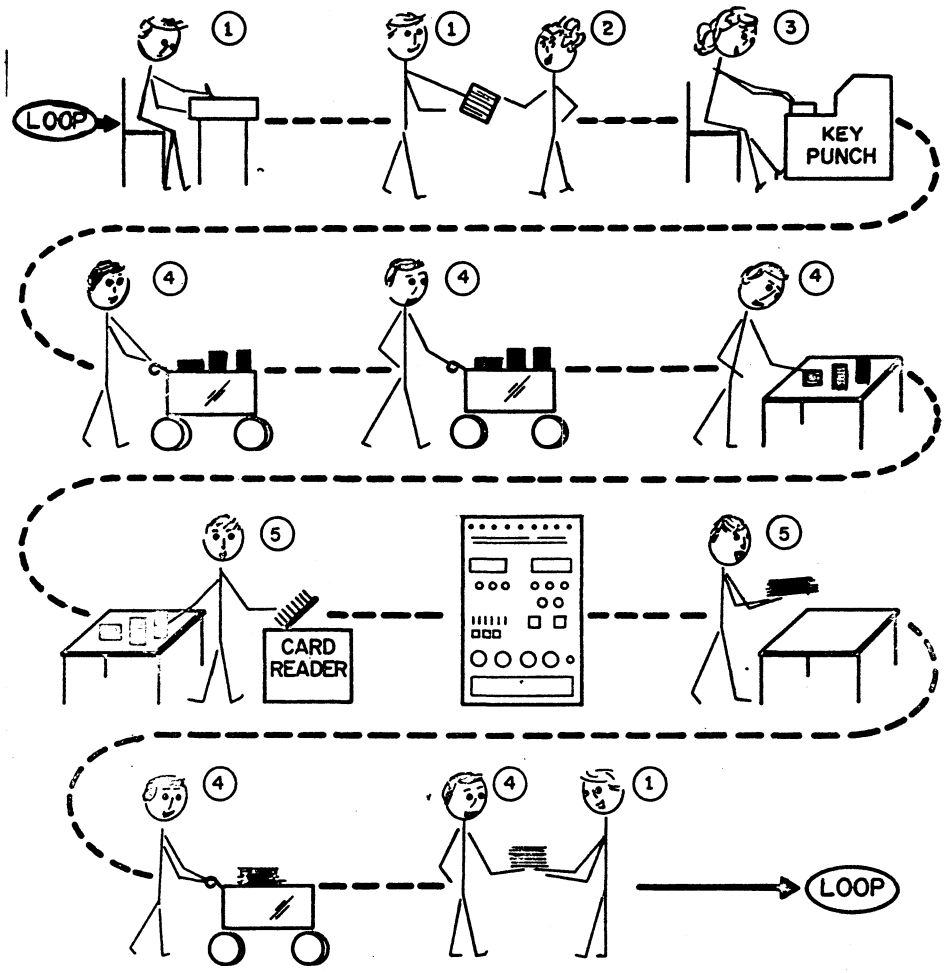
3. **Characteristics of traditional batch system:**

   a. **Thruput has been the major consideration in the development of** the local or on-site batch system.
   b. **The jobs to be run are accumulated in batches.**
   c. The system depends on the queue of jobs to keep the system **busy.**
   d. **The user is divorced from direct contact with the computer. He may even be divorced from the program where professional programmers operate in a "closed shop" environment.**
   e. **Turnaround time is a problem in the batch system.**
   f. **Directly coupled systems such as ASP were developed to reduce turnaround time.**
   g. **Simple errors cause frustrating re-runs and the delay of turnaround adds greatly to total problem solution time.**
   h. **"Local" batch system jobs are submitted physically to computer center.**

4. **Characteristics of remote batch:**

   a. **Communication facilities are used in lieu of submitting jobs physically to the computer center.**
   b. Jobs are submitted on-line into the normal batch processing mode.
   c. **Response is a function of the design of the host batch system. Job scheduling can be handled in a number of ways. With priority scheduling, remote jobs can be integrated into the local batch queue on an automatic basis. On the other hand, a separate queue may be used to accumulate remote jobs. The operator would have to intervene to switch to this alternate input job stream.**
   d. **Remote batch processing does not necessarily mean that results will be returned to the input station. Remote job entry (RJE) does not imply remote job output (RJO).**
   e. Keydriven devices such as Teletype, 1050, and 2740/41 lend themselves to low speed remote job entry, but are too slow for output listings and dumps, etc.
   f. **Higher speed devices such as the 2780 Data Transmission Terminal provide both job input and output to the remote station over communication facilities.**
   g. So-called "intelligent" terminals, i. e. the 1130, Model 20, and other models of System/360 have the speed, storage, and logic of stored programming to enhance the input/output station capability. Pre-editing, formatting, and data compaction are functions possible with "intelligent" terminal stations.

BATCH



h. Remote batch improves the access for remote users and reduces turn-around time. The amount of improvement is a function of the local batch performance and the scheduling of remote jobs into the batch.

5. Characteristics of "time-sharing:"

   a. "Time-sharing" is a system in which multiple users concurrently engage in a series of interactions with a system via terminal devices in order to develop a program, solve a problem, or get information from the system.
   b. "Series of interactions" is key. "Time-sharing" is interactive - also called conversational. Users engage in a direct and continuous dialogue - input, "reject" response, corrected input, "accepted" response, etc.
   c. Multiple concurrent users are necessary to economically justify a "time-sharing" system.
   d. Each user appears to have the full resources of the system at his disposal. Rapid multi-plexing from user to user provides this illusion.
   e. Although "time-sharing" is classified as remote computing, a user of a "time-sharing" system is not necessarily located very far from the host computer. User terminals could be located in the same room with the computer. On the other hand, the user may be located many miles away. Distance is not a criteria for "time-sharing". Man-machine direct interaction is key!

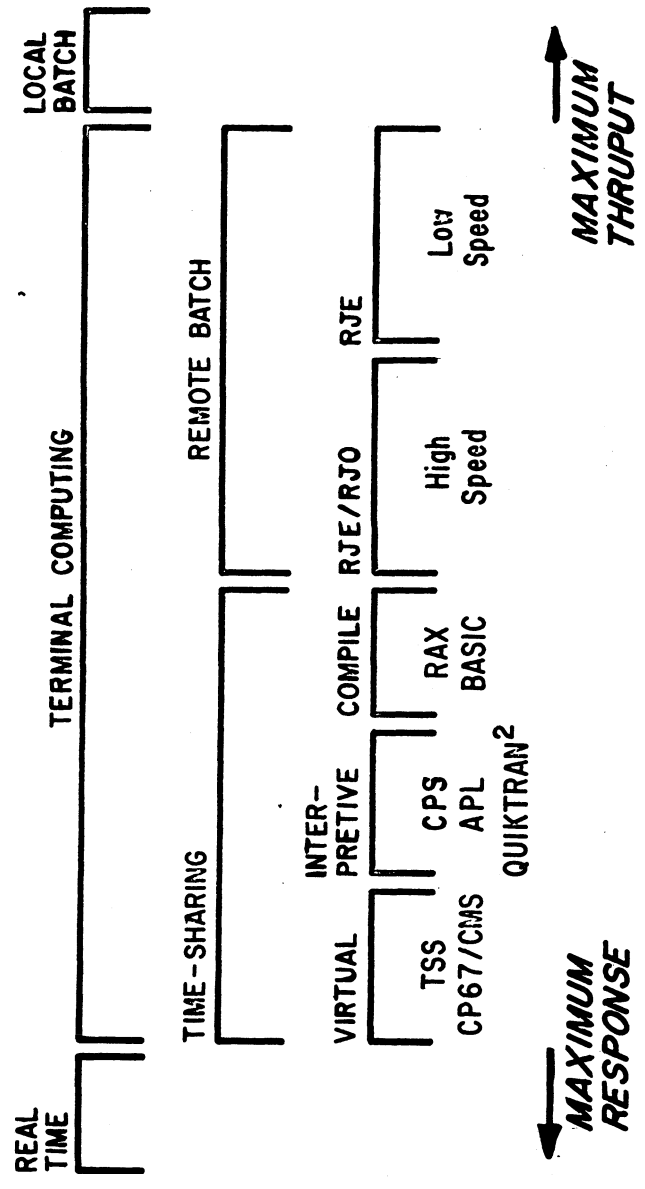
6. Characteristics of real-time computing:

   a. A real-time system is one that controls an environment by receiving data, processing it and taking action or returning results fast enough to affect the functioning of the environment at that time.
   b. Some definitions of real-time would put a limiting time response as a criteria.
   c. Time response is critical; the system cannot degrade or defer. There is no second chance.
   d. Since strict time dependency is the key, systems are often dedicated, special purpose. Often one cannot afford the overhead of a general purpose programming system in meeting real time requirements.

7. Spectrum of computing:

   a. The great bulk of computing is done today in the local batch mode. Although terminal oriented systems will grow significantly in the years ahead, batch processing will be with us for a long time.

SPECTRUM OF COMPUTING



b. The first steps in the development of programming systems was to automate flow of work through the computer and make it effeciently process work in the batch mode.
c. Emphasis has been on keeping the CPU busy and maximize thruput.
d. At the other end of the spectrum, response is the critical consideration in real time applications.
e. In between these extremes, terminal computing provides improved service to the user.
f. Terminal computing can be classified as remote batch or "time-sharing" where "time-sharing" means interactive or conversational computing.
g. The various techniques are highly inter-related.
h. It is possible to have the entire spectrum of techniques employed concurrently in the same hardware software system. Multiprogramming makes this possible.

C. Some Myths Exploded

1. Multi-programming and its relation to "time-sharing:"

a. Multi-programming takes place when two or more independent programs reside in main storage at the same time and operate in an inter-leaved manner. The programs time share the CPU sequentially.
b. Local batch systems such as DOS, OS-MFT, and OS-MVT are multi-programming systems.
c. In multi-programming, the highest priority program or task demands the CPU and will always get it when it is ready. Lower priority programs will only get the CPU resource when no higher priority programs are ready to use the CPU, i.e. are waiting for the completion of some event such as I/O activity.
d. In "time-sharing," the CPU time is divided among the user programs on a scheduled basis. All ready users will get a "slice" of the CPU time based upon some scheduling procedure. This scheduling procedure, or scheduling algorithm, determines which user gets the CPU next and for what length of time.
e. The attached chart compares multi-programming and "time-sharing" characteristics.

2. "Time-sharing" is not limited to the "computing" applications:

a. The "time-sharing" technique applies equally well to the applications that are data oriented.
b. The Spectrum of Applications diagram, shows on-line or remote processing as well as the remote "computing" requirements.
c. Computer Assisted Instruction (CAI) and Text Processing (ATS and DATATEXT) are interactive applications that are non-computing type.
d. Most applications of the on-line file maintenance/inquiry type are non-conversational. Inputs are strictly formatted and functions possible from the terminal are limited.

MULTIPROGRAMING

Shares CPU time on demand basis.

Prime Objective: keep computer busy.

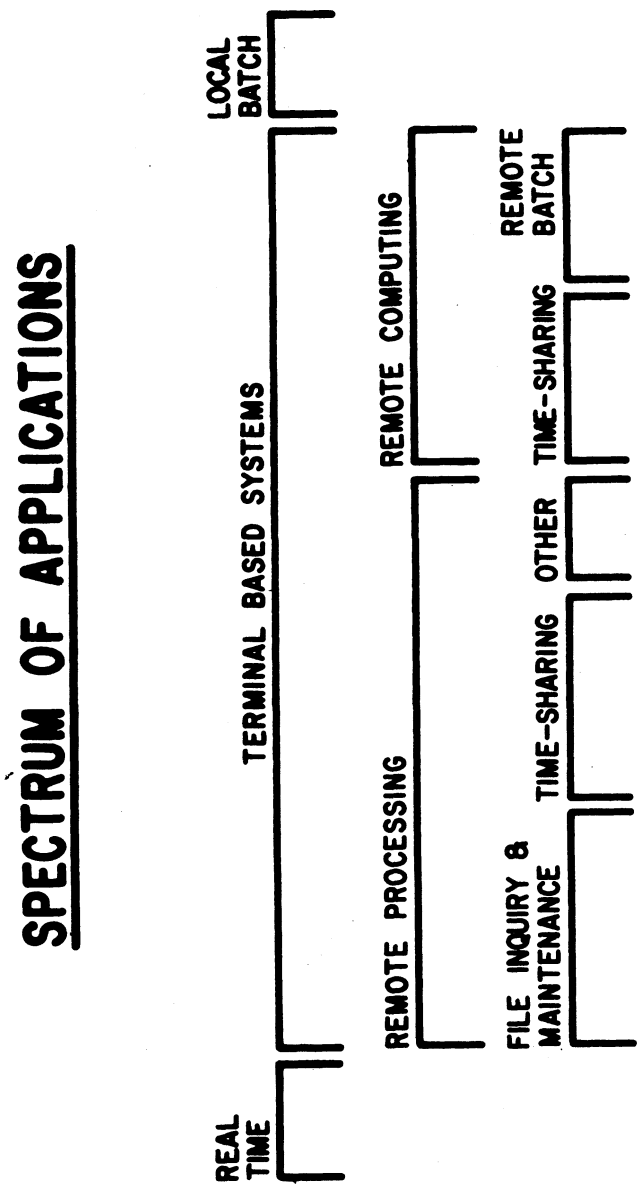
Little man/machine interaction.

"Slow" Response Time

Response implies completion of computing requirements.

TIME SHARING

- Shares CPU time on scheduled basis.
- Prime Objective: keep user busy.
- Basic design philosophy emphasizes man/machine interaction.
- "Rapid" Response Time
- Initial and successive responses imply partial completion of computing requirements.



e. More flexibility and capability from the terminal will be necessary to provide access to the on-line integrated information systems required in the years ahead. The time-sharing mode of operation can meet this requirement of Management Information Systems.
f. "Time-sharing" is much more than just providing a better slide rule for the engineer.

3. **"Time-sharing" is not a meaningful term:**

a. There is much more to time-sharing than just the sharing of CPU time. Time-sharing involves sharing resources of the entire hardware-software system.
b. Space-sharing is required in main core storage and in direct access auxiliary storage. Storage allocation is an important consideration in a time-sharing system.
c. Program sharing is a capability that makes time-sharing particularly valuable and operationally efficient.
d. A program can be shared "externally" from a program library when the "owner" makes the program available to other users. Each user granted access can have his own copy to use or modify.
e. System programs, compilers for instance, can be shared "internally" when the code is designed as serially re-usuable or re-entrant. Each user executes the same "public" copy of read-only code.
f. Data sharing is very much like program sharing, and is a facility of great value in a time-sharing system. Security controls, as in program sharing, permit limited or unlimited access to read or change datasets.

4. "Time-sharing" is not a specific system:

a. The IBM System/360 Model 67 Time Sharing System is only one specific "time-sharing" system.
b. RAX, CPS, APL, CALL/360: BASIC, QUIKTRAN, CP67/CMS, ATS, and TSS are all IBM "time-sharing" systems.
c. No one specific system will satisfy the needs of all our customers.
d. "Time-sharing" can be provided by a customer "in-house" system or he can buy service from a vendor of "time-sharing."
e. Purchasing of "time-sharing" service provides capability with a minimum investment by the user.

5. Classification of "time-sharing" systems

a. Time-sharing systems can be categorized by several attributes:
   1) Function provided to the user
   2) Implementation technique
   3) Environment

b. General purpose as opposed to special purpose or limited function systems.
   1) The definition of "time-sharing" given earlier listed the following user functions:
      a) Solve a problem
      b) Create a program
      c) Get information from a system
   2) A general purpose system would permit the terminal user to perform any of the above functions.
   3) A special purpose system might be limited to one of the above functions. Airline Reservations, for for example, can be considered a special purpose "time-sharing" system because the terminal user is limited to data entry or retrieval.
   4) Generality in a "time-sharing" system is determined to some extent by the number of programming languages available, size of programs and file space available. Certainly a system that provides only one language is limited.
   5) Note that some people consider "time-sharing" as general purpose if batch processing takes place concurrently.

c. Classification by implementation techniques
   1) A "time-sharing" systems may use a traditional batch compiler that produces object code from source language entered at the terminal. RAX and Call/360: BASIC are examples of such systems.
   2) Interpretive systems or incremental compilers provide another type of implementation for "time-sharing" systems. QUIKTRAN, CPS, and APL are examples of interpretive systems.
   3) Virtual memory systems employ special hardware for dynamic address translation and paging. TSS and CP67 may be classified as such. Note that 1) and 2) are concerned with a compiler technique while 3) is a method of memory management. The compilers used in virtual memory systems are essentially batch compiler types but there is no reason why an interpretive system could not be incorporated into a virtual memory system.

d. Classification by environment
   1) A "time-sharing" system can be implemented as a subsystem of an operating system such as OS or DOS.
   2) Conversely, a "time-sharing" system can be a stand-alone variety. A stand-alone system is dedicated to the time-sharing function. As a result, it has no background or batch capability. On the other hand, the stand-alone version will generally require less core, be easier to install and maintain, and perform more reliably and faster than its counterpart imbedded in an operating system.

6. Characteristics of the Ideal Time-Sharing System:

a. From the standpoint of the user, the ideal system should have unlimited function with instantaneous response.
b. Program development languages should fit the range of problems to be solved: FORTRAN, COBOL, PL/I, ASSEMBLER.
c. Application oriented languages for problem solving for non-computer professionals: i. e. COGO, STRESS, statistical packages, Simulators, and list processing languages.
d. Full data management capabilities are necessary to satisfy file maintenance and information retrieval requirements. Program libraries are included.
e. No limit is placed on program size and number of users.
f. Debugging capability.
g. Simplicity of operation.
h. Optional help or prompting information when required.
i. Selection of terminal support - including keyboard, card, paper tape, CRT display.
j. Language capability compatible with batch systems.
k. One must recognize that there is a trade-off in terms of function and response. Clearly, as all the functions of the ideal system are implemented, it becomes increasingly difficult to maintain ideal performance. More facility means more complex relationships and more over head. As the capability offered at the terminal grows, the language necessary to invoke the capability, becomes more comprehensive. Simplicity of operation gives way as capability grows.
l. As of this time, no one has an "ideal time-sharing" system as described above. All of the existing systems have limited capability and features. The "ideal" may never be achieved but sound, general-purpose time-sharing systems are not far off as we approach the 1970s.

II. Why is "time-sharing"?

A. What motivates the growing desire to use computers interactively?

1. The frustration with delays involved in the batch processing mode have generated a desire for a more responsive and accessible mode of operation.

2. The shortage of trained and talented people make it imperative that the available resources be used to the best advantage.

3. The growth and diversification of computer usage demands a man-computer relationship that enhances man's problem solving ability.

B. Time-sharing offers the opportunity to improve performance in what is currently being done with computers.

1. Interaction in program development has proven very effective.

2. Elapsed time for program development has been reduced and consequently lowered the total cost of development.

3. Better utilization of a limited resource - qualified people.

4. A time-sharing approach permits a user to concentrate on a single job without fill-ins or additional programming tasks because of "wait" periods. Immediate turnaround eliminates need for secondary tasks.

5. Management of programming development is simplified where "time-sharing" permits continuous, dedicated effort to a single problem solution.

6. There are an estimated 100,000 trained programmers active at the present time. Another 50,000 to 75,000 are probably needed. Every programmer is a potential "time-sharing" user for program creation.

7. Interactive development, debugging, and execution can reduce some of the erroneous output and unnecessary listings and voluminous dumps that frequent the batch processing mode.

8. If the user can interact, as the solution unfolds, unproductive runs can be eliminated, exception coding can be reduced simplifying program logic, and the printing of repetitive listings, dumps, and results can be avoided.

C. "Time-sharing" extends the range of computer usage to the problems "too small" for traditional computer methods.

1. Small jobs presently done by slide rule, desk calculator, "seat of the pants," or not done at all because of impracticality in batch mode are made practical in "time-sharing mode."

2. The experience of the Ford Motor Company with 16,000 small jobs, cited in EDP Analyzer, showed that the median user was on the terminal for 10 minutes.

3. In the university environment, student problem solving can be best handled in the time-sharing mode. There are nearly three million students in the schools and universities of the United States.

D. "Time-sharing"extends the range of computer usage to the problems "too complex" for traditional computer methods.

1. There is a whole span of problems that probably cannot be solved except by a highly interactive man-machine environment.

2. In problem solving, man excels in setting goals, in laying down guide lines, choosing approaches, following intuition, exercising judgement, and evaluating results. These aspects are heuristic, meaning that they lead toward or facilitate invention or discovery.

3. The great value of computers lie in their ability to execute very rapidly and very accurately, procedures that have been defined explicitly and in detail. These procedures are called algorithms.

4. In batch mode processing, the heuristic contributions are supplied by the user before the program gets into the computer. The heuristic contribution then ceases abruptly, and the execution of the algorithm begins. This separation of the two aspects is a serious handicap to the solution of the problems that face our customers in the frontiers of computing.

E. "Time-sharing" extends the range of computer usage by making the computer more accessible to other professionals.

1. In addition to the 800,000 to 1,000,000 scientists and engineers at work in the United States, other non-programmer professionals are potential users of "time-sharing" systems.

2. Financial institutions are particularly active in using and providing "time-sharing" service to their customers. Investment analysis is the salient application area.

3. Statisticians, actuaries, market-forecasters, and analysts of every kind have need of the immediate problem solving made possible by "time-sharing."

4. Doctors, lawyers, accountants, brokers, credit managers, real estate people, and legislators have need of "data bank" access that "time-sharing" can provide.

5. Authors, educators, and technical writers have found text processing at the terminal to be an excellent tool to facilitate text-book and manual preparation.

6. Application oriented languages are vital in extending computer usage to non-computer professionals. Conversational programs like COGO, STRESS, GPSS, ECAP, PMS, STATPAK, MPS (LP), AND GIS will pave the way for other professionals to avail themselves of computing or processing utility.

7. Professional users who are not programmers and have no intention of becoming such are often motivated or stimulated by "time-sharing" to incorporate additions or modifications to packages or even write processors of their own.

F. Expertise with attendant publicity, salesmanship, and financial support have contributed significantly to "time-sharing" evolvement.

1. Acknowledged centers of computer technology have been leaders in the development of "time-sharing" systems. Our leading universities and the so-called "think tanks" have contributed the expertise in the early and continuing evolvement of "time-sharing."

2. Many respected and articulate proponents of "time-sharing" have voiced their convictions fervently and repeatedly. Computer conventions, and journals have provided a continuing forum for papers and presentations.

3. The United States government through ARPA of Department of Defense (DOD), National Science Foundation (NSF), Atomic Energy Commission (AEC), and National Institutes of Health (NIH), has contributed millions of dollars toward development costs at the universities and "think tanks." ARPA spending reported at $12 million to $13 million a year on "time-sharing" by FORTRAN article, August, 1967.

4. IBM has contributed significantly to the "time-sharing" development with internal experimental systems, i.e. TSM, M44, CP40, and APL. Many customer projects in "time-sharing" have been jointly developed with IBM. IBM programs in addition to TSS, include RAX, CPS, ATS, QUIKTRAN, CALL/360: BASIC, CALL/360: DATATEXT, and CP67.

5. First generation systems have proven the concept to be workable. Satisfied users continue to promote and extol the virtues of "time-sharing." Demonstration is crucial in selling interactive computing.

6. Although there are many statements about improved productivity and efficiency of users, there are little or no statistics to cost justify "time-sharing" facilities.

III. How is "time-sharing" accomplished? Implementation techniques.

A. Consideration of user environment

1. "Time-sharing" is defined as interactive or conversational computing. The degree of interaction is highly variable. Depending on design and implementation techniques, it is possible to develop a wide range of conversational capability.

2. The degree of interaction can vary with the function being performed. During program creation one level of interaction may be possible, whereas program execution may have a lesser degree of conversationality. Again, the type and sophistication of the system implementation will determine the amount of user interaction.

3. A very high degree of interaction may be achieved, but only at the expense of some other desirable characteristic, i.e., response or execution speed. As in any system, trade-offs between facility and performance must be considered.

4. The user must have the capability to specify the "time-sharing" system function to be used - whether program creation, modification, execution, etc. A "Command Language" is necessary to permit the user to direct his activity.

5. Commands may be explicitly stated by key words or verbs in the terminal user's language to select the mode of operation.

6. Some commands are administrative in nature. Other commands allow the user to load, list, create, modify, execute a program, list the program names in the library, interrupt a program, and save a program in the library.

7. The concept of "time-sharing" works or is economically feasible because at any instant of time, a large number of the terminal users are making little or no demand on the system. So-called "think" time and user typing time make no demand or very little demand on the system. Output printing, likewise, requires little system effort.

8. Users in "hard execution," compilation for instance, do make heavy demands on the system. Few of the total population of users are in "hard execution" at the same time however.

B. Components of the "time-sharing" system.

1. Like the traditional batch system, the heart of a "time-sharing" system is the supervisor. The normal functions of interrupt handling, physical IOCS, communication region, etc., are included in the "time-sharing" supervisor.

2. Additional capability required in the supervisor for "time-sharing" is as follows:

   a) Terminal Access Method
   b) Command Language Interpreter
   c) Storage management
   d) Scheduler

3. On-line libraries, providing "private" and "public" storage for programs are a must in "time-sharing" systems. Data storage is also required on-line.

4. Language processors and application oriented programs are a necessity.

5. Utility routines are generally included for house-keeping functions in the "time-sharing" system. Source program update, library purge, and accounting information processing routines are typical utility routines required.

C. Batch Compiler Implementation.

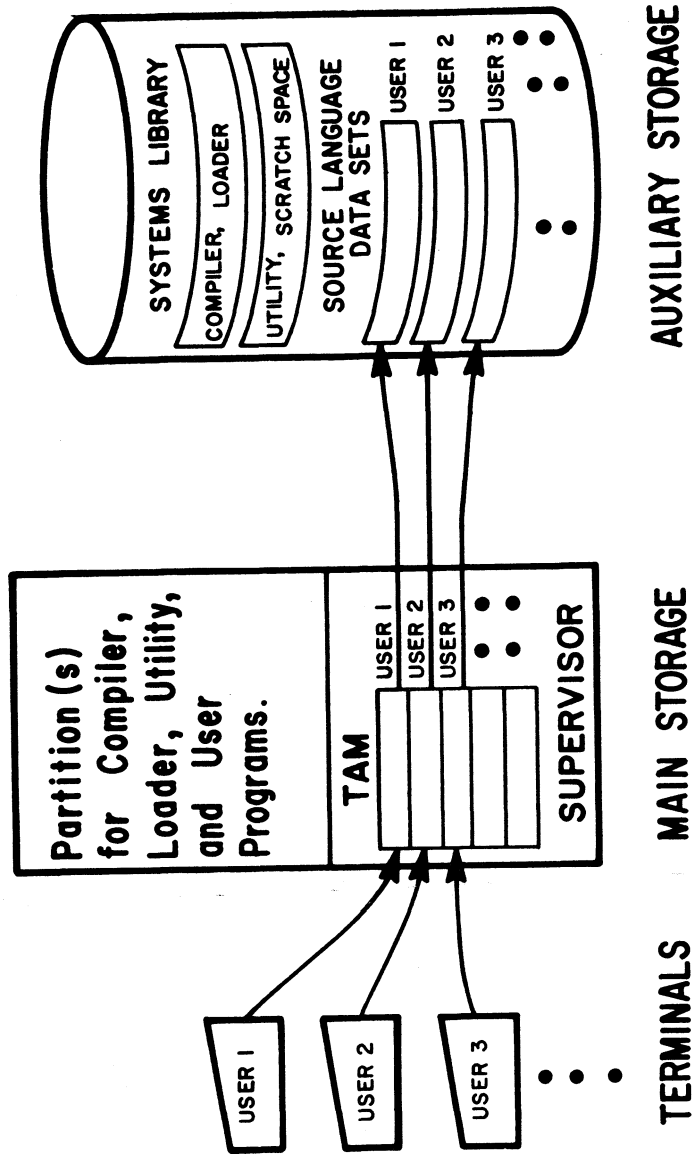
1. One technique of developing a "time-sharing" system is to use components originally developed for a batch compiler system. A compiler, loader, basic supervisor, and library facility provide the framework and are modified and extended for "time-sharing."

2. BPS FORTRAN provided the basis for RAX. Many changes were required. Not the least of which was a change of residence from tape to disk.

3. The user creates a source language data set from the terminal. Each character of an input line is transmitted to a terminal buffer in core storage. At the end of the input line, the terminal buffer is transferred to the input source data set on auxiliary storage. Eventually, the complete source data set is processed by the batch compiler when the appropriate command is entered.

4. Diagnostic messages are returned to the terminal so that input source language can be corrected - assuming errors were detected. User then enters update mode to make changes to the source program. When the user finishes modifications, the changes are merged into the source input data set. The compiler is again invoked and the process is repeated.

BATCH-COMPILER IMPLEMENTATION OF TIME-SHARING



5. When a compiler run is without serious errors, the program is loaded and executed. Execution is time-sliced. Each user who is ready to execute will get an increment of time. The time-slice can be a fixed length of time set by a sysgen parameter.

6. Note that interaction is relatively limited in the batch compiler implementation of "time-sharing." As each line is entered, there is generally no response from the system. The user is informed of detected errors only at the completion of the compilation.

7. A pre-processor can be added to perform certain analysis as each line is entered. This analysis is called line-by-line syntax checking. It basically examines the ordering or format of the input line. A good example of something that can be checked on an intra-line basis is a balanced number of parentheses.

8. A line-by-line syntax check duplicates some compiler function. Global checking, that is the inter-statement consistency of the program, requires the compiler.

9. Adequate response from the batch-compiler time-sharing system depends on the compiler performance. Speed, efficiency and minimum overhead are obviously required. The need for line-by-line syntax is negated if the compiler performance is high.

10. Program modification permits program changes without re-entering the entire program. Lines in error are generally completely re-entered in the batch-compiler system. More sophisticated editors permit "context editing" whereby only the item in error in the line is changed.

11. During program execution, conversational READ and WRITE from and to the terminal allow the user to interact. The running program can solicit input from the terminal with prompting messages.

12. Conversational READ and WRITE are basic requirements in order for the user to interact with a program during execution. Additional capability for user interaction can be added with a de-bug language for core "snap-shots'

13. The batch compiler produces object code. Program execution is direct without loss of speed. Given an assignment statement:

Y = A + B

The possible symbolic code is as follows:

```
LD     8, A
AD     8, B
STD    8, Y
```

14. The CALL/360: BASIC system is a compile-type time-sharing system. However, the compiler was specifically designed for the time-sharing environment. It may or may not reside "in-core" depending on the instantaneous load on the system.

D. Interpretive Time-Sharing Implementation:

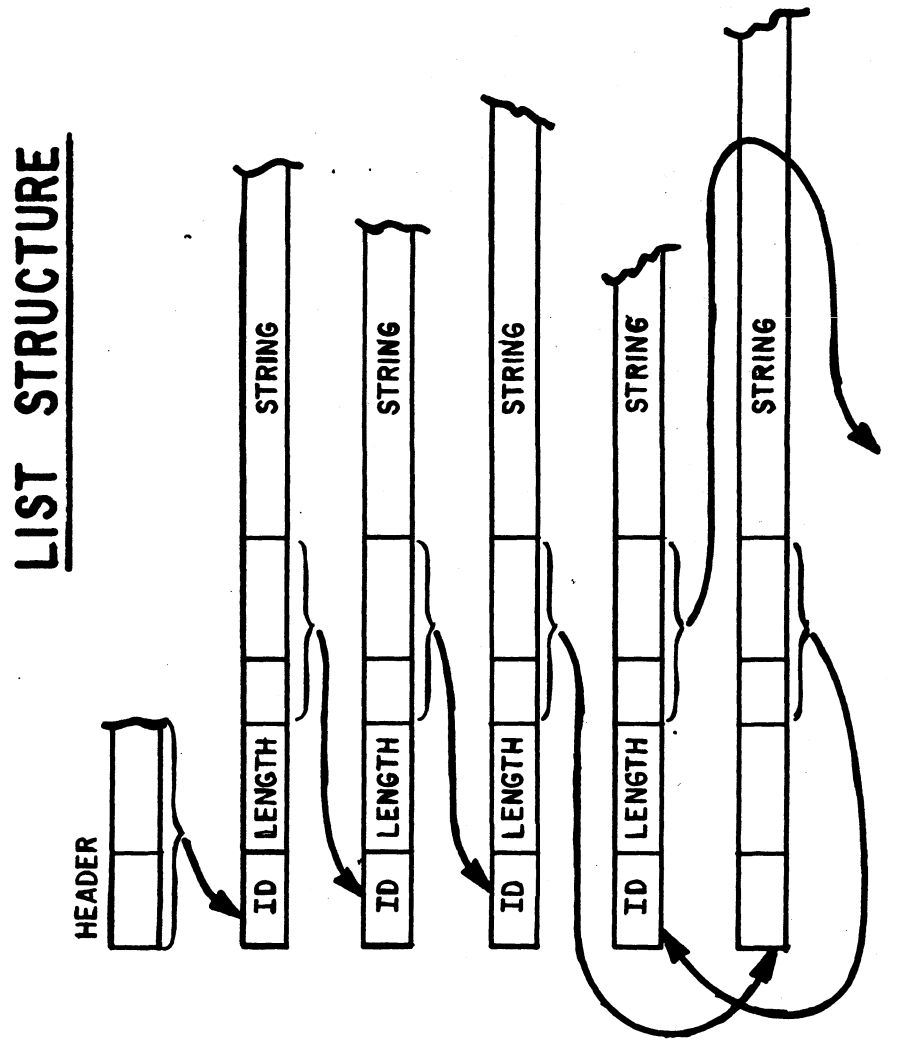
1. One method of achieving a high degree of conversationality in a time-sharing system, is to employ an interpretive system for program creation and execution.

2. In the interpretive system, the source language is converted to an internal tabular form instead of being compiled to object code. The internal tabular form must then be "interpreted" to be executed.

3. Each statement or line of the source language is handled individually as entered. The program is compiled incrementally into the internal tabular form a statement at a time. Hence, the term "incremental compiler" is inherent to an interpretive system.

4. This mode of operation provides interaction on a line-by-line basis. The user will know immediately that a statement has been rejected or accepted since each statement is compiled individually.

5. The internal form is generally a chained list structure. Each statement represents an element in the chain. The statements are chained together in statement number order. Since each statement points to the next one in the chain, variable length elements and changes to elements are accommodated easily.



6. Each element of the chain includes a control portion that contains the statement number or identity, the length of the element, the forward pointer to the next element, statement type, etc. In addition to the control information, the element contains the string representing the source language statement.

7. Generally, a form of Polish notation is used to represent the source language in the "string." This string must be interpreted by an execution monitor in order to run the program. A discussion of Polish notation is included in the following section of the outline.

8. Since a program is executed interpretively by chaining through the statements of the program, it is possible to limit execution to a range of the total statements. The interpretive system provides "incremental execution" of the source program.

9. A program can be partially executed, variables then displayed, and then execution continued. The nature of the interpretive system makes for a highly interactive mode of operation during both program creation and execution.

10. New statements are added and incorrect statements are replaced by simply adding or replacing elements in the chain. Except for changing the pointer in at least one statement, the remaining statements are unaffected. The complete program is not recompiled each time a change is made. Only the new or corrected statements are processed by the "incremental compiler."

11. The capability of incremental compilation and execution provides a valuable feature in interpretive systems called, "Desk Calculator" mode. The user can, in effect, write a one statement program that is immediately compiled and executed. Thus one-time slide rule or desk calculator type computations can be performed at the time-sharing terminal. This mode of operation must be distinguished from the normal program creation mode. A special character can be employed to indicate the mode desired.

12. The interactive advantages of the interpretive system are obvious. Unfortunately, the gain in conversationality is paid for in slower execution speed. Interpretive execution is not efficient when compared to object code performance.

13. This problem becomes acute with the "number cruncher" type job. There is one line of reasoning that says that "number crunchers" or "floating-point grinders" should not be run from the time-sharing terminal. Develop the program there, but run the production job in the traditional batch environment. Several approaches have been tried to improve execution speeds of the interpretive system.

14. Special hardware in the form of additional instructions to process the list structures is one approach to solving the performance problem. There is an RPQ for the Model 50 that will increase execution speed on the order of 20 to 30 per cent for CPS programs.

15. Another approach to increasing execution speed is to provide an option to compile object code from the internal tabular form. This technique was employed in QUIKTRAN[2] to improve program execution performance. The following figures provide some relative guide to execution performance:

Interpretive Execution	200 statements/sec.
Compile from internal form to object code	8,000 statements/sec.
Compile from source language to object code	20,000 statements/sec.

NOTE: These are execution speeds, not compilation speeds!

16. In the interpretive system, the program is generally retained only in the internal tabular form. The source language input is discarded when the internal form is compiled. In order to list a program, it is necessary to recompose the source language from the internal tabular form.

17. The recomposition capability provides an interesting possibility for interpretive systems. If multiple source language compilers are available to produce the same internal form, the recomp- capability could be used to translate one source language, i.e., FORTRAN to PL/I or vise versa.

18. The first interpretive time-sharing system is credited to the Rand Corp. The system was called JOSS (Johnniac Open-Shop System) and provided an ALGOL dialect language called the JOSS language. QUIKTRAN, CPS, and APL are IBM interpretive time-sharing systems.

E. What is Polish notation?

1. Polish notation is an alternate method of expressing the relationship between an operator (in this instance arithmetic) and the two operands (variables) which it binds. The notation that all of us are familiar with and the one used in the Fortran and PL/I languages is Infix. That is, the operator in question lies between its operands; thus:

A + B is Infix notation whereas
AB + is one form of Polish notation

2. In Polish notation, if the operator lies to the right of its operand pair, then the form is called reverse or suffix Polish. However, sometimes a form of Polish called prefix Polish is used, in which case the operator is placed to the left of the operand pair that it binds; e. g. :

+ AB

3. Here are some examples of infix notation paired with the corresponding reverse Polish form:

Infix	Polish
A + B - C	AB + C -
A + B x C	ABC x +
Z = A + B	Z A B + =
Z = A x B + C	Z A B x C + =
Z = (A + B) x C	Z A B + C x =

4. The advantage of Polish notation over the more familiar infix notation lies in the fact that the order of the operators in the Polish string reflects the order in which these operators are to be executed. Consequently, in an interpretive mode of program execution only a single scan is needed to convert the Polish string into executable object code.

5. Operands, when encountered in the scan, are placed in a push down stack. When an operator is met, the two top most operands are accessed from the stack and combined with the operator. The temporary operand representing the result of the operation relative to the accessed operands is then placed back in the stack and the scan continues as before.

6. Polish notation was developed by the Polish mathematician J. Lukasewicz as a "normal form" for the representation of formulae in logic. The notation permits an unambiguous sequential specification of the order of evaluation of logical and arithmetic expressions without requiring the use of parentheses. For this reason it has been found useful as a normal form for computer oriented mathematical languages. The work by the above named mathematician was done in the 1920s in Warsaw on a broken typewriter, and thereby hangs a tale.

7. CPS uses reverse Polish notation to represent the PL/I statement in the "L" string.

8. Converting from Infix to Polish Form

a. The order of operands remains the same in both source and target string. A push down stack is used to permute the operators so that they occur in the target (Polish) string in the order in which they are to be used. This order reflects the precedence hierarchy given below.

low precedence	(	when in the stack
	=	
to	+ -	
	* /	
high precedence	(	when outside the stack

b. The rules for manipulating the stack are as follows:

1. Source operands bypass the pushdown stack and enter the target string directly.

2. If an operator in the source string has greater precedence than the operator at the top of the stack, then enter the operator into the stack; otherwise, remove the operator currently at the top of the stack, move it to the target string and repeat this test.

3. A left parenthesis always enters the stack and is assigned lowest precedence.

4. A right parenthesis always forces operators from the stack until a left parenthesis is encountered.

c. The attached page contains an example which shows the intermediate steps in forming a Polish string.

9. Converting from Polish Form to Object Code

a. Rules

1. An operand, when encountered in the scan of the Polish string is entered into a pushdown stack.

2. When an operator is met, the two topmost operands are accessed from the stack.

3. Object code with a temporary operand is outputted.

4. The temporary operand is placed in the stack.

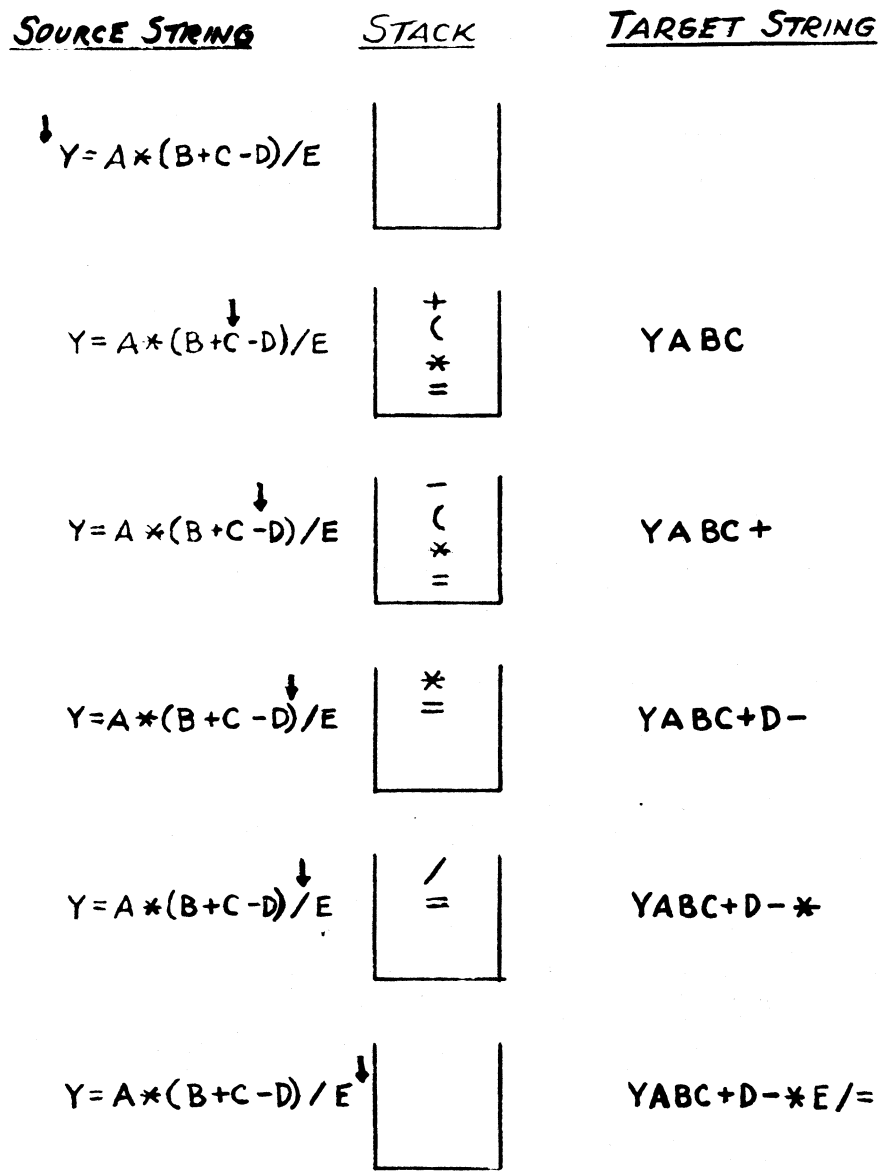
b. Conversion of a Polish string to object code is illustrated on a following page.

10. Summary

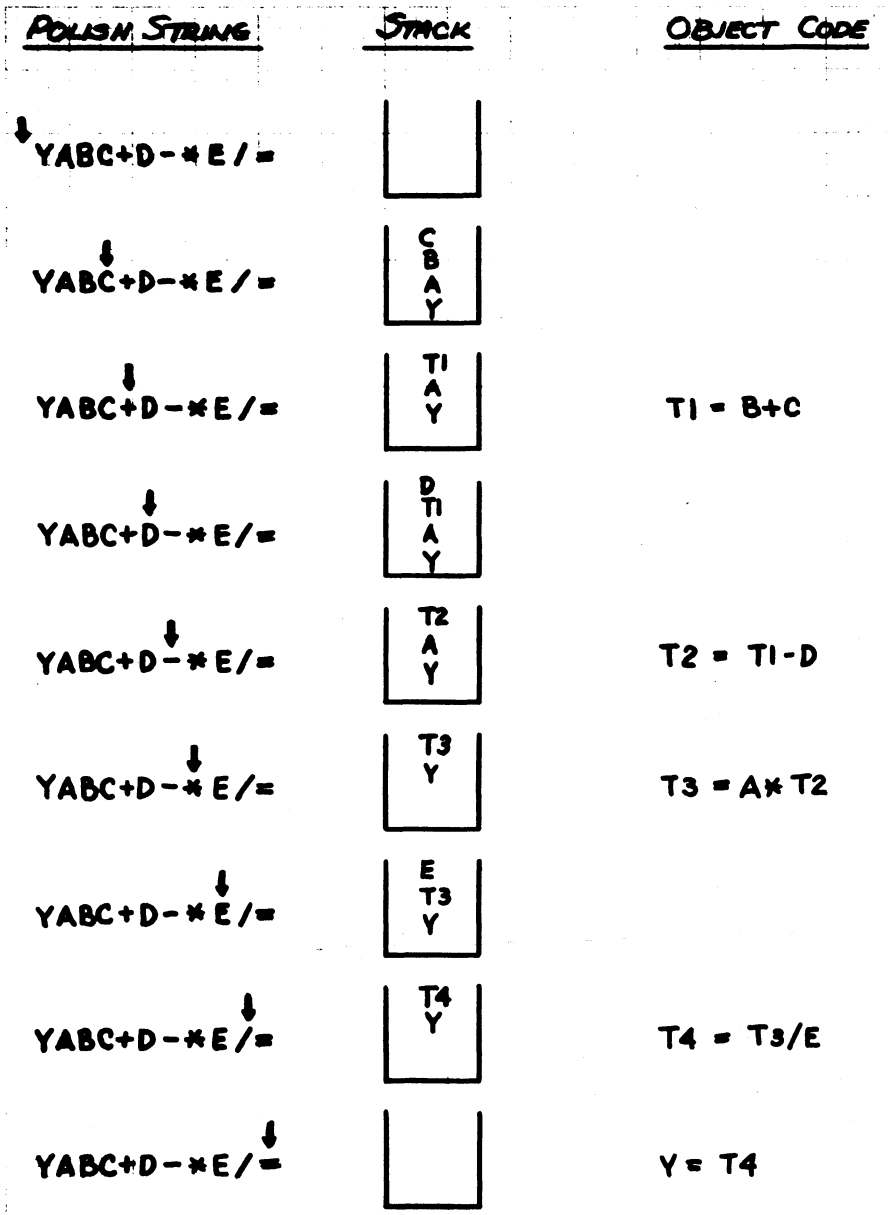
a. The order of operators in a Polish string reflects the order in which these operations are to be executed. It is for this reason that the original infix form of the arithmetic statement is converted to Polish notation for the L string.

b. This conversion from infix to Polish form is done only once, namely by the incremental compiler at the time the L string is formed.

c. The subsequent scan of the Polish string to produce object code is done each time the original arithmetic statement is to be executed. However, this scan is relatively fast because of the ordering of the operators within the string.



INFIX TO POLISH CONVERSION



POLISH FORM TO OBJECT CODE

F. Storage Management in Time-Sharing Systems:

1. Storage management is a fundamental consideration in the implementation of time-sharing systems. As in any multi-programming environment, storage allocation and management techniques are critical design factors determining the number of users, system response, and system thruput.

2. One can conceive of unlimited main core storage in a time-sharing system that would permit all users to reside in-core at all times. This "infinite" size one-level store would accommodate programs of any size for each user. If this main storage were combined with an infinitely fast CPU, then the users could be serviced instantaneously. An "ideal" time-sharing system would be the result.

3. Of course, in practice the "ideal" time-sharing system does not exist and various techniques attempt to simulate this "ideal" system. The techniques of main storage management can be classified as follows:

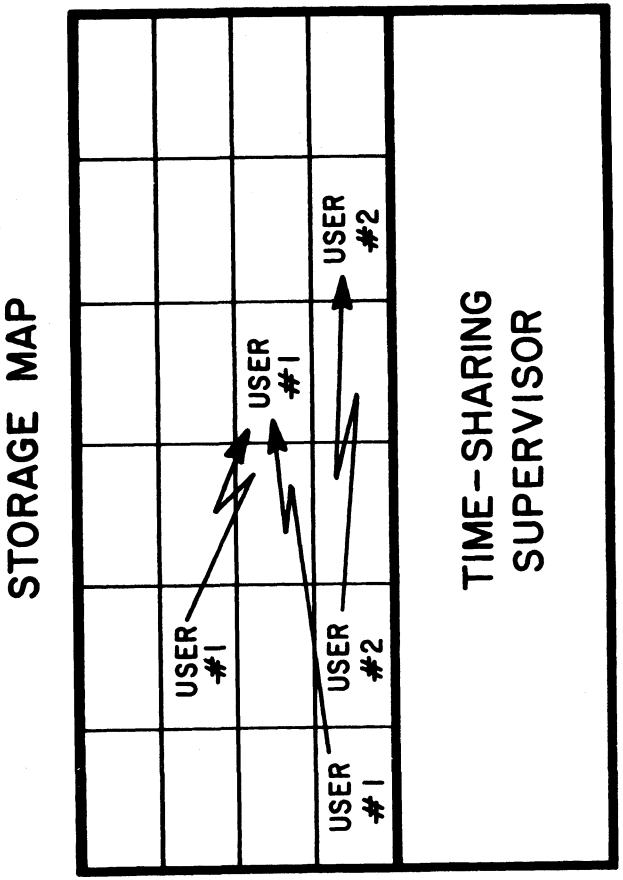
   a) Segmented-core-resident
   b) Swapping or Roll-in/Roll-out
   c) Paging

4. In segmented -core-resident storage management, core storage is divided into blocks. These blocks are assigned to users as required from the pool of blocks made available when main core was segmented. A user retains those blocks assigned to him as long as he is logged-on at the terminal. When he logs-off, the blocks are put back into the pool for subsequent use by others. Note that for a given user, the program (s) and data remain in the same blocks of core storage. They are not swapped-out to a second level storage device while another user's program is swapped-in.

5. The advantages of the segmented-core-resident system are obvious. It is the closest approach to the "ideal" system cited above. System overhead is minimized since users' programs are not swapped between time-slices. It is not necessary to wait for one program to be rolled-out before another can be rolled-in for execution. Where programs remain in storage, there is no contention for channel time and storage access cycles.

6. Since the segmented -core-resident system uses standard size blocks, it is impossible to have pieces of non-contiguous storage wasted because they are too small.

SEGMENTED-CORE-RESIDENT SYSTEM



7. One disadvantage of this method of storage management is the large core necessary to implement this technique. LCS is required in order to put a number of users in core and keep them there. Even so, this implementation strategy limits program size to the maximum number of blocks allowed each user.

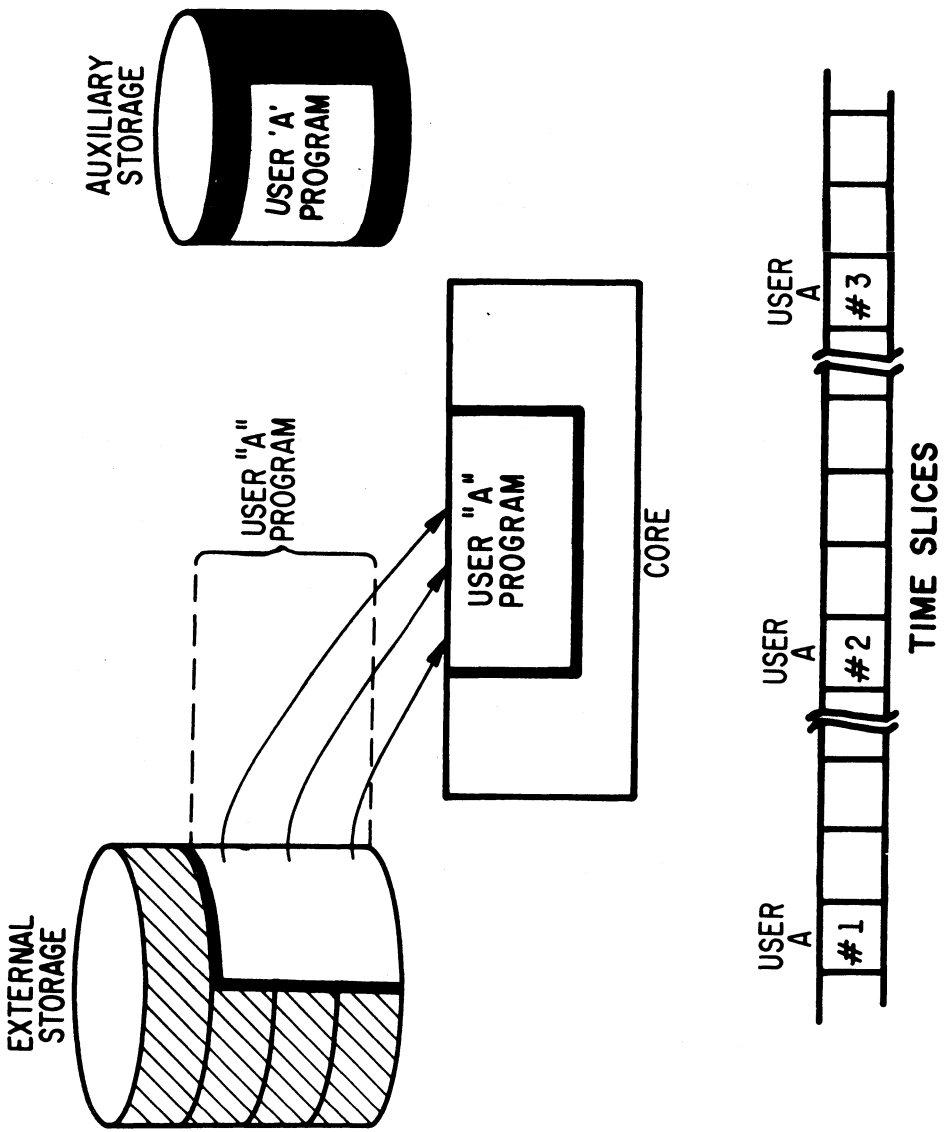
8. From a practical standpoint, only the interpretive type system, using list processing techniques can effectively use the non-contiguous set of blocks that a user would acquire in the fragmented system. CPS is an example of the segmented-core-resident storage management method. The CPS Monitor program and the user blocks make up a partition of MFT or a region of MVT.

9. The most common method for storage management in time-sharing systems is some form of swapping. Program swapping or roll-out/roll-in is the technique whereby a user program, User A for instance, at the end of its time slice is written out to an auxiliary storage device. The necessary registers, etc., are saved and written out with the program. Another program, User B, is then read from the auxiliary storage into core storage. When the loading is complete, User B's program is executed. At the next end of time slice, Program B is in turn swapped with the next program in the ready queue, User C for instance.

10. Swapping techniques can use one or more partitions for swapping. In the single partition swap, the CPU will have to "wait" while one program is rolled-out and the next is rolled-in. In this serial approach, necessary with one partition, the CPU will be waiting 30 - 35 per cent of the time. Of course, the single partition swap requires the least amount of memory to implement time-sharing.

11. Multiple partitions permit several user programs to be on their way in or their way out to auxiliary storage at the same time. Hence, roll-out/roll-in is overlapped with execution. The more partitions, the more likelihood that the CPU will be executing a users program. Of course, the more partitions there are, the greater the core requirement. The housekeeping and overhead are also greater because of the complexity of multiple partitions.

12. A slight variation of the single partition swap is the so-called "onionskin" approach. In this technique, only enough of the first program is rolled out to accommodate the next program in the queue. Program A is "peeled back" just enough to make room for program B. If program C is greater than B, additional "peeling" will be required to accommodate C. The CTSS system at MIT uses the onionskin approach.



13. It is relatively easy to relocate a program when it is initially loaded before execution begins. It is another matter to relocate a program that has been partially executed during a previous time slice. Consider the program that came to time-slice-end when a base register had just been loaded. If this program is saved and properly re-loaded to the same storage locations, the program can be re-started without problem. However, to move this program to another partition or set of locations introduces some real complications---in base addressing, adcon resolution, etc. Dynamic relocation generally requires some hardware assist such as a relocation register to adjust addresses by the relocation constant. It is possible to relocate dynamically by programming if the user programs adhere to very strict, perhaps intolerable, programming conventions.

14. In the single-partition swap, all programs are loaded into the same starting locations. Hence, dynamic relocation is not necessary. Dynamic relocation is also not necessary in the multiple-partition swap if the programs are always rolled in to the same partition on each time slice. Interpretive-type systems using list structures can be easily relocated to different swapping partitions since address-dependent information is either relative or nonexistent.

15. The problem of "Who's minding the store?" extends to auxiliary storage also. Space for swapping, program libraries, scratch-work areas, check point, and data file areas must be allocated on the direct-access storage units of the time-sharing system.

16. Rolling out a program is essentially a check point operation. Certain status information must be saved and written out with the program. Contents of the general registers, floating-point registers, PSWs, etc., are obviously required for the next time-slice. The simplest approach is to allocate enough direct-access storage for each user to accommodate the saved status information plus the entire swapping partition.

17. RAX uses this straightforward approach in rolling out its single swap partition. A 2311 cylinder is used for each user-swap area. A simple count of available cylinders is incremented and decremented as user programs are rolled in and rolled out.

18. APL uses a sophisticated extension of the above approach in swapping a "work space." Only that portion of the "work space" that is occupied is actually written to disk, thereby reducing I/O time and space required on the disk. The swapped areas are no longer fixed-size blocks. Thus, allocation and management of auxiliary storage space becomes more complex as variable length blocks are accommodated.

19. It should be noted that the complete program is rolled out at the end of time slice and rolled in before the next time slice started. No matter how much of the code is actually executed during a time slice, all of it is written out and subsequently read in. Obviously, a great deal of code, especially exception routines, is repeatedly moved in and out but never executed. This characteristic of swapping contributes to the overhead and degrades performance. Time-sharing architects developed new techniques in second generation systems to eliminate or reduce the swapping of unnecessary code. Paging or page-turning is a refined storage management technique intended to improve program swapping time. Programs are developed in fixed-size blocks called pages. Only those pages actually required or demanded are placed in core storage during a time slice. At time-slice end only changed pages need be written to auxiliary storage device. The subject of paging and virtual memory is treated in a following section.

G. Paging--An Advanced Storage Management Technique

1. In a multiprogramming environment a job step generally is constrained to execute in a contiguous area of core storage. This area of core may be static and hence used sequentially by a succession of jobs. On the other hand, the areas may be dynamic in that its fences come down after completion of the job step in order to release the core used. The static area of core is called a partition and is representative of multiprogramming under DOS or OS-MFT. In contrast, the region concept of OS-MVT represents a more general approach to storage management. A job step is allowed to specify its core requirement on a control card. The operating system will then attempt to satisfy this request from its pool of unassigned core. However, in both the partition and region approach, all core requests made by the job step during execution must be satisfied from the partition or region. Both region and partition, to repeat, represent contiguous areas of core storage.

2. When a time-sharing subsystem uses an existing operating system which handles the multiprogramming by either a partition or region approach, the time-sharing tasks roll out from and roll back into an assigned partition or region. Furthermore, a module of code which is rolled out of some core area prior to completion of execution must be rolled back into its original core locations. This is the essence of static relocation. In contrast, dynamic relocation implies the freedom of the system to move code which has not completed execution to another set of core locations before allowing execution to resume. To do this without restriction extra hardware is required.

3. Consider the following example:

loc	Instruction
100	L 15, X' 108'
104	BALR 14, 15
108	DC A (700)

This program loads register 15 with the address constant 700 at location 108 and then branches to location 700 within the program. If this code is moved unaltered elsewhere we have, say:

loc	Instruction
1500	L 15, X' 108'
1504	BALR 14, 15
1508	DC A(700)

In order for the relocated program to execute properly, the instruction at location 1500 must generate address 1508 and the instruction at 1504 must cause a branch to location 2100 to take place. This can be accomplished by introducing a single hardware register. The content of this relocation register is added to core addresses generated during instruction executions. It is these relocated core addresses that are sent to the address register. In our example the relocation register should contain, obviously, 1400.

4. It has long been thought that to support a large general purpose time-sharing system, a less restrictive method of managing core is necessary. Partitioned core, together with the absence of a dynamic relocation facility, imposes many system constraints. In the following allocation scheme real core is divided into standard-size units called blocks, whose size depends upon system architecture and software design considerations. A given program can occupy several core blocks. These need not be contiguous. Any available blocks may be assigned to a program. Moreover, if a core block is assigned to a section of a program, it is only because that section is actually referenced during program execution. This program section to which a block can be assigned is called a page. Page size and block size are the same. A page is referenced by execution of an RX, RS, or SS type instruction or by the instruction address in the PSW. If the referenced page already has a core block assigned, instruction execution continues to completion. Otherwise an interrupt occurs, and the supervisor assigns a core block to the offending page within the program. Note that no unreferenced page can have a core block assigned to it. Consequently, only the active portions of a program will take up core.

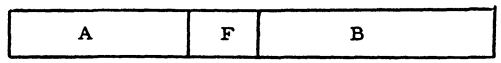
5. With this introduction let us take another tack. Often a programmer is forced to code an application in two phases. In phase one he constructs an algorithm to implement the application. In phase two he faces up to the fact that there is not sufficient core to execute his algorithm. That is to say, the programmer structures the application as an overlay module. Inasmuch as phase two coding is concerned with implementing a strategy to execute the algorithm, it is evidently desirable to relieve the programmer of this burden and place it on the system.

6. This was first done on the ATLAS machine produced in 1961 by Ferranti Ltd. in England. In this machine a program was spread out over core and drum. During execution the program, of course, referenced code not yet in core, hence an interrupt occurred. The page containing the referenced item was assigned a block of core by the supervisor, and the contents of that page was read from the drum into the newly assigned block. When I/O was complete, the program was again given control, and the instruction that caused the interrupt was re-executed. All of this was entirely transparent to the programmer. He merely assumes that he always has as much core as he needs for his application and hence can dispense with creation of an overlay program structure.

7. The key to implementing such a scheme is recognition that a core address represents two things: First, it represents the unique name of some data item; and secondly, it also represents the core location of the said item. The thought occurs of splitting apart these attributes. Let what is ordinarily the address now represent only the name of the data. Using this name as an argument, do a table look-up to determine the location of the data. Thus, we now have the concept of a set of names whose range is dependent only on the addressing capability of the hardware. In contrast, the set of core locations has a range determined by the amount of core possessed by the system. Inasmuch as System/360 works with a 24 bit effective address, names can range from 0 to $2^{24}-1=16$ million, whereas core locations range from 0 to typically 512K or 1M, depending upon how much core the machine possesses. The programmer codes within this name space of 16 million bytes and so can dispense with overlays. The memory corresponding to these names is nonexistent. This conceptual memory within which the programmer works is called a virtual memory. Only its addresses are real. These addresses are generated in the usual fashion by adding a 12-bit displacement to the contents of the base register. However, this virtual memory address provides an argument for a table look-up to determine the corresponding real core address, if any. If the translation (look-up) process indicates that the referenced item is not in core, an interrupt occurs. The supervisor assigns a core block to the virtual memory page containing the referenced item. Then an I/O operation is initiated to read the page contents (on drum or disk) into the assigned core block. One can think of a typical translation table entry as

consisting of the following fields:

Virtual Memory Page Address



A — Core block address

F — Flag
Off--core block address is valid
On--core block address is invalid
(i.e., no block is assigned)

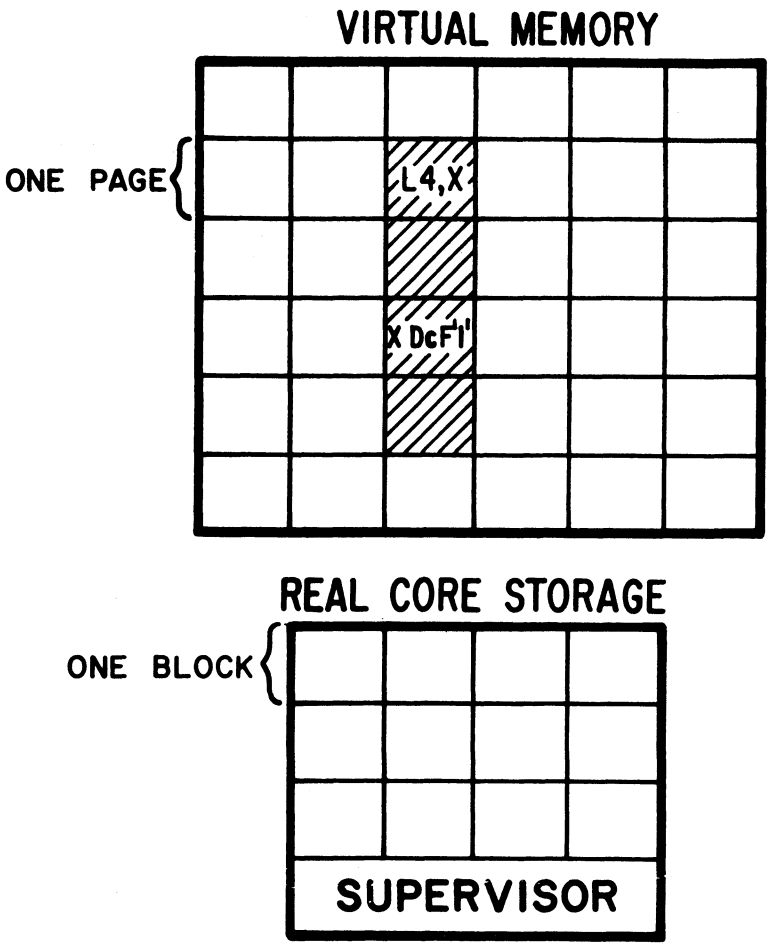
B — External page address =
Device address and relative
page number within the device

Upon completion of the read operation the program is eligible to resume execution. The instruction which caused the original interrupt is re-executed.

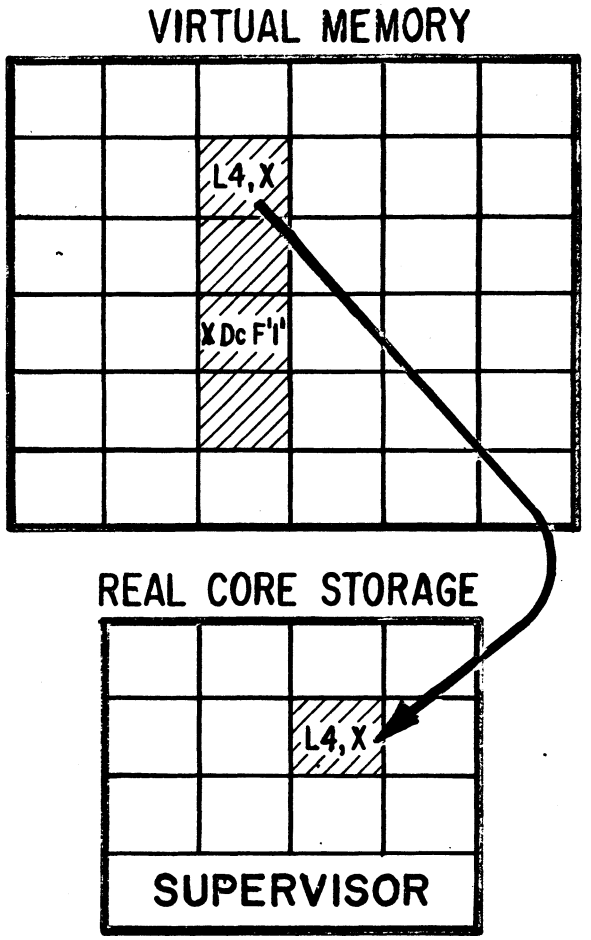
8. Note that this method of memory management allows a program to execute within core blocks which are scattered throughout memory. Any available core block is eligible for assignment to any program. Moreover, if a page is assigned at one time to some core block, later during execution (say during a subsequent time slice) it is not necessary for that page to be assigned to the same core block.

Thus, this fragmented approach to storage management offers dynamic relocation. There is an obvious penalty in implementing this scheme, and that is the increased instruction execution time resulting from the need to translate the page address to the core block address. However, typically extra hardware is introduced to minimize this overhead. Finally, let it be emphasized that a virtual memory is unique to each user. That is to say the system keeps a separate translation table for each task in the system. Depending upon system architecture, the translation may be accomplished completely by means of hardware registers or by use of tables in core at time of use, together with associative hardware to decrease most translation times.

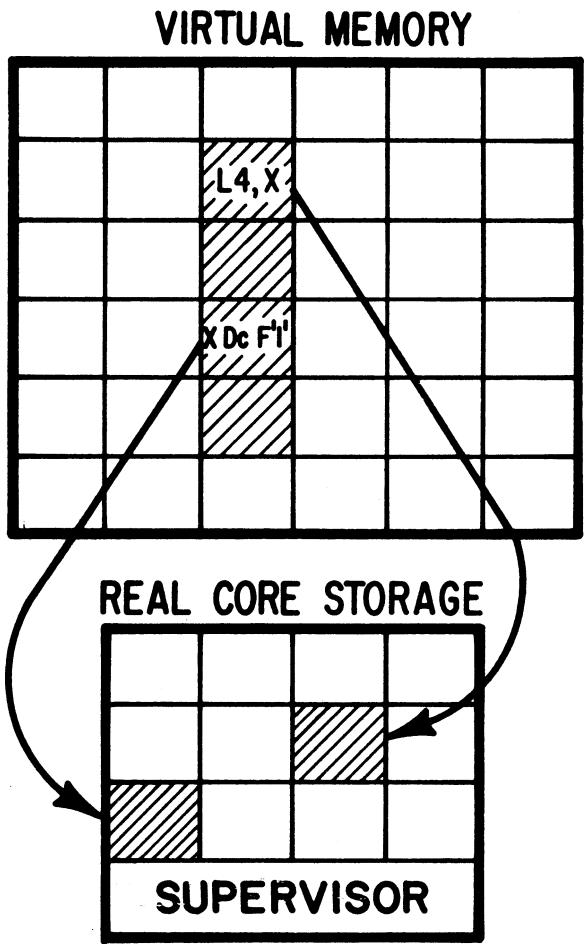
9. Consider the following with reference to Figures 1, 2, 3, and 4. Figure 1 illustrates the situation extant after a programmer has caused a four-page program to be "loaded" into his virtual memory. Since program execution has not yet commenced, no portion of the program has been mapped into core storage. "Loading" the program into virtual memory involves essentially making entries in the user's translation table. These entries relate the addresses of the "loaded" virtual memory pages to addresses of program pages in some library on a direct access device.



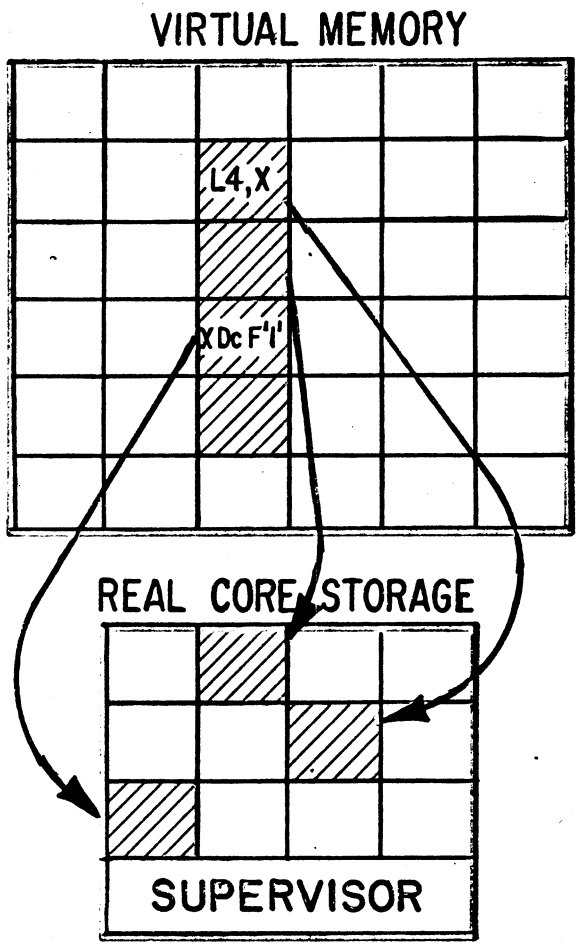
1. LOADED VIRTUAL MEMORY - NO REAL CORE BLOCKS ASSIGNED



2. LOADED VIRTUAL MEMORY - ONE REAL CORE BLOCK ASSIGNED



3. LOADED VIRTUAL MEMORY - TWO REAL CORE BLOCKS ASSIGNED



4. LOADED VIRTUAL MEMORY — THREE CORE BLOCKS ASSIGNED

In Figure 2 program execution has commenced. In the time span between the snapshot represented by Figure 1 and that of Figure 2, the system has assigned a core block to the initial virtual memory page. The content of that page is read into the assigned core block. Note that during execution within this page there is a reference to data item X located in another page--one to which no core block has been assigned.

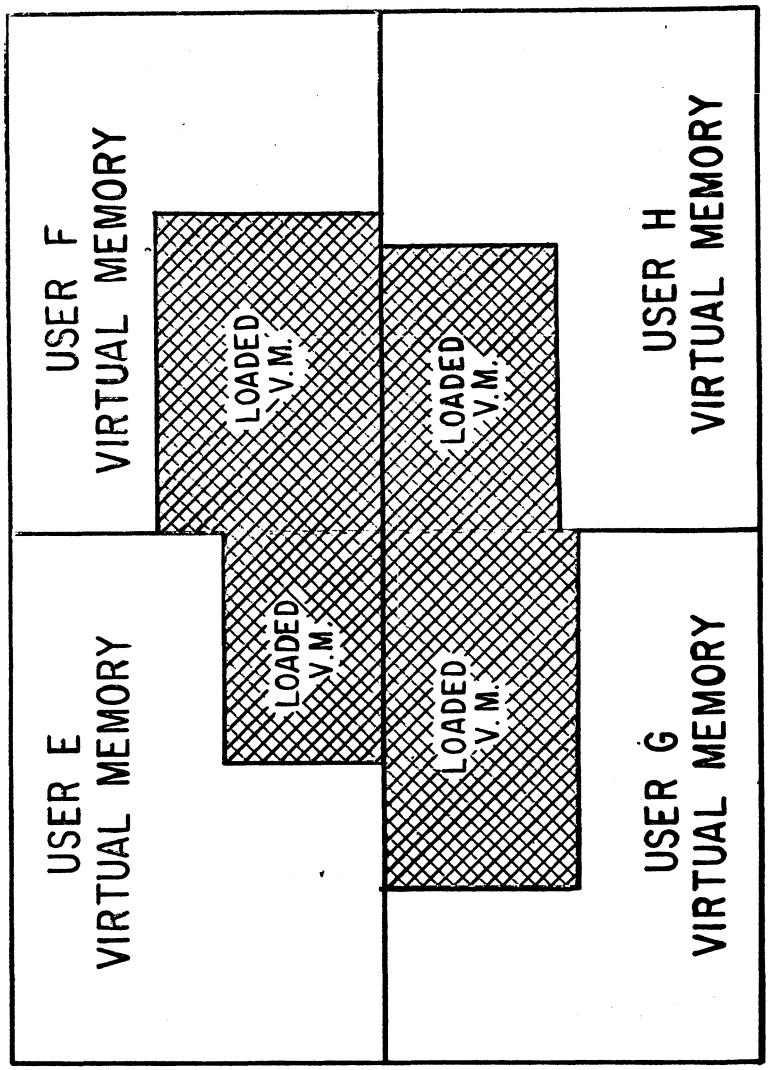
Figure 3 illustrates the correspondence between virtual memory pages and core blocks as a result of execution of the L 4, X instruction in the first page. In the process of translating the virtual memory address corresponding to the symbol X, the hardware was signaled that no block had been assigned to the page containing the data item X. The resulting interrupt caused control of the system to pass to the supervisor. The supervisor assigned a core block to the "offending" virtual memory page and caused its content to be read into this block.

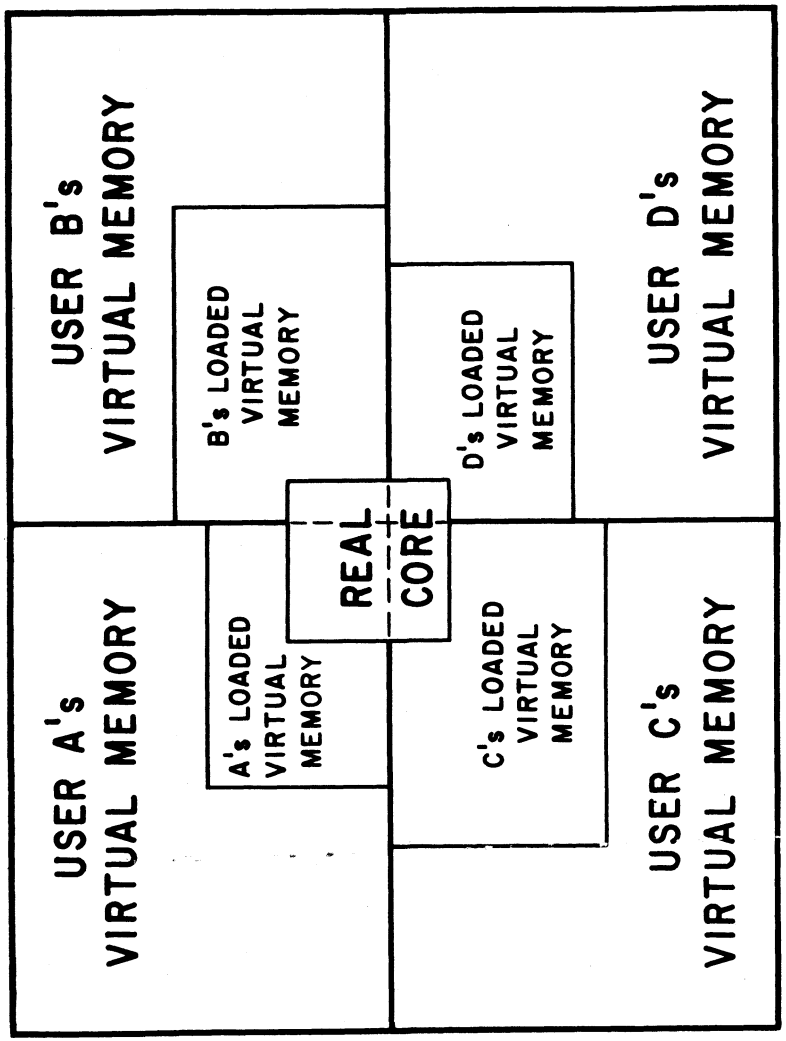
Figure 4 indicates that another block has been assigned to a page in virtual memory. This assignment, for example, could have been triggered by the instruction counter stepping across the boundary from Page 1 to Page 2 in virtual memory. Dynamic address translation of the instruction address would generate an interrupt to the supervisor. Note that Page 5 in virtual memory, although a part of the program under execution, does not have core block assigned to it. If during program execution no reference is ever made to this page, no block will be assigned at all.

10. The above discussion makes reference to a "loaded" virtual memory. Since a virtual memory is one which the user has but the real system does not, some remarks about "loading" a program module into something purely conceptual are in order.

A program module to be loaded into virtual memory is located at load time in some library on direct access storage. The loader of virtual memory causes the module to be loaded by making entries in the translation tables which exist for the user and his virtual memory. The loader allocates a set of virtual memory addresses as yet unassigned and assigns these to the module. Table entries are made to relate these allocated virtual memory addresses to the symbolic device address and relative page numbers used by the module on direct access storage. No text ever moves.

Note that the loading of information into virtual memory is independent of the contents of real core memory. Indeed, only the paging mechanism of the system will cause a real core block to be assigned to a loaded virtual memory page.





The two accompanying figures each illustrate four users with loaded virtual memories. In one case none of these users have real core blocks assigned to the loaded portion of their virtual memories. In the other illustration each user has a portion of his loaded virtual memory mapped into real core.

H. Scheduling Techniques in Time-sharing Systems

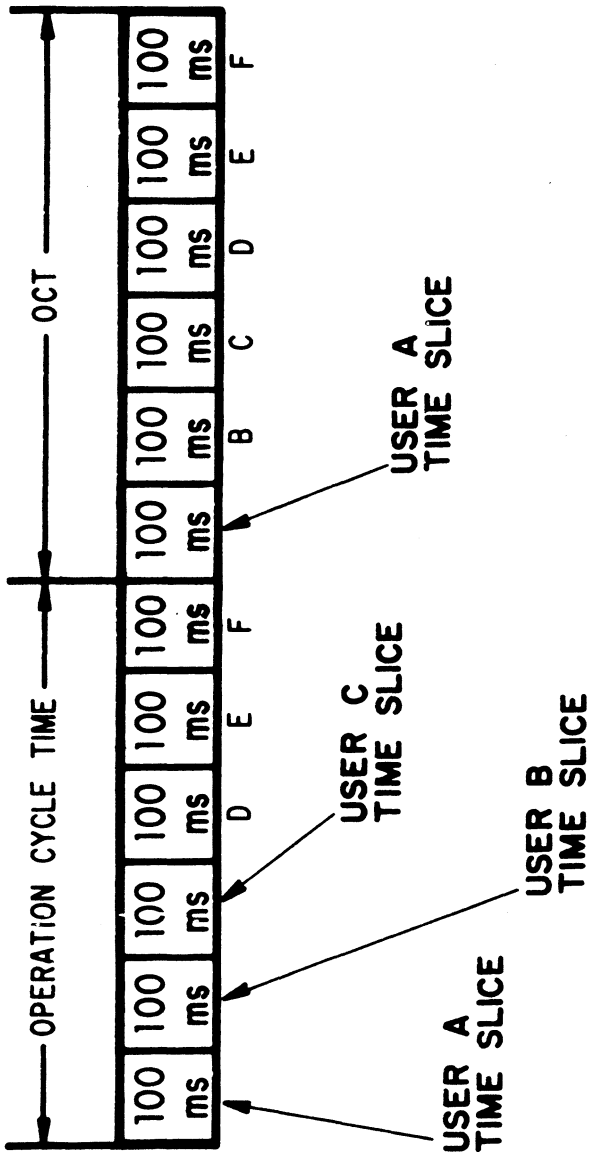
1. "Who gets the CPU and for how long?" is the question addressed by the scheduler. In managing this important resource, the scheduler is an integral part of the time-sharing supervisor. It plays the role of time-keeper, dispatcher, and housekeeper. Scheduler designs can range from a very simple, straightforward approach to very complex algorithms to multiplex user programs. The purpose of this section is to provide an understanding of this range of scheduling techniques. Its activity affects response and performance and is a key consideration in the design of the system.

2. In a non multi-programming system the CPU scheduling problem does not exist. There is only a single program in the user area of storage, and this program is to run to completion prior to starting the next program. Even in a multiprogrammed batch operating system, scheduling the CPU is relatively minor in nature and is usually referred to as task switching. In this latter case the supervisor contains a task switching routine. This routine is entered when the supervisor has completed all the processing it can do as the result of some interrupt. Task switching passes control of the CPU to the highest priority ready task. In many instances determination of this task can be made by comparing the dispatching priority of the task which was last in execution with the priority of the task causing the interrupt. At worst only a partial scan of the chain of task control blocks is necessary.

   In contrast, scheduling the CPU in a time-sharing system is accomplished in many ways--some simple, others highly sophisticated. Some of these scheduling algorithms are discussed in the following paragraphs.

3. Simple Round Robin

   This method is the most straightforward. All tasks are served on a first-come-first-served basis and are ordered in a single queue. Whenever a task reaches time slice end, its dispatching priority is lowered by one, and those of all the tasks are raised by one. Note that these contending tasks could be core resident or using storage on a roll-in, roll-out basis. CPS tasks, for example, are core resident and use the terminal line number instead of an internal priority as an ordering device for its queue. The queue consists of commutator bytes, one for each terminal line in the system. The dispatcher scans the queue in circular fashion. If a



given commutator byte indicates that the associated task is either not logged on or is in head-scratching mode, then it is by-passed, and the scan continues in its search for a task with processing requirements.

In this category of CPU scheduling algorithms, the time slice value may be fixed or may be supplanted by a function work slice. In a work slice approach, a task is allowed to hold the CPU, if possible, until a compilation is completed or in the case of interpretive systems, until a fixed number of statements have been executed.

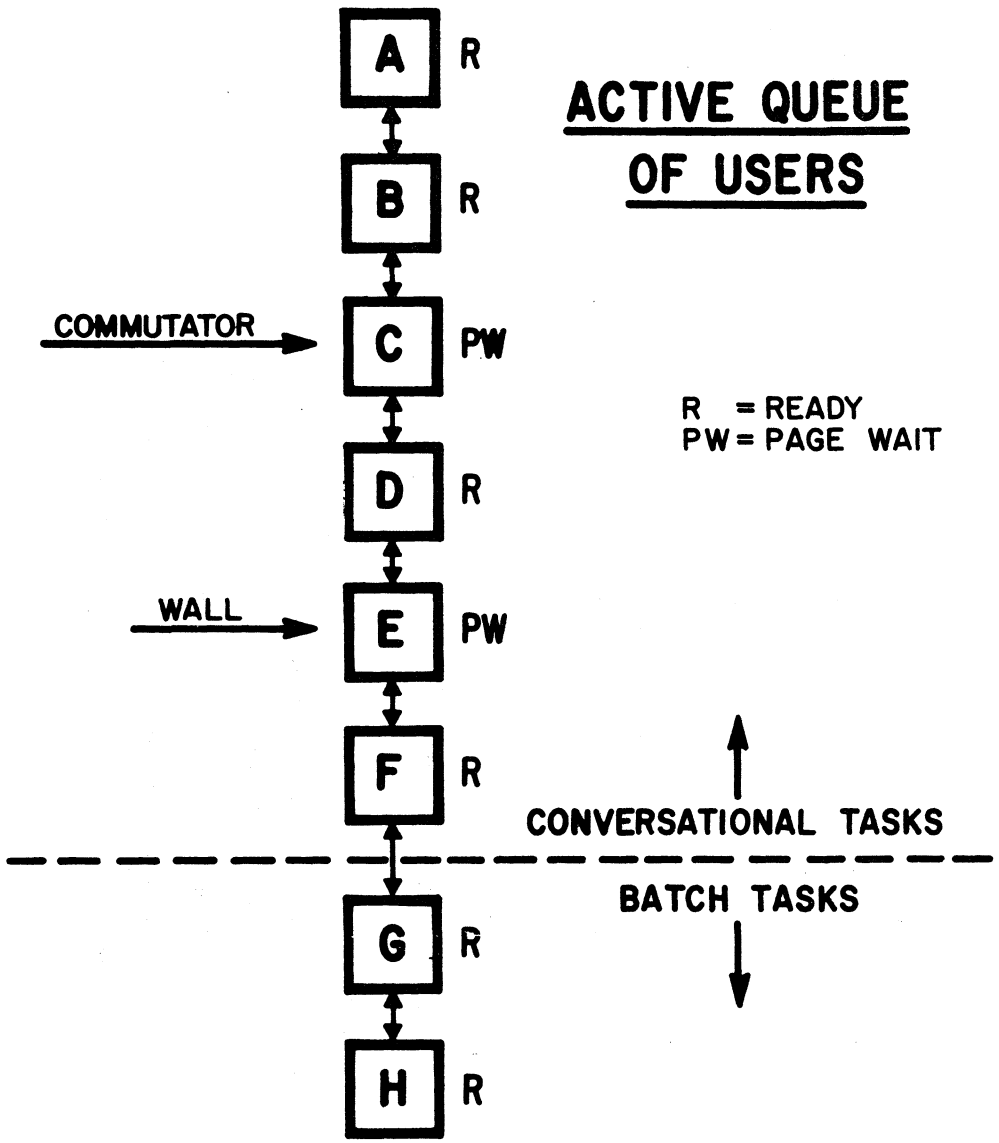
TSS uses a variant of round robin with variable quanta. However, this will be treated separately.

4. Multi-queued Exponential Scheduling

In this scheme used by MIT in their CTS System, a given task is in one of several queues. Each queue has an assigned priority (level number.) Choice of an initial queue for any task is dependent on the core requirements of that task. The less core required, the higher the priority of the queue to which it is assigned. All tasks are handled FIFO within a queue. No queue is serviced unless all higher priority queues are empty. Whenever a task makes a terminal I/O request, it is placed at the end of the highest priority queue. But if a task is in hard execution at the end of time slice, then it is placed at the end of the next lower priority queue. Finally, let Q represent the time slice value associated with the highest priority queue. Successively lower priority queues are assigned time slice values of 2Q, 4Q, 8Q, 16 Q ... . This scheme favors small short running programs typical of a student environment. "Grinder" type jobs progress to successively lower priority queues, but once getting the CPU, they receive longer and longer time slices. In the MIT CTS System, storage is managed in a roll in- roll out basis.

5. Modified Round Robin with Variable Quanta

This system is used by IBM Model 67 TSS to schedule virtual memory resident tasks for the CPU. Task control blocks representing users belong to one of two queues. Those tasks in the Inactive Queue are those waiting for a response from the terminal. This type response is measured in seconds or longer. Such tasks have previously been paged out of core storage and even from the paging drum to a slower direct access device. The scheduler (task dispatcher) never searches this queue of dormant tasks. Instead, it confines itself to a subset of tasks in the Active Queue. This subset of tasks in the active queue lies between two pointers, the Commutator, and the wall, as illustrated in the accompanying figure.



Those tasks above the Commutator are ready for their next time slice. Those below the Wall are ready for their current time slice. The tasks between the Commutator and the Wall pointers are currently undergoing their time slice. Their referenced virtual memory pages have been assigned core blocks, to be held until end of time slice. These are the tasks currently being multiprogrammed. The scheduler scans the tasks between the Commutator and the Wall (i.e. scans the tasks "in the Wall") and gives the CPU to the first ready task encountered. If no task in the wall is ready, then the wall pointer is moved down to include a new task. At this time, the translation tables for this task are paged in by the dispatcher.

When a task in the wall comes to time slice end, it is so flagged and hence will no longer be considered for CPU control. When the task pointed to by the Commutator reaches time slice end, the Commutator moves down the active queue. It stops when it finds a task is Ready or Page Wait status. Intervening tasks in Time Slice End status are passed by and reset to Ready. Thus, effectively all tasks in the active queue are treated in round robin fashion. Those tasks in the Inactive Queue which receive a terminal response are moved back to the Active Queue. They are flagged "Ready" and placed just below the Wall pointer.

6. There is a refinement of the above scheduling algorithm which reflects the use of demand paging as the basis of storage management. It is evident that the ability of a task to hold the CPU improves steadily during its time slice, i. e. during the period of time the task resides in the wall. Each page exception causes another core block to be assigned to the offending virtual memory page within the task's virtual memory. Unfortunately, at time slice end just when the task has accumulated a number of core blocks, these core blocks must be given up. Unchanged pages are immediately made available for assignment to other tasks. Changed pages are scheduled for writing to paging drum or disk then freed for use by other tasks. In an attempt to keep the entire system in balance, the following decision is made whenever a task reaches time slice end. If the paging time during the time slice exceeds the CPU time accumulated during the time slice (this latter is the time slice value), then the task is flagged ready, hence stays within the wall and receives another time slice. Thus the task does not relinquish its assigned core blocks. It continues to accumulate them as paging exceptions occur during the extended time. Additional time slice values called quanta are made available to this task. However, for each N additional quanta obtained while in the wall, tasks are penalized by being flagged

Time Slice End the next N times they are included in the wall. Thus, in the long run any task averages a single quantum per time slice. The maximum number of successive quanta allowed any task is set as a system parameter.

7. IBM Model 67 TSS makes provision for servicing background (batch) tasks as well as time-shared tasks. Batch tasks are represented by task control blocks at the end of the Active Queue. When the wall pointer has reached the last time-shared task, a decision must be made whether to return the wall pointer to the top of the Active Queue or allow it to include tasks to be run in batch mode. This decision is made on the basis of the operational cycle time set as a system parameter. If this time, typically two or three seconds, has not been exceeded, then a batch task is included in the wall. Otherwise the wall moves immediately to the top of the Active Queue and no batch task receives a time slice this operational cycle time.

8. Table Driven Schedulers

It is difficult to find two people with the same philosophy of CPU scheduling in a time sharing environment. This arises, in part, because the subject is not yet well understood. As yet, there is a paucity of practical experience with large scale time-sharing systems. In any event, there is ever present the desire of most installations to tailor the algorithm to their job profile or at least to adjust it according to the dynamic work load on the system.

In spite of the parameterization of key variables, such as length of time slice, maximum number of quanta per time slice, operational cycle time, etc., flexibility of CPU scheduling has been very difficult to achieve. The most flexible method of implementing a scheduler-dispatcher is to have the behavior of every task dependent upon entries stored in a table. These entries can be easily changed, perhaps even dynamically, in response to a changing environment.

In one such system, each row in the table contains all the parameters necessary to control the behavior of any task. Each task with processing requirements is queued relative to a single row in the table. All tasks in one such queue have the same priority. They are ordered within the queue by their scheduled start time for their next time slice. The accompanying illustration shows a hypothetical table and suggests entries relevant to CPU Scheduling. Notice that on the basis of parameters stored in a given row (level number) a task may be subsequently queued on the same or on a different row. Thus, over a terminal session a task may be scheduled in a variety of ways. A single table can easily accomodate, for instance,

A SAMPLE SCHEDULING TABLE

Level No.	P	Q	T/S	DTR	TSE Level	TWAIT Level	Max. Pages
0	0	1	50 ms	0 ms	1	0	100%
1	0	10	100 ms	3400 ms	1	0	25%
2	1	10	100 ms	4200 ms	2	2	25%
3	2	3	100 ms	4200 ms	3	3	25%
9	8	16	100 ms	5000 ms	10	6	20%
10	9	32	100 ms	6000 ms	11	6	20%
11	10	64	100 ms	6800 ms	11	6	20%

P = Priority

Q = No. of successive quanta per time slice

T/S = Amount placed in timer at start of time slice

DTR= Relative length of interval at which a task wants to be given a time slice

TSE level = Scheduling table entry to be used when time slice end occurs

TWAIT level = Scheduling table entry to be used when a task leaves terminal wait status

Max. Pages = Percentage of total core blocks a task may accumulate during a time slice

both exponential multi-queued scheduling plus the round robin variant of TSS.

It is expected that the table driven approach to CPU scheduling will provide a flexibility not heretofore present in more conventional approaches. Less of the algorithm is embodied in fixed coding scattered throughout various modules of the supervisor. In addition the table represents a single, easily modifiable entity by means of which any installation may readily adapt to its unique time sharing requirements.

IV. Multi-Processing Systems

A. Introduction:

1. The term multi-processing system literally designates any computing complex formed by assembling together several processors, presumably CPU's. Thus, such diverse systems as ASP, SABRE, the QUIKTRAN configuration, the FAA 9020 complex should all qualify as multi-processors. However, there is a strong tendency to restrict the term multi-processing system to one which offers "availability" and which, in addition, provides all CPU's a common jointly addressable memory.

2. Typical multi-CPU configurations tend to fall into one of several categories.

Tandem — In the 7740 - 7044 system used by QUIKTRAN the 7740 pre-processes messages from the communications network while the 7044 does the main processing.

Satellite-parent (slave-master) — The ASP system uses the parent as a support processor (typically a model 50) to handle SPOOLing and job scheduling. The slave(s) (a model 75, say) executes the jobs in the input queue. There is sufficient flexibility in this arrangement to allow job executions to take place in the satellite processor (as in version 2 of ASP).

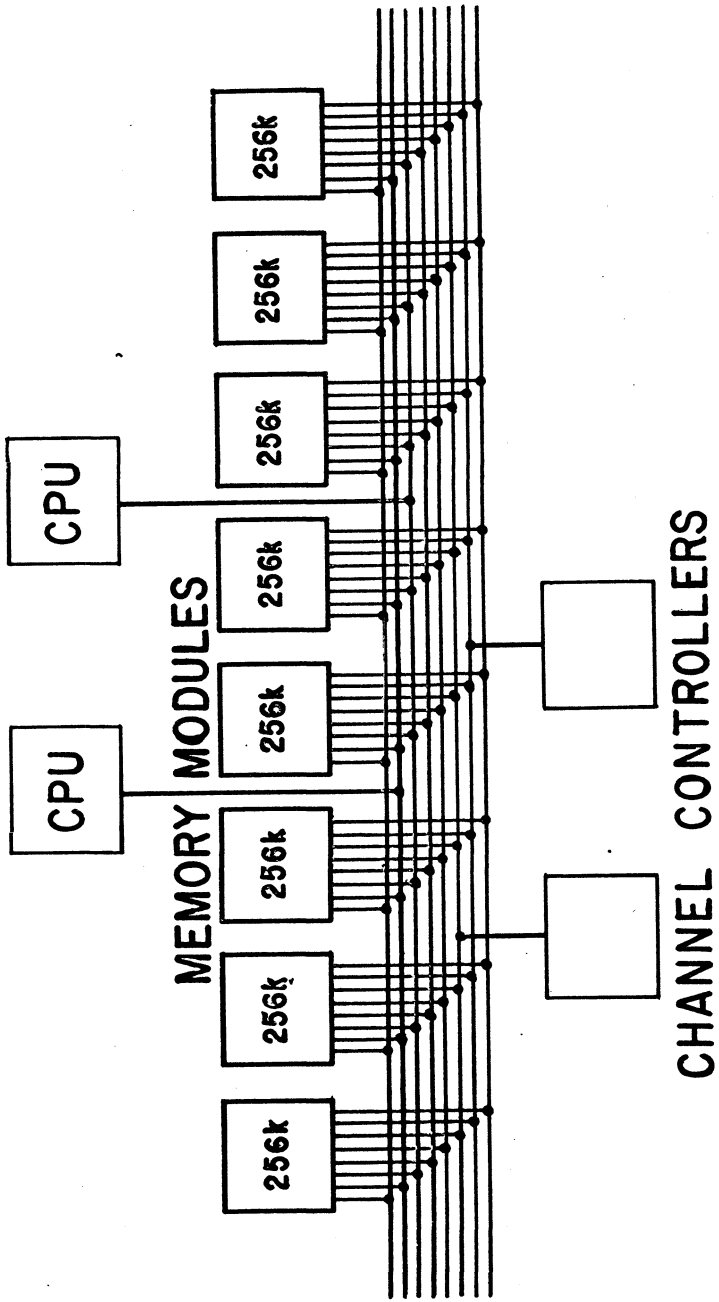
Duplex — The SABRE system uses two 7090's to process the American Airlines Reservation applications. One 7090 handles on-line transactions from some 1400 terminals located throughout the USA. The other performs in-house operations while serving as a switchover backup in case of failure of the on-line 7090.

If system reliability is exceptionally critical, as in manned spacecraft tracking, then the duplex system may use both CPU's running in lock step. In this case, inputs to each sub-system are identical and all work is performed in duplicate. The term dual is often used to describe this mode of operation.

3. Do any of the above offer the ability to carry on when a system component, including a CPU fails? If the answer is yes, then such systems offer "availability" and can justify being called multi-processing systems. However, some people require the CPU's to address a common bank of memories to qualify the system as multi-processing.

4. The multi-processing systems discussed below are those which were designed expressly with "availability" as the prime objective. This ability of a system to continue to perform even though at a degraded level, is achieved through a redundancy of components. Multiple channels and control units provide alternate data paths. Modular memory boxes and multiple CPU's complete the redundancy requirements. In the case of processors which share a common memory, there is in addition a flexibility gained from use of a pool of system resources. Such systems operate under a single control program in the common memory. All CPU's have access to the single job queue and single set of system libraries.

5. In summary, a multi-processor system in the context of this monograph, embodies "availability" as its paramount objective. As an additional benefit, the multi-processing system is more flexible than two stand-alone systems. In the latter, all resources of a failing system are lost. Furthermore, the resources of two stand-alone systems cannot be pooled to rectify a temporary imbalance in one of the systems. In contrast, "load leveling" is an automatic by product of a true multi-processor design. It is impossible to have a queue of work present for one CPU and have the other CPU in WAIT state. In conclusion, let it be noted that computing power is a secondary goal in multi-processing systems. Performance is somewhat degraded because of contention. There is hardware contention arising when two or more CPU's attempt to access the same memory box. There is software contention when two CPU's attempt to access the same system data sets in main or in auxiliary storage. Availability is the key word for multi-programming design.



B. System Considerations

1. IBM currently markets commercially two multi-processing systems in which the CPU's address a common memory. These systems are Multiprocessor 65 using OS-MVT and the Model 67 using TSS. Before examining each system separately, features common to both are treated below.

2. Multi-System CPU Signals

a. A CPU can receive an external interrupt for one of several reasons. Its timer may have turned negative or some other CPU may have executed a WRITE DIRECT instruction. In addition, an external interrupt can be caused by a malfunction alert signal which is automatically generated when another CPU encounters a machine check.

b. Inter-CPU communication, known as shoulder tapping, is not restricted to system recovery operations. The multi-processor Model 65 makes use of the shoulder tapping ability of the system to conform to the basic design standards of OS-MVT. This will be discussed in more detail later.

3. Storage Areas for CPU Control

a. Every model in the System 360 line uses an area in low core which contains machine oriented words. Old and new Program Status Words (PSW's). the timer. the Channel Status Word (CSW) and Diagnostic Scan-out area are fixed fields in this area. In a single CPU system. this area encompasses addresses 0 to N where N is less than 4096 The area is "owned" by the CPU in the sense that it contains data used or generated by the CPU.

b. In a multi-processor environment each CPU must have its own "low core" area. The area is a full 4096 bytes long and is called either the Prefix Storage Area or Permanent Storage Area (PSA). Each CPU must have the ability to reference the PSA's of the other CPU s. The PSA of a failing CPU contains information which would possibly allow another CPU to "take over".

4. Floating Address Switches

Core storage in a multi-processing system is packaged in modular units, typically 256 K bytes each. A system with 512 K bytes of storage has two memory boxes, whereas a million byte system has four. The address range of each memory box is controlled by the setting of a separate address switch located on a configuration control console. If a particular memory box fails, it can be varied off-line and the floating address switches set to give the remaining on-line boxes a continuous range from zero to the maximum now.

possible. It is obviously desirable for an installation to locate the various prefix storage areas in separate memory boxes in the interest of availability.

5. Configuration Control Panel

A multi-processing system can be divided into two isolated subsystems. Each subsystem will have a CPU, channels, control units and memory boxes. Each processor can sense the status of all partitioning switches on the control panel regardless of the partitioned status of the processor. Through partitioning, a Model 67 system, for example, could run TSS in one subsystem and run OS-MVT in the other with its CPU in Model 65 mode.

The floating address switches on this panel are used to assign memory addresses starting from zero for each partitioned subsystem.

The panel is also used to reconfigure the system or subsystem by isolating a failing component. Such reconfiguration is manual or automatic depending upon the sophistication of the software.

6. Operating System

When any CPU in the multi-processor system (assuming it has not been partitioned) takes an interrupt, control passes to the single supervisor. Therefore, it is possible to have more than one CPU executing supervisor code at the same time. To prevent two CPU's from simultaneously updating the same supervisor table, test and set discipline is enforced.

At the beginning of any subroutine which modifies a table, the TEST AND SET instruction is executed. The referenced storage byte is called a lock byte. If the condition code indicates that the lock byte is already set (all ones) then another CPU is currently in the act of updating the table. The CPU attempting to execute the subroutine will generally continue to execute the TEST AND SET instruction. When the other CPU completes the table update, it resets (zeros) the associated lock byte and returns to the calling program. The CPU attempting to enter the routine now generates a condition code from the TEST AND SET which indicates the lock byte is reset. It now commences execution of the routine. Use of test and set discipline forces CPU's to queue for use of a serially reusable resource. Modules of code which can be executed simultaneously by more than one CPU are called parallel reenterable.

The TSS Resident Supervisor contains a separate lock byte for each of its tables. OS-MVT as adapted for the M65 system contains a single lock byte to control all supervisor updating.

C. The Model 67 Multiprocessing System

1. Introduction:

Although IBM has developed systems with high availability in the past, such systems were tailored to meet specialized customer processing requirements. The Model 67 represents the first IBM true multi-processing system available as a standard model in the product line. Its basic design principles are derived from the need for extensive "fail soft" capability and to explore the use of dynamic address translation as an advanced method of core storage management. The supporting operating system is TSS. However, the CP 67 system is being used, for example, to use the Dynamic Address Translation Unit as a means of exploring the virtual machine concept.

2. Model 67 Configuration:

a. The System/360, Model 67 is a sophisticated extension of the Model 65. There are three configurations possible. These are simplex, half duplex and duplex.

b. The simplex system is not memory oriented. Thus it closely resembles the Model 65 except for the addition of a Dynamic Address Translation Unit to the CPU.

c. The half duplex configuration uses all components of the full duplex system. These are multi-tailed memories, a configuration console and Channel Controller Units. A memory oriented computing system requires a different type of core storage unit. This unit differs from that present on the simplex system. For this reason, a simplex system cannot be field upgraded to half duplex.

d. The duplex system contains multiple CPU's. A half duplex system can be upgraded in the field to a full duplex configuration by addition of another CPU and presumably more memory boxes.

3. Features

a. Modular Memories

Each memory box has multiple connections for use by CPU's and channels. Thus different core storage units can be accessed simultaneously by multiple CPU's and channels. These multiple direct connections are called tails, hence the term multi-tailed memories.

A special hardware component called tie breaker is brought into play when two accesses are made simultaneously on the same multi-tailed memory. The tiebreaker makes its decision based on predefined hardware priority levels.

b. Extended PSW

Power on brings up the Model 67 in Model 65 mode. In 65 mode the system runs with a standard PSW controlling the CPU. The change to 67 mode is made under program control. In this mode, the CPU runs under control of an extended PSW. This PSW has a completely different format from the standard PSW used by all other System/360 models. In particular, the instruction counter is 32 bits long. There is a bit to indicate whether the Dynamic Address Translation Unit is to be active or not. Fields in the standard PSW set at time of interrupt have been relegated to fixed locations in the prefix storage area of the CPU. A bit in one of sixteen special hardware registers governs the mode of operation of the system. "Extended PSW mode" is a term used to indicate that the Model 67 is executing as a Model 67, not as a Model 65.

c. Multi-Paths Between Devices and Memory

Access to a device from more than one control unit is accomplished by tape and disk switches. Access to a device through one of several channels is accomplished by control unit switches.

Each channel in the duplex system is attached to the system via a Channel Controller Unit (CCU). Channel addressing is defined by the CCU to which a given channel is attached.

Channel addresses 0 -6, for example, can be assigned to either CCU when the system is operating in the extended PSW mode.

d. Floating Address Switches

The position of a rotary switch gives the corresponding memory box a particular 256K range of addresses. When a memory box is undergoing maintenance, these switches are set to reflect the reduced address range starting from zero. When the system is partitioned, then the address switches are set to provide two address ranges both starting at zero.

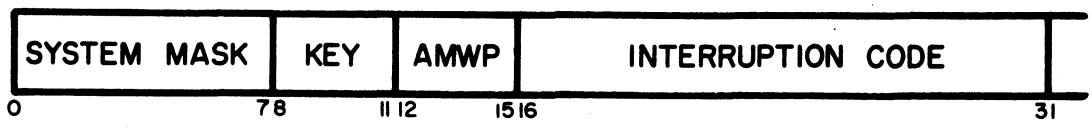
e. Configuration Unit Control Panel

This panel contains the following:

1). The floating address switches
2). A separate rotary switch operative when the system is running in the standard PSW mode. There is one such switch for each CPU, used to assign to that CPU either one of two Channel Controller Units.
3). Status switches to indicate the partitioned status of system units. The settings of these switches can be accessed by a CPU for sensing under program control . The status switches also serve to mark defective units.

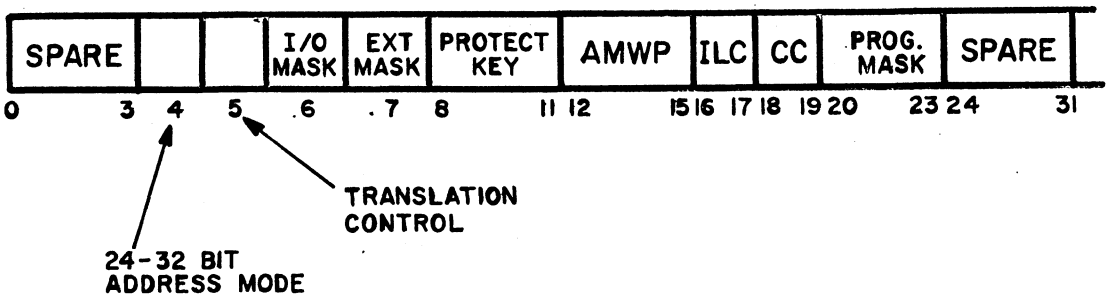
SYSTEM/360 PSW

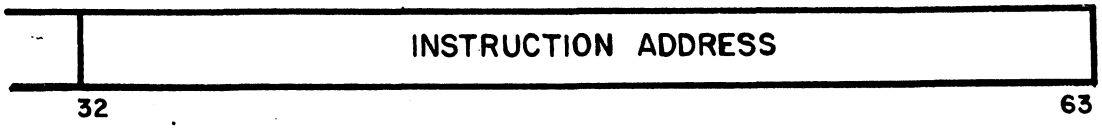
FORMAT:



MODE 67
EXTENDED PSW MODE

REDEFINED FORMAT:





f. High Resolution Timer

The interval timer has a resolution time of 13 microseconds. In contrast, most interval timers "tick" once every so many milliseconds. For third generation systems which operate at nanosecond speeds, resolution on the order of milliseconds is not fine enough. Time sharing systems, particularly, need high resolution interval times for accurate job accounting and task dispatching.

g. Extended Storage Protect

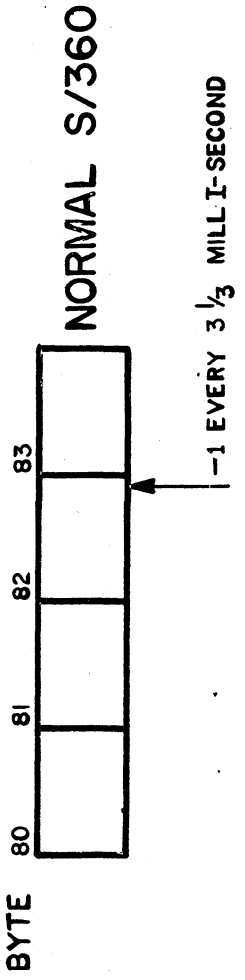
The storage protect key is increased from four to seven bits. In addition to the normal four bit storage key, there is a reference bit, change bit, and fetch protect bit. The change bit is used by TSS to determine if a core block has been modified during a time slice. Changed blocks must be paged out to auxiliary storage (drum or disk) before being made available for reassignment. The fetch bit is used to implement read only code. The reference bit together with the change bit is useful to determine if a core block containing reenterable code and shared by several users has been used recently. Unused shared pages are merely occupying core storage and should be paged out.

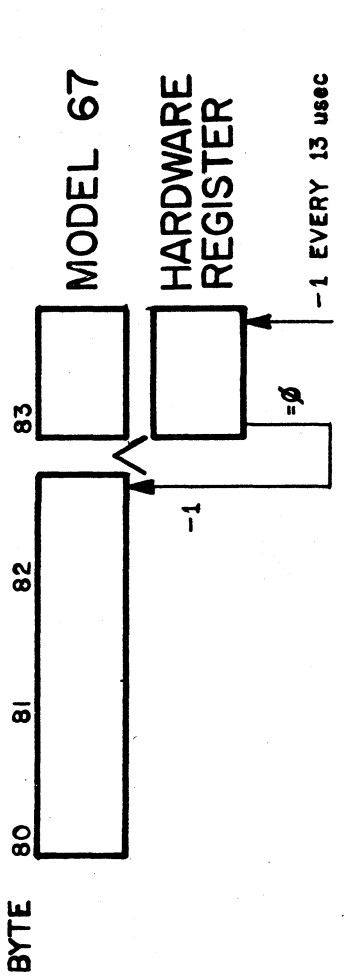
h. Direct Address Relocation

A Model 67 installation selects multiple 4K areas in core storage for CPU Control. None of these can coincide with the true low core area extending from 0 to 4095 in the common memory. The customer engineer will establish hardware prefixing at the time of installation. A CPU will reference its own pseudo low core area when an address in the range 0 to 4095 is generated by that CPU. This is accomplished by causing the hardware to monitor all addresses. Whenever an effective address with twelve high order bits all zero is generated, the prefix wiring will cause these zeros to be replaced by the 12 bit block address of the prefix storage area for that CPU.

To access data within the PSA of another CPU, a CPU need only execute an instruction whose operand generates the absolute address of the desired data. Such an address does not lie in the range 0 to 4095, hence does not contain twelve high order zeros. Prefixing does not take place and so the desired location can be referenced. Each CPU has a primary and alternate (back up) PSA. Note that the true low core area in the common memory is not usable by the system since prefixing prevents any CPU from accessing it.

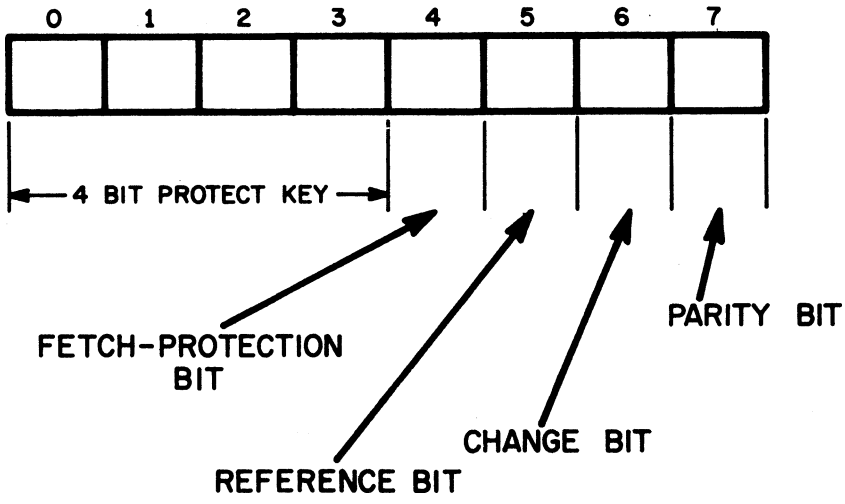
MODEL 67 HIGH RESOLUTION INTERVAL TIMER





MODEL 67
STORAGE PROTECTION EXTENSION

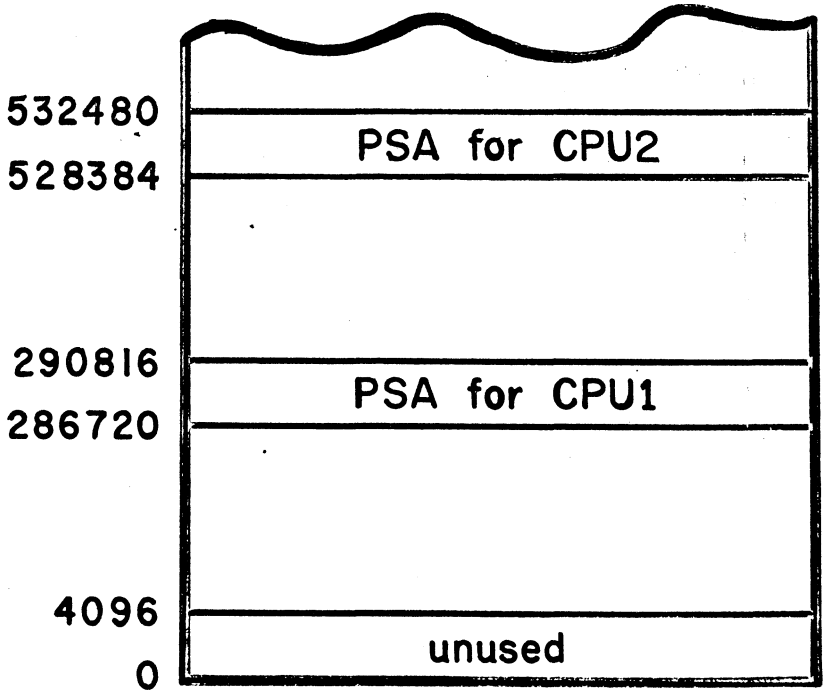
FORMAT of STORAGE PROTECT KEY:



REFERENCE AND CHANGE RECORDING IS ALWAYS ACTIVE!

65

CPU PREFIX STORAGE AREAS
IN A MULTIPROCESSOR CONFIGURATION



66

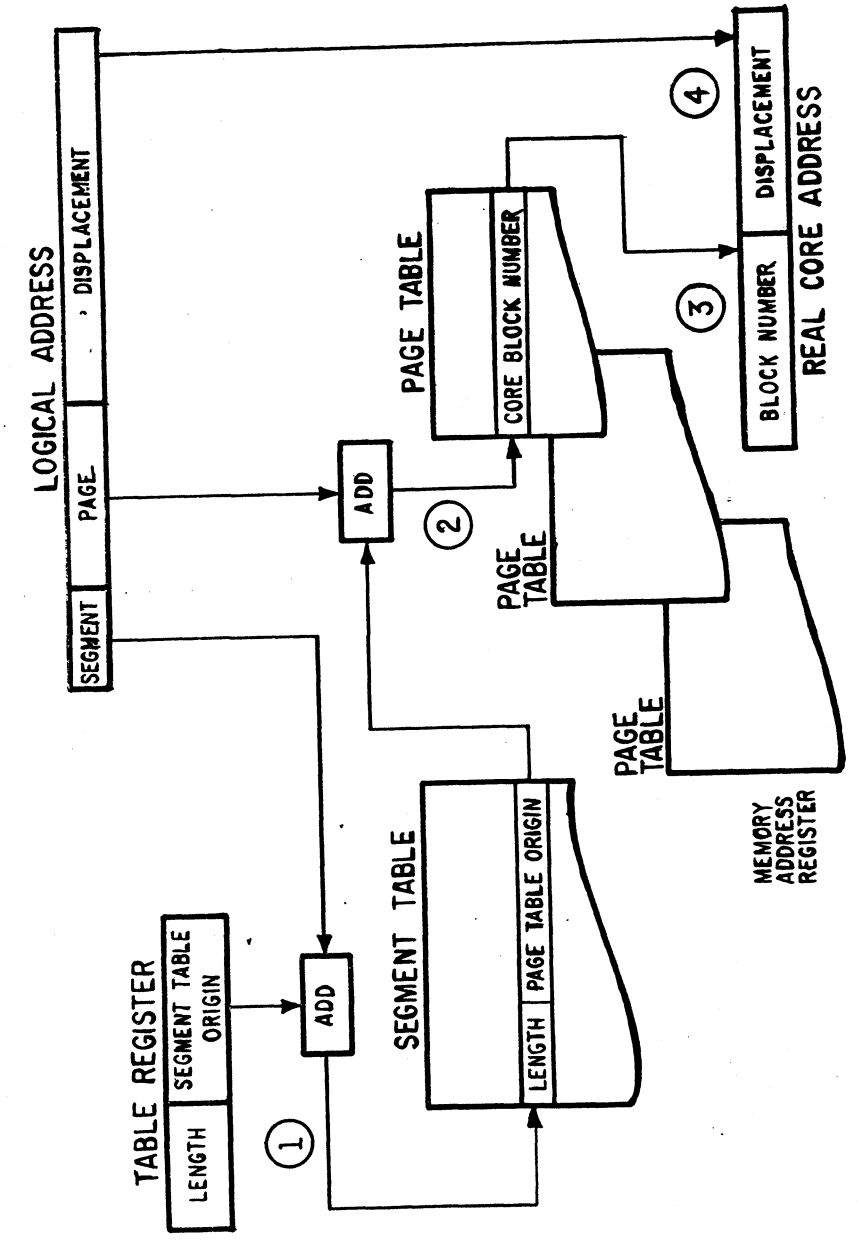
4. Dynamic Address Translation

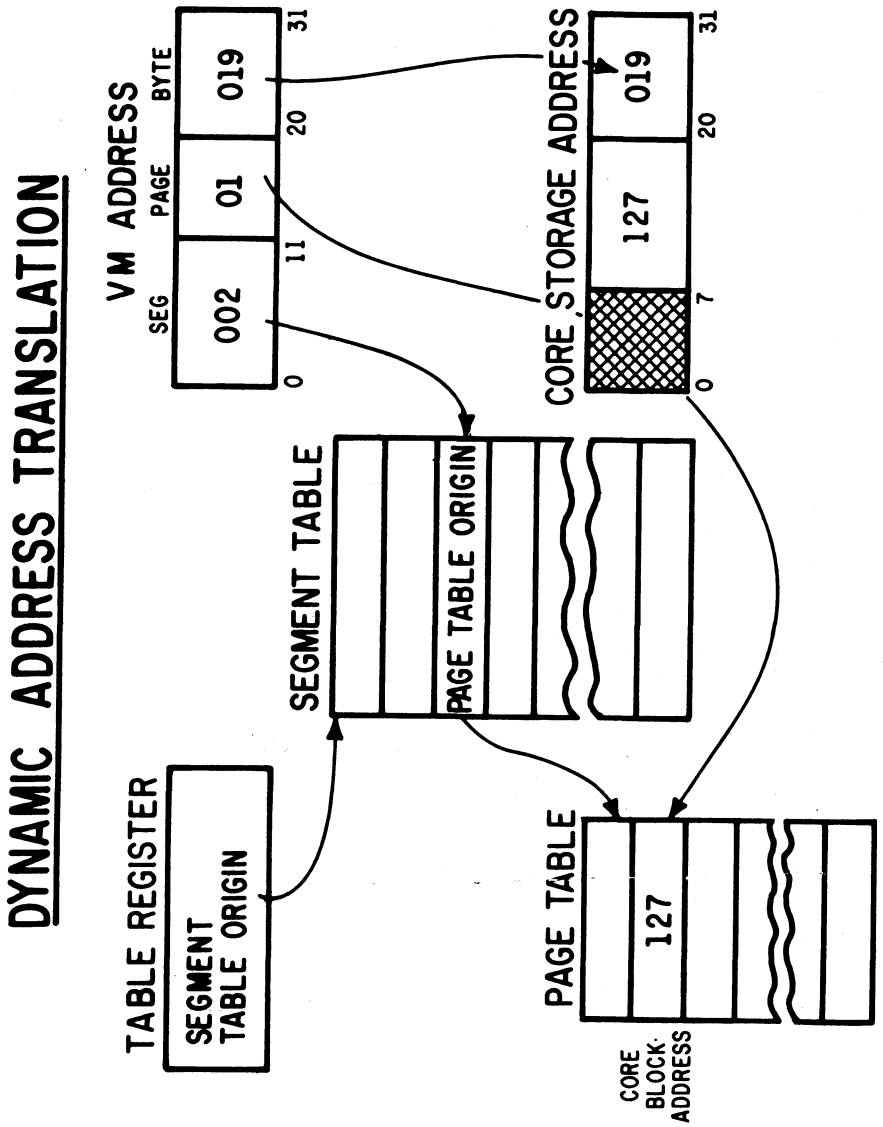
a. A Model 67 CPU contains a Dynamic Address Translation (D. A. T.) Unit. This unit is never active when the CPU is executing in standard PSW mode. When dynamic address translation is active, all processor storage addresses that originate from the processor (except for hardware generated addresses such as old PSW store and new PSW fetch) are subject to translation. Storage addresses originating from the channels are not subject to translation.

b. When an instruction referencing core storage is executed, an operand is formed from the base register, index and displacement as determined by the instruction format. When the D. A. T. unit is inactive, this operand is an effective address, hence is sent to an address register. When the extended PSW indicates that the D. A. T. unit is to be active, then the operand is a virtual memory address and so undergoes translation.

c. Translation is made by a double table look-up using a Segment Table and a Page Table. There are separate segment and page tables for each user in the system. A special hardware register called a Table Register contains the location of the segment table belonging to the user currently active. Within the virtual address being translated, the left most twelve bits serve as an index to the segment table. The corresponding segment table entry, if valid, points to the core storage location of a page table. Bits 12 through 19 in the virtual address serve as an index to this page table. The corresponding entry, if valid, contains the core block number assigned to the page in virtual memory being translated. The displacement is an invariant under the translation process and serves to reduce the translated entity to the byte level.

RELOCATION ACTION
TABLE LOOKUP





d. When a user is undergoing a time slice, both his Segment Table and Auxiliary Segment Table are in core. The latter table contains entries in one to one correspondence with the entries in the segment table. In the translation process, if the segment table entry records an invalid status, then a special program interrupt type 16 is generated. It indicates that the corresponding page table is not in core storage. The system supervisor can determine the page table location in auxiliary storage by accessing the Auxiliary Segment Table.

If a user Page Table is in core storage, then so is a corresponding External Page Table. Whenever dynamic address translation references a page table entry recorded as invalid, then a program interrupt type 17 is generated. The virtual memory page being translated does not have a core block assigned. The interrupt causes the supervisor to initiate page turning. After assignment of some available core block to the offending virtual memory page, the supervisor accesses the External Page Table to locate the contents of the virtual memory Page in auxiliary or external storage. The assigned page frame (core block) is then filled with the virtual memory page.

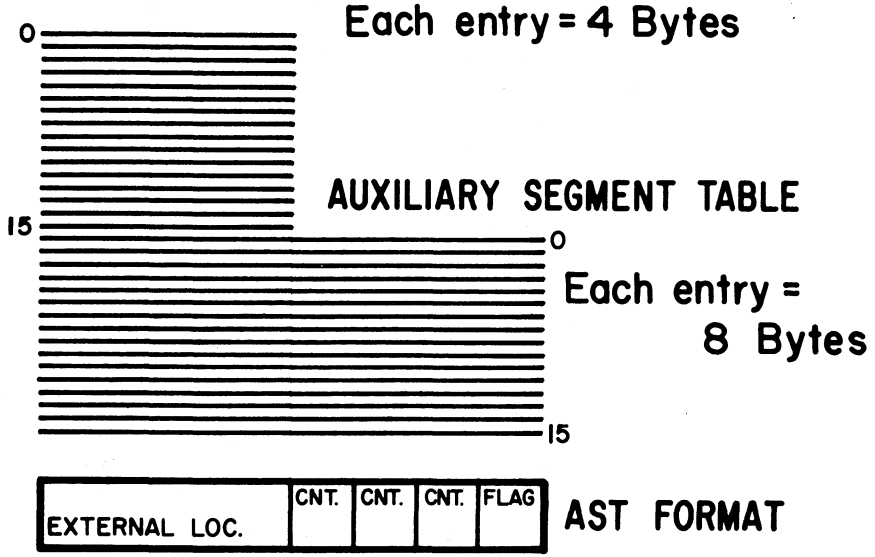
e. Program interrupts 16 and 17 also serve to indicate that the translation process is attempting to index beyond the Segment Table and Page Table respectively. This can take place when a user attempts to make reference outside his "loaded" virtual memory.

f. To speed up the translation process, the Model 67 contains eight associative registers per CPU. These contain for the user currently executing, the segment and page number and corresponding assigned core block number for the eight most recently referenced pages. At the start of dynamic address translation there is a compare in parallel on all eight associative registers. The segment and page number (i.e. left most twenty bits in the virtual memory operand) are the comparand. A match on one of the associative registers causes the core block number to be determined in 150 nanoseconds. At the same time the double table look up as described above is started. This look up goes to completion only if the associative register compare fails to generate a "hit". In this case, if the table look up produces a core block number, this together with the segment and page fields from the virtual memory address replaces the contents of one of the associative registers.

g. There are two basic reasons for using a double index (segment and page fields) in translating to a core block number. One, the entire translation table for a user need not all be in core at one time. The above discussion indicates that it is possible for one or more page tables to be out of core while a user is in the midst of his time slice. Secondly, the double index provides a convenient mechanism for implementing shared code.

SEGMENT TABLE

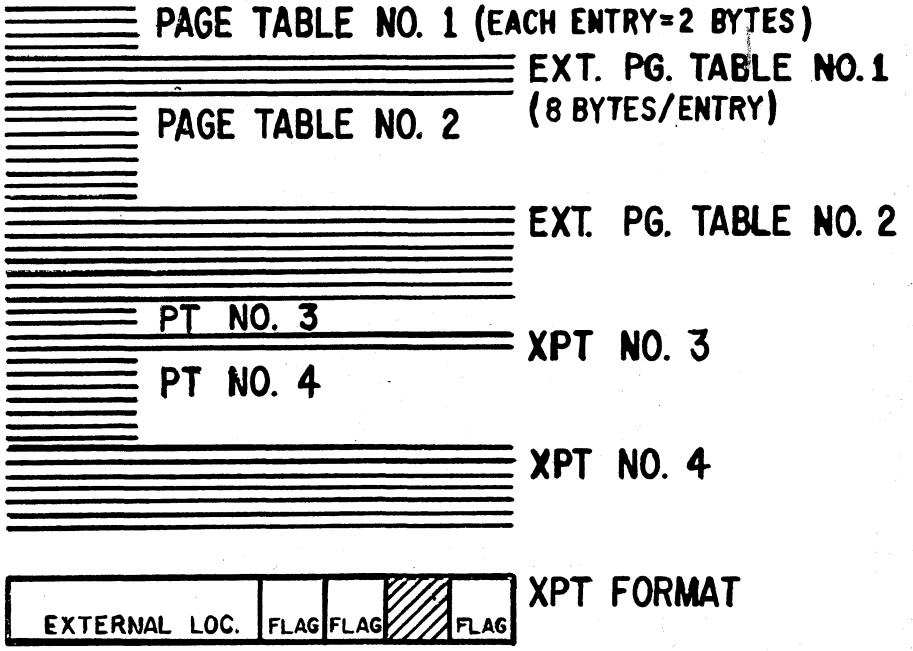
24 BIT ADDRESSING



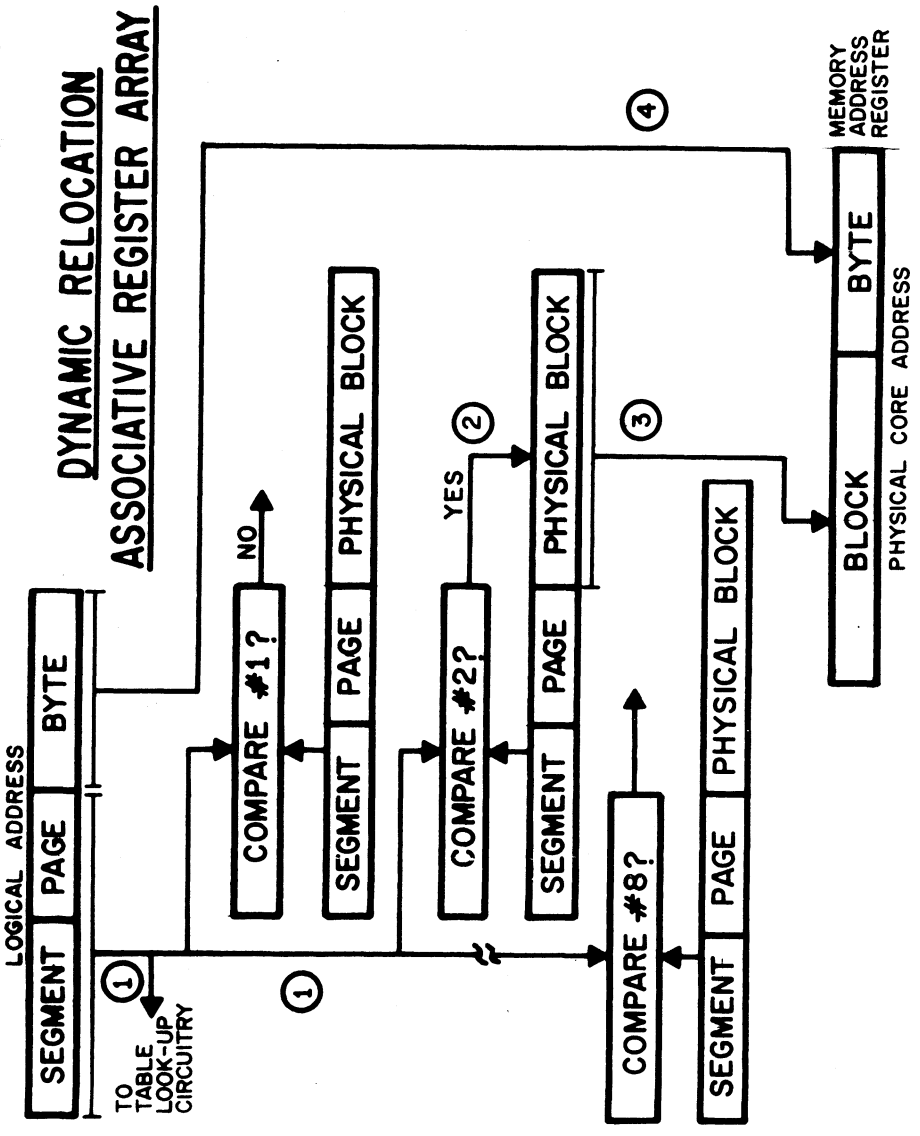
WITH 32 BIT ADDRESSING, THE SEGMENT TABLE CONSISTS OF BLOCKS OF 64 CONTIGUOUS BYTES FOLLOWED IMMEDIATELY BY THE AUXILIARY SEGMENT TABLE CONSISTING OF BLOCKS OF 128 CONTIGUOUS BYTES.

PAGE TABLE

EXTERNAL PAGE TABLE



EACH PAGE TABLE OF 2-BYT ENTRIES IS FOLLOWED IMMEDIATELY BY AN EXTERNAL PAGE TABLE WITH A CORRESPONDING NUMBER OF 8-BYTE ENTRIES.



D. The Multiprocessor 65 System

1. The Multiprocessor Model 65 fulfills all the requirements of a true multi-processing system. Although this system is less elaborate than a fully duplexed Model 67, nevertheless it offers the features one expects of a multi-processor.

   a. Availability arising from a redundancy of components together with error detection and correction capability.

   b. Pooling capabilities ensure:

      1). Better resource management through a single supervisor and job queue
      2). Load leveling
      3). Common program library
      4). Shared I/O and main storage

   c. Increased CPU power

   d. Configuration flexibility through the ability to partition

2. The configuration of an M65 multi-processor is symmetric as shown by the accompanying diagram. Each channel in the system is wedded to a particular CPU. In contrast, the Model 67 Channel Controllers allow any channel to float from CPU to CPU as needed.

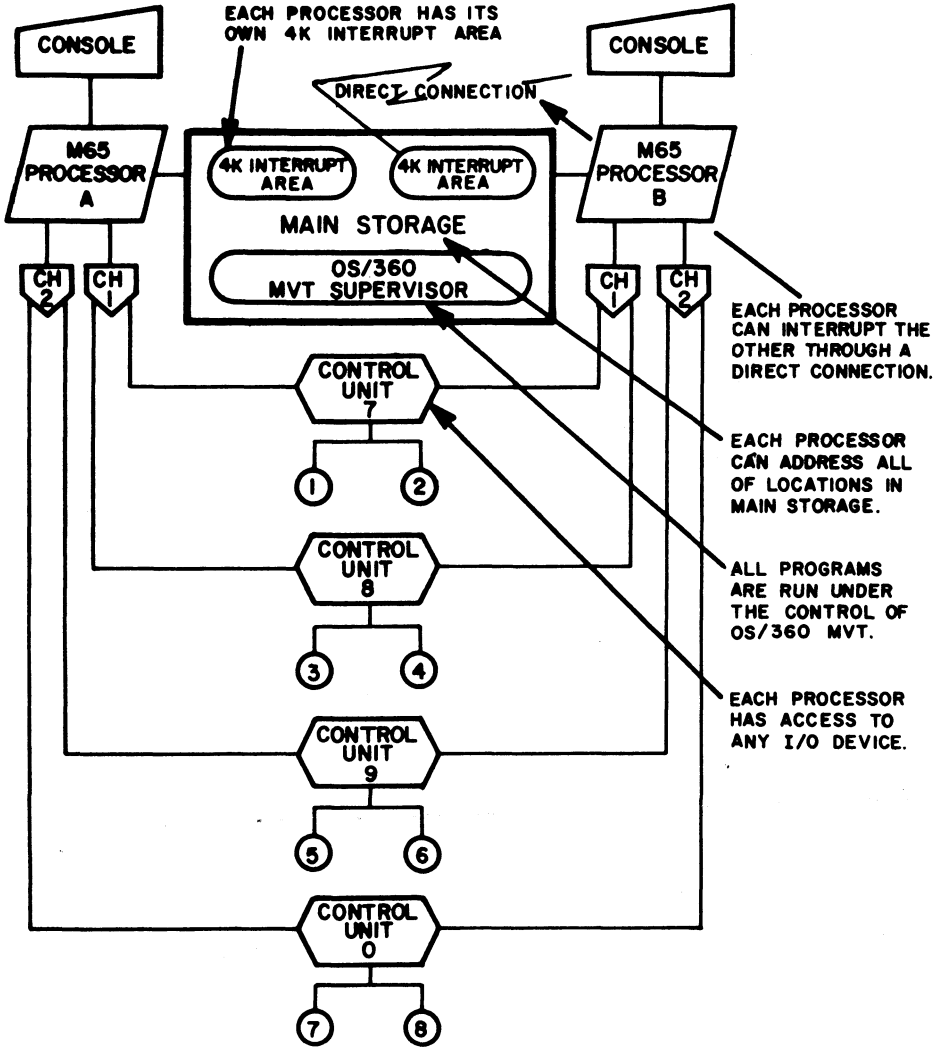
3. Inter-CPU communication is used in three different ways

   a. Task Management under OS-MVT uses this facility to:

      1. Insure that the two highest priority ready tasks are being serviced.
      2. Abend a subtask executing in one CPU when the mother task terminates abnormally in the other.
      3. Exit to a special processing routine when the interrupt is detected by one CPU and the task is being executed by the other.

   b. Alternate Path I/O Control also uses "shoulder tapping" when a path to some device from one CPU is active or unavailable. The supervisor Input-Output System (IOS) can request that the other CPU perform the I/O operation. This function of inter-CPU communication overcomes the inability of any channel to float from CPU to CPU.

M65 MULTIPROCESSING SYSTEM IN MULTIPROCESSING MODE



c. Machine or channel malfunction control is entered when the malfunction alert signal from a failing CPU triggers an external interrupt in the other CPU.

4. Recovery Management Support provides the software needed to implement the "fail soft" capability embodied in the hardware.

   a. The Machine Check Handler allows recovery from many CPU or main storage failures by retrying the failing operation, correcting invalid data, or terminating the failing job.

   b. The Channel Check Handler provides recovery from many intermittent channel malfunctions.

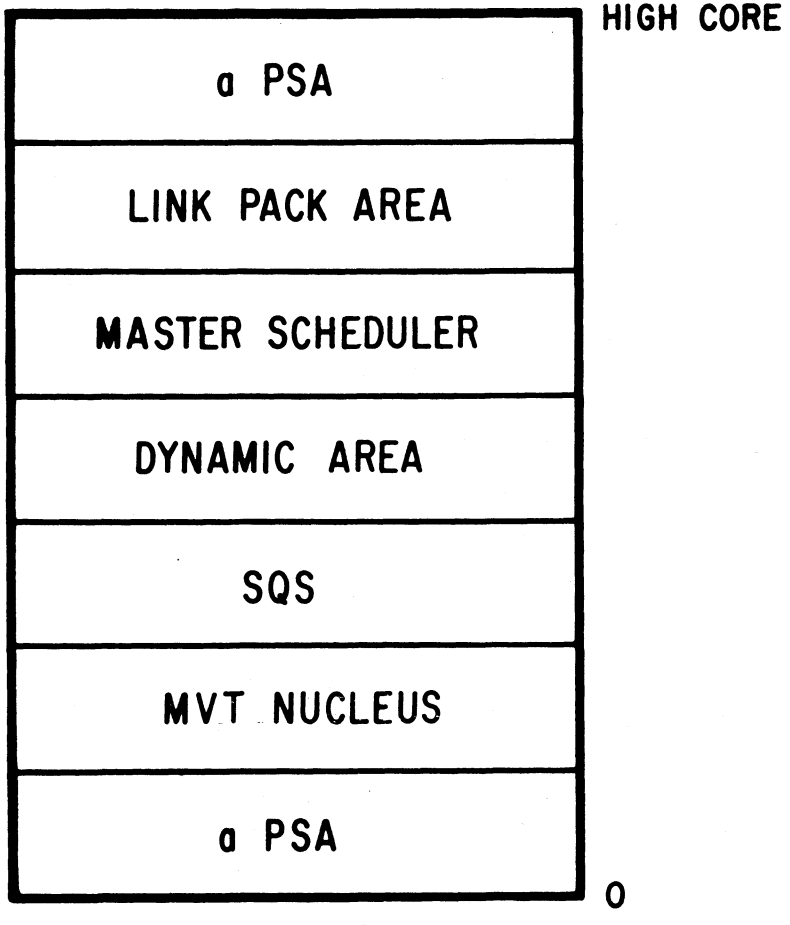
5. Direct Address Relocation

   a. The MP65 system uses two Permanent Storage Areas (PSA s). One occupies the area encompassing addresses 0 through 4095 and the other lies in the uppermost 4K bytes of core storage.

   b. There is a toggle switch for each CPU on the Configuration Control Panel. The setting of the switch controls whether the corresponding CPU prefixes or not. The non-prefixing CPU owns the PSA in low core.

   c. The CPU which undergoes prefixing owns the high core PSA. Prefixing implies reverse prefixing as explained below. When the prefixing CPU generates an address in the range 0 to 4095, hardware monitoring detects that the high order twelve bits are zero. This address is mapped into a corresponding address in the high PSA. Thus the prefixing CPU references its own "low core" area. On the other hand when the prefixing CPU generates an address in the range spanned by the uppermost 4K bytes of core storage in the system, reverse prefixing occurs. This address is mapped by hardware onto a corresponding one in the true low core area of the system. In this way a prefixing CPU can access the PSA of the alternate CPU.

MP65 STORAGE AREA



V. Design of a Virtual Memory Based Programming System

A. Introduction:

1. Is the design of an operating system which controls many users, each with their own virtual memory, like that of a conventional system? Or is the design fundamentally different? The answer lies below.

2. In a conventional system, core storage is divided into two parts, each representing a separate level of control. Level one provides a supervisor containing interrupt handling routines, Input-Output Subsystem (IOS), task management services and so forth. In level two, there resides at execution time user written programs, language processors, or utility routines. Control passes from level two to level one by means of an interrupt.

3. The interruption is a request by the program in level two for service on the part of the supervisor in level one. Even in a multi-programming environment the design of the operating system is still two level in nature. The overall user area may contain more than one program competing for use of the CPU. Nevertheless, when any given program is executing, the supervisor in level one is the sole support for that program. It therefore provides all the requisite services needed to maintain any program executing in level two.

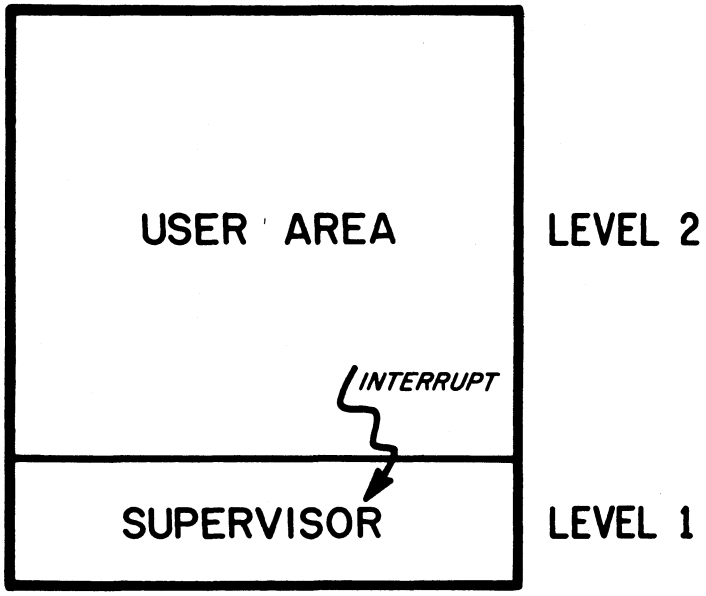
4. A virtual memory based programming system, like Caesar's Gaul, is divided into three parts. That is to say, it is a tripartite structure in contrast to a conventional system which is two level only.

B. Contemplation of the Trinity

1. Introduction:

a. In a virtual memory based programming system, there are two storage areas to be considered -- core storage and the virtual memory belonging to some user of the system. Core storage is divided into standard size units called core blocks. Each user virtual memory is divided into units called pages.

CORE STORAGE MAP



A CONVENTIONAL PROGRAMMING SYSTEM

b. Within core storage some core blocks are permanently assigned to the system supervisor. This supervisor constitutes level one in the tripartite structure. The remaining core blocks represent a resource of core storage available for allocation to the multi-users of the system or even to the supervisor itself. Typically a core block is assigned to some user as the result of a paging demand. Occasionally the supervisor requires a block to fashion temporary control blocks.

c. Each user's virtual memory is divided into two basic areas, each constituting another level of control within the overall programming system. Level two contains a control program designed to provide the usual program support services. Level three contains user written programs, language processors, and utilities. Note that each virtual memory presents to its user the appearance of a conventional (two level) programming system. Indeed, control passes from level three (the user area in virtual memory) to level two (the control program within virtual memory) by means of an interrupt - a virtual interrupt.

d. The extended PSW contains a bit to control the supervisor/problem state and a separate bit to control the non-relocated (D. A. T. unit inactive)/relocated mode of operation. In the two IBM programming systems which exploit Model 67 technology, level one is assigned the supervisor state and non-relocated mode of operation. Levels two and three are assigned the problem state and relocated mode. This choice is purely one of software design.

2. Levels One and Two Compared

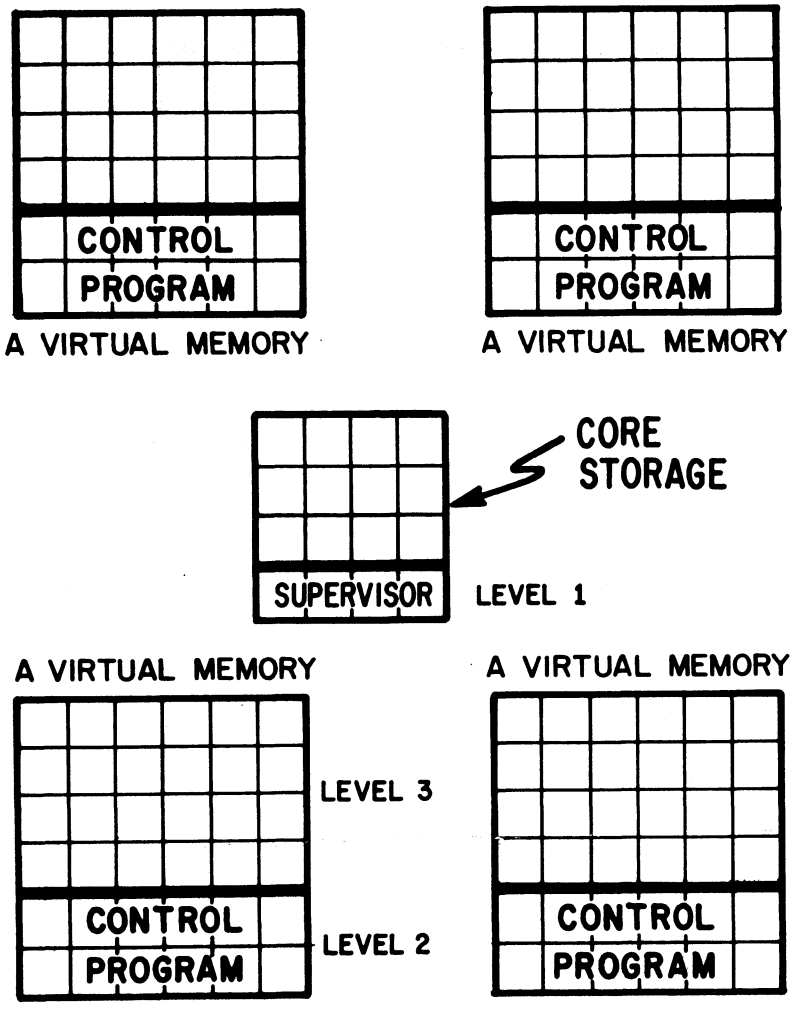
a. The overall programming system contains two control programs. There is a system oriented supervisor resident in core storage. In addition, there is a task oriented supervisor resident in each user's virtual memory. In what ways are these alike or different?

b.

Level One Characteristics	Level Two Characteristics
• one per system	• one per user
• supervisor state	• problem state
• non-relocated mode	• relocated mode
• not time sliced	• time sliced
• permanently core resident	• pageable

c. All characteristics listed above for level two

A VIRTUAL-MEMORY-BASED PROGRAMMING SYSTEM



hold as well for level three. But the control program in level two must be given a degree of privilege denied to level three. Since both levels run in the problem state, this "privilege" must be implemented by software. The method used depends upon the particular programming system in question. In any case, only modules executing in level two are privileged to make certain service requests to the system supervisor in level one. Such requests are monitored by level one to ascertain that they were issued from level two within the virtual system.

d. Level One Functions

1. Field, analyze, queue hardware interrupts.
2. Process system oriented interrupts
   - paging-in
   - paging-out
   - I/O requests
3. Operate the system recovery nucleus and inter-CPU communication routine.

e. Level Two Functions

All functions not specifically undertaken by the level one supervisor default to the virtual system.

3. Interrupt Handling Examples

a. Floating Point Overflow

This program interrupt is recognized by level one as germane to the originating task only. Therefore, the interrupt is reflected back to the task as a virtual program interrupt. Level one merely fields the hardware interrupt and queues it on the task for processing within the virtual system.

b. Timer Interrupt

If the interrupt results from a time slice end for some task, processing is handled completely within level one. Unchanged core blocks are made available immediately and changed blocks are queued for writing to auxiliary storage. Upon receipt of the subsequent I/O interrupt, these latter blocks are also made available as a system resource.

On the other hand, if the interrupt represents a user timer interrupt, then level one merely gives the relevant task a pending virtual timer interrupt. All processing in this case is done at a task (virtual system) level.

c. I/O Interrupt From a User Device

A user read or write request emanates from an access method within a task, but a Start I/O instruction executed in level one implements the request. The subsequent I/O interrupt is processed on two levels. The initial processing of the I/O interrupt takes place in level one. The remaining processing is handed over to the task as a pending virtual I/O interrupt.

4. The Virtual Interrupt

a. When a hardware interrupt occurs, the current PSW is stored as an old PSW and the corresponding new PSW is made the current. This interrupt sequence is driven by a micro-program resident in Read Only Storage. This latter entity can be thought of as constituting a level zero in support of level 1.

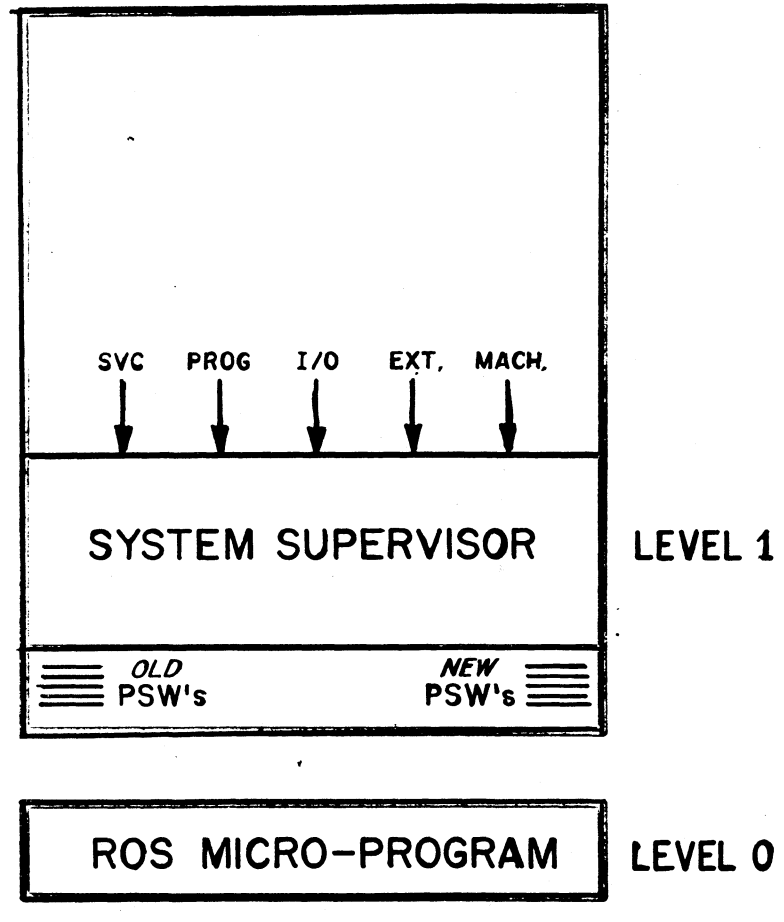
b. The new PSW contains the location of an entry point in level one. Therefore, any hardware interrupt causes control to pass initially to the system supervisor. The interrupt may possibly be fully processed at this level. However, if level one determines that some or all processing is to take place in the virtual system, then at this time the virtual system is given a "pending" interrupt. That is to say, the control block representing this processing is queued on the task as a future interrupt of one of the types defined for the virtual system.

c. At the time of its dispatching, the task "takes" one of its pending interrupts. In this way, task oriented interrupts are fielded by the system supervisor and reflected back to the virtual system. All of this is transparent to the user. As far as he is concerned, his control program has fielded an interrupt originating from some point in his virtual memory.

d. The virtual interrupt procedure is completely software implemented. Each virtual memory contains a "low core" area in which are stored both old and new virtual PSW's. These need not agree in number and type with those associated with the hardware of the real system. In addition, each task possesses a current PSW. This is nothing more than the old PSW stored by the real interrupt mechanism at the instant the task last "lost the CPU."

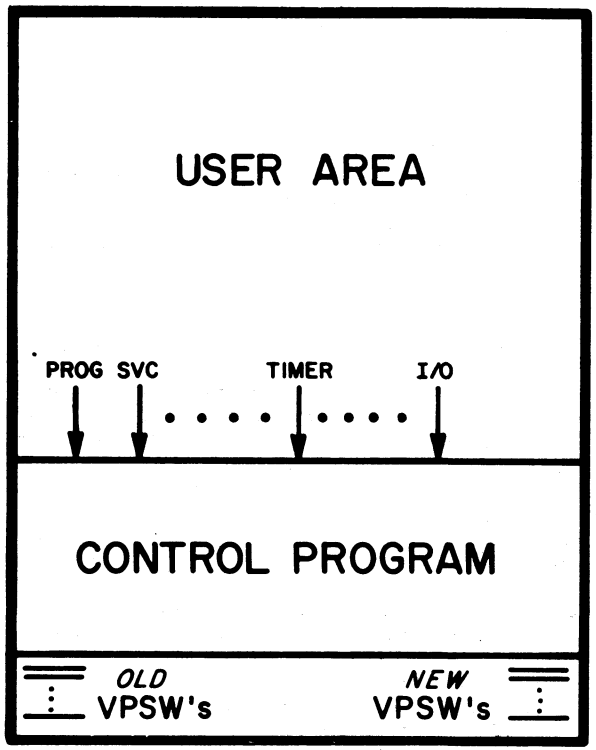
e. Recall that a micro-program in level zero causes the real PSW's in level one to be swapped when a CPU takes an interrupt. By analogy, a routine in level one swaps the virtual PSW's in level two when the corresponding task receives a virtual interrupt.

CORE STORAGE MAP



REAL INTERRUPTS

VIRTUAL MEMORY MAP



VIRTUAL INTERRUPTS

This routine is entered just after the task has been selected by the scheduling algorithm to receive a CPU. The routine checks the task for unmasked pending interrupts. If one or more exist, the highest priority such interrupt is taken. The current PSW for the task is stored as an old virtual PSW and the corresponding new virtual PSW is made the current PSW.

f. Each new virtual PSW contains the location of an entry point in the control program comprising level two of a virtual system. When a task is dispatched, its current PSW is loaded into the hardware register of the machine. If a task interrupt has been taken, then execution within the virtual system commences at an entry point in the level two control program. To the user, control has passed to one of the defined entry points in an interrupt driven control program in the lower portion of his virtual memory.

C. The Virtual Machine Concept

1. Introduction

a. The IBM Model 67 TSS provides each user with a virtual memory containing a standard control program. At the time of log-on, the newly defined task comes provided with translation tables containing predefined entries to establish addressability for this standard control program.

b. This control program can be modified by any user to provide linkages to user written interrupt handling routines. Such user written routines handle program interrupts caused by arithmetic overflow and underflow conditions, user written SVC's and user timer interrupts.

c. These linkages, needed to "boost" the task interrupt from level two to level three, are inserted into the level two control program by means of macro instructions. These macros are issued by the user and represent a means of tailoring the standard control program to suit his individual needs.

2. The Virtual Machine

a. CP67 is another programming system based on the Model 67 technology. It was developed as an experimental system. by the IBM Cambridge Scientific Center. It provides a system supervisor resident in core storage and a separate virtual memory for each user of the system. However, at

log-on time, the system sets up a virtual machine for the user based on his user identification. The virtual machine is a functional simulation of a real computer and its associated I/O devices. It is indistinguishable from a real system but is really one that the level one supervisor is managing. This supervisor allocates the resources of the real machine to each virtual machine, in turn, for a short time slice.

b. Since virtual machines are simulated, their configurations differ from each other and from real machines. Regardless of the configurations, each user controls his virtual machine from his terminal, which is, effectively, his console key-board.

c. Virtual machines, in common with real machines, will operate most efficiently under an operating system. One user may choose to "load" OS into his virtual memory, whereas another may decide in favor of DOS. In general, any system which does not contain either time dependent code or self-modifying I/O sequences can be used. In each such case, the user terminal becomes the operator console for the system loaded.

d. Those users of the CP67 system who wish to engage in conversational computing must "IPL" CMS into their respective virtual memories. This system, the Cambridge Monitor System, like OS and DOS, is a stand-alone system. In addition, it is terminal oriented, and conversational, designed to handle a single user at his terminal.

VI. Remote Batch Computing

A. Background

1. The need for remote access to a computer facility is not a new requirement. The need has existed from the time of the earliest first-generation computer installations. For some fifteen years various techniques have been employed to make the computer power available to remotely-located users. Program decks and data have been mailed or transported to the computer center by every conceivable method. Likewise, computer output has been returned to the remote user by a wide range of delivery schemes--some informal and formal, internal and external, but generally slow.

2. As soon as remote access to a computer is established, the requirement follows to improve or speed up the access. The need to improve the service to the remote user by reducing his turnaround time becomes immediately apparent. The time from problem submission to receipt of results must be collapsed. Service to the user must be timely in order to enhance problem-solving capability.

3. It was natural and logical early to consider communication facilities to help solve the remote input of work and output of results. The first steps were off-line operations--teletype-to-teletype, card-to-card, and eventually tape-to-tape. Program decks and data were not physically transported to the computer center but transmitted over communication facilities and recreated at the receiving end in paper tape, cards, or magnetic tape. Later development permitted remote terminals to go on-line to the host computing system. Program decks and data were inputted directly into the computing system for subsequent processing.

4. Current operating systems that provide multiprogramming facility are particularly suited to remote access processing. A partition or region can be effectively used to interface remote terminals. The jobs originating from remote stations are collected and fed into the traditional local or on-site batch processing.

B. Remote Batch Entry Compared to "Time-sharing"

1. Terminal-based computing has been divided into two categories in this document. The first category is interactive or conversational computing, the generally accepted definition of "time-sharing." The second category is remote batch. Both categories can also be considered as "convenience computing" since the user benefits by having improved access to the computing facility.

2. In interactive computing, i.e., "time-sharing," the user or problem-solver uses a key-driven terminal to communicate directly with the system. Although the degree of interaction will vary with the implementation technique, the user will engage in a continuous dialogue with the system. Responses from the system are predictable and may be on a line-by-line basis.

3. Remote batch in its simplest form allows submission of the normal preplanned job deck to the batch processing system by communication facilities. It eliminates the need to deliver the program physically to the computing center.

4. Remote batch, again in simplest form, responds only at the completion of the processing requirements. It does not provide the interactive environment of "time-sharing." Errors in the preplanned job deck are eliminated by the normal repetitive runs inherent in batch processing.

5. Whereas the response from a "time-sharing" system is predictable, such is not the case with remote batch entry. The program is entered as an entity and is run to completion. The turn-around time or response is a function of the system loading, job priority, and maybe the whim of the computer operator.

6. A low-speed job entry system where the user types input at a keyboard may have some of the features of an interactive system. Selective program modification allows an RJE user to change or edit a data set previously submitted. The entire program is not resubmitted when alterations are required. Line-by-line syntax analysis can also be incorporated to make a RJE more conversational.

7. A sophisticated remote batch entry system can approach the capability of a simple "time-sharing" system. The distinction between the two types of terminal computing methods becomes less clear as the features of low-speed batch input are increased. Examples of such features would be on-line program modification, line-by-line syntax analysis, and "desk calculator" capability.

C. Design Considerations in Remote Batch Systems

1. The response from a remote batch system (or job turn-around time) depends upon system design. A complete remote batch computing system allows remote job entry (RJE) and remote job output (RJO). A limited system would allow RJE but require output--listings, dumps, etc., to be delivered to remote locations. Low-speed terminals in particular might be used for RJE but are too slow for output printing loads. Selective printing is sometimes implemented to provide limited RJO at the low speed, i.e., key-driven terminals.

2. Remote batch systems generally employ the higher-speed terminals, such as the IBM 2780 and IBM 1130, Model 20, Model 25, and up, to provide both RJE and RJO. The user or problem-solver does not use a key-driven device to deal directly with the system. His job is just one of the batch of job decks to be submitted at the remote terminal.

3. One approach to integrating remote batch processing into the central processing system is simply a job collection method. Remote jobs are accumulated into a special queue on direct access storage allocated for this purpose. The system operator is advised as to the status of the queue as jobs are entered. The jobs collected from the remote terminal are processed when the system operator takes the necessary steps to switch to this special queue. Scheduling of the remote jobs is dependent upon operator intervention. Response or job turn-around time may be highly variable and unpredictable.

4. Automatic merging of remote jobs into the batch stream provides significant improvement over the job collection method. Using a standard priority scheme for both remote and local batch jobs, an integrated job queue is created. Remote jobs are not segregated but are processed on the basis of individual job priority. The turn-around time for remote jobs is equivalent to the on-site batch performance.

5. Remotely submitted jobs are subject to the same fate as any job submitted to the batch processing system. An error--trivial or otherwise--will frequently flush the job out of the system. It is desirable to eliminate such errors in any system. Remote batch provides potential processing points to reduce errors in JCL and source language programs.

6. If the remote station is an intelligent terminal, processing can be incorporated at the remote location to detect errors in the input job decks. Processing at the remote station can also be used for data compaction, i.e., the elimination of blanks in the input job decks. This technique permits efficient transmission where communication costs are significant. Obviously, the central station has the additional burden of converting the input back to the original job deck input.

7. The remote batch processing of the central station can also incorporate preprocessing to catch errors prior to entry into the "batch." Syntax analysis for JOB Control Language and programming languages can be performed on each line as received at the central station. This technique makes sense in low-speed batch entry where a user is entering a job directly from a terminal keyboard. Format errors are detected and indicated to the user who corrects them immediately.

8. A desk calculator capability is also a likely feature when the user types input at a low-speed terminal. The user can enter arithmetic expressions for immediate evaluation. The terminal is used as a desk calculator in this mode of operation to perform one-time arithmetic computations. A third feature that can be provided is a message facility. Remote stations can send messages to the central station operator and other remote station-users.

9. The size of the partition or region that supports the remote batch system is a function of the number and variety of terminals and the features provided. Obviously, syntax analysis, "desk calculator" mode, and other features require additional core storage over and above the terminal I/O buffers and access method. The trade-off between space required and function must be considered in the evaluation of remote batch entry systems.

VII. Communications for Terminal Based Computing

A. Introduction

1. Terminal based computing encompasses both "time-sharing", i.e. interactive or conversational computing, and non-interactive remote computing. "Time-sharing" generally involves the use of low-speed or key driven terminals while remote computing spans the full spectrum of communications capability from the low-speed to the highest speeds available. Hence, Communications for Terminal Based Computing is a very broad subject covering the full range of communications facilities. These notes are provided as an introduction to the subject. They rely heavily on the definitions as given in the IBM Computer Description Manuals. The first section deals with the definition of the key words used in the communications aspect of terminal based computing.

2. What makes a terminal based computing system different from other on-line systems?

Terminal based computing systems are intended to give improved service to the user--manifested in immediate access, rapid response or minimum turn-around time. A time-sharing system will have a specific number of connections available to the users of the system. At any point in time, the number of terminal users on the system is limited to the number of connections or lines into the system. Some users or terminal locations may have dedicated lines. They have guaranteed access into the system. Other users must contend for the remaining lines or connections. Obviously, a small number of connections cannot serve a large population of active users without excessive delays. Continuous dialing with busy signals mean frustration and this defeats the intent of time-sharing systems.

Time-sharing, particularly, is based on the concept of one terminal per line. One access to the system is dedicated to one user when the connection is made. Multi-drop connections are not consistent with this philosophy. There should be no controlled sharing of the line via addressing or polling. There should be no time out. The terminal must remain active.

In addition, the time-sharing enviroment requires full interactive capability. The user must be able to signal the CPU while in either send or receive mode. He must have the ability to abort execution, halt output, or cancel familiar messages. In turn, the system should have the ability to interrupt the user when required. Rapid line-turnaround from receive to transmit and vice versa is necessary for rapid conversational computing.

Time-sharing systems generally permit the user greater capability at the terminal compared to other on-line systems such as inquiry. Program creation and modification are basic functions of a time-sharing system. This greater capability could be used either ignorantly or maliciously to injure the system. Hence, system integrity must be assured in spite of the user's capability. The security of program libraries and data sets must be maintained as in any on-line system.

Telephone communication networks are designed to satisfy the demands of normal telephone calls. Switching networks are designed to handle only a small percentage of subscribers at any one time--say 20 percent for instance with an average call considered to be on the order of two minutes. In contrast, a user terminal session on a time-sharing system has an unpredictable length. A typical session may tie up a telephone circuit for hours. Thus the "holding time" per call makes terminal based computing a different ball game. A number of lines into a time-sharing system can seriously impact the loading of a telephone switching exchange and saturate the facilities. It is critical that the telephone company be advised as soon as possible about the potential loading caused by terminal based systems.

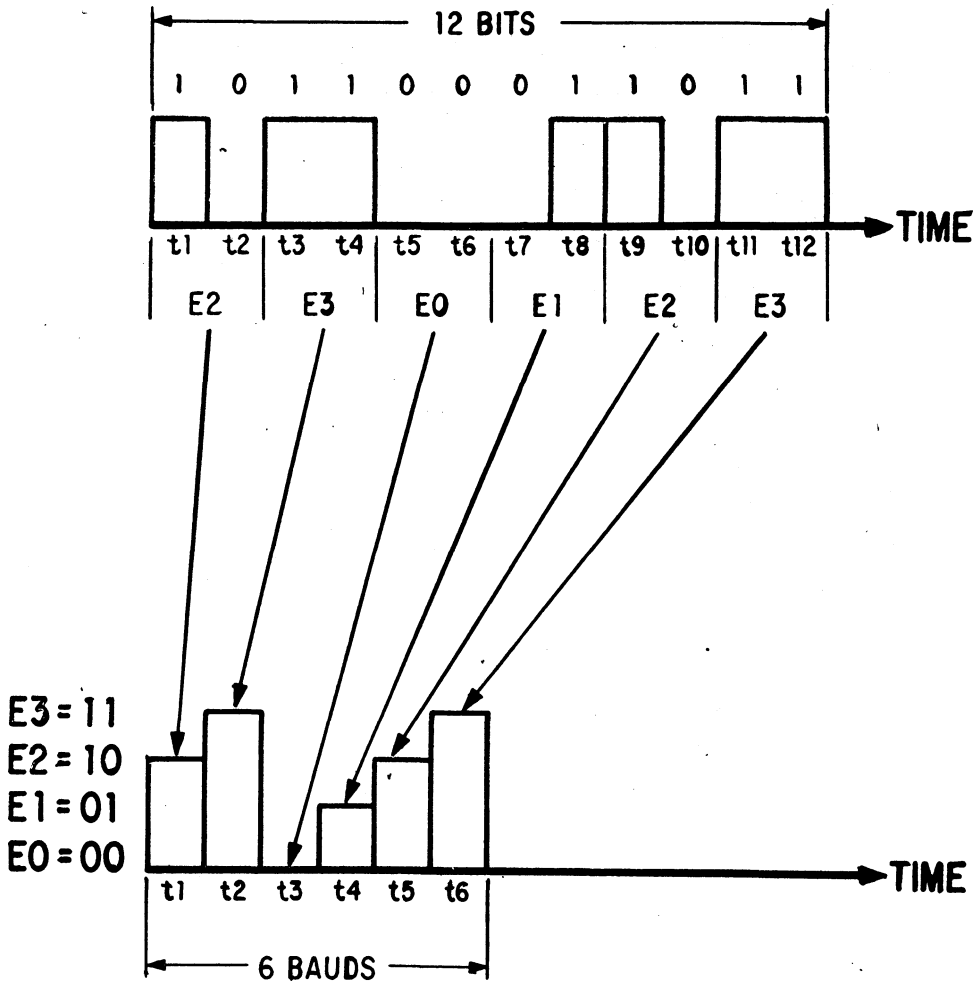
B. Telecommunications Definitions

Key words used in the communications side of time-sharing will be discussed in this section.

1. Terminals A terminal refers to any device or collection of components that is capable of sending and/or receiving information over a communication channel. In a time-sharing enviroment, a terminal does not necessarily have to be connected to a communication channel. However, it must have the capability of such a connection.

A method for classifying the type of terminal support offered by a time-sharing system is in relation to the speed of transmission of that terminal. The unit of signaling speed is called a Baud. "The speed in Bauds is the number of discrete conditions or signal events per second. If each signal represents only one bit condition. Baud is the same as Bits per second."(1) The classes of transmission speeds are low, medium, and high. Low speed is defined as "usually, data

BAUDS vs. BITS



transmission speed of 600 bits per second or less", and medium speed as "usually data transmission speed between 600 bps and the limit of a voice grade's facility". (2) High speed transmission is not defined in the Glossary.

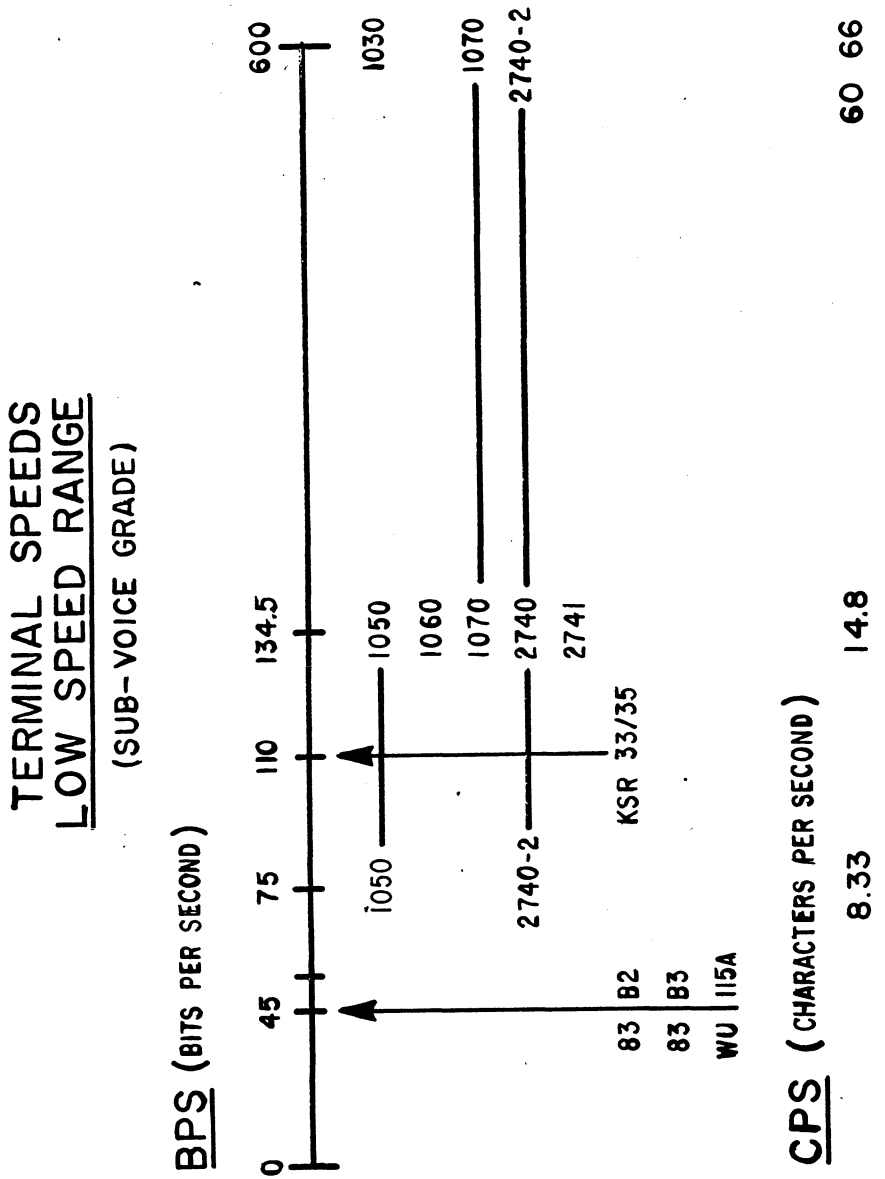
Because there does not seem to be an industry standard of what is meant by the terms low, medium and high speed when used in connection with data transmission, the time-sharing system portion of the CDM will use the term "low speed" terminal and "high speed" terminal in the following manner.

a. Low speed terminals will be the class of all keyboard-driven terminals. Other devices such as paper tape readers and card readers may be attached to the keyboard-driven, device, but if the terminal may be keyboard-driven, it will classed as a low speed terminal. If the terminal is a computer system or a group of components which do not possess the property of being keyboard-driven, this terminal will be called

b. a high speed terminal. Therefore, the two definitions are user oriented rather than transmission speed oriented. In a low speed environment, a user may elect to communicate with a system via a keyboard whereas in a high speed environment, the only method of communication the user has are through devices such as tapes and card readers. (When analyzing individual terminals, it will be found that most keyboard-driven devices transmit at less than 300 BAUD, and most high speed terminals at more than 600 BAUD.)

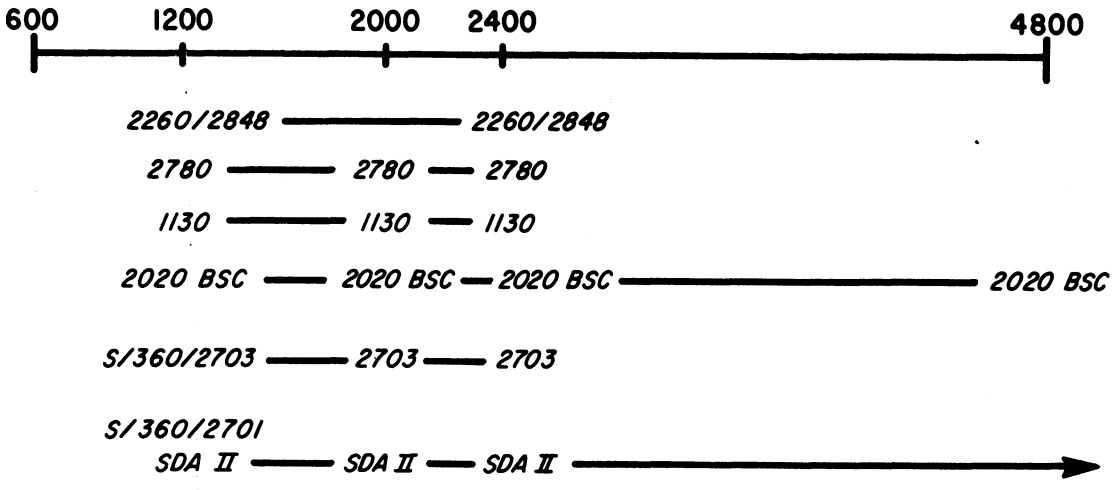
c. Remote/Local A local terminal is a terminal which is connected directly to the computing system via its con-control unit. A remote terminal is a terminal which is connected to the computing system via communication lines. The method of connection, and not the type of terminal denotes the difference. For example, the IBM RAX System supports local and remote IBM 1050 terminals, but only local IBM 2260's.

d. Home Loop/Line Loop Operation A home loop operation is one which involves only those input/output units associated with the local terminal. When local terminal is in home loop operations, it is said to be "off-line". A line loop operation takes place over a communication line between the input units at one terminal and the output units of another terminal. In both of these modes of operations, the resources of the central computer system are used.



TERMINAL SPEEDS
MEDIUM SPEED RANGE
(VOICE GRADE)

BPS (BITS PER SECOND)

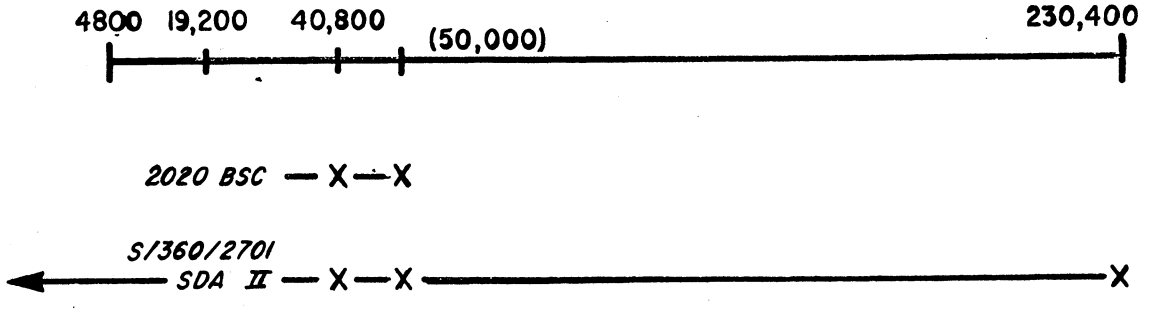


CPS (CHARACTERS PER SECOND)

150 250 300

TERMINAL SPEEDS
HIGH SPEED RANGE
(MULTI-VOICE GRADE)

BPS (BITS PER SECOND)



CPS (CHARACTERS PER SECOND)

2400 5100 6250 28,800

2. Facilities

a. Simplex, Half-Duplex (HDX), Full-Duplex Lines (FDX)
A simplex line is a circuit capable of one-way operations only. Communication may proceed in one direction only, with no capability for reversing the direction. A half-duplexed (HDX) circuit or line is capable of transmitting and receiving in both directions, but is not capable of simultaneous and independent transmission and reception. It cannot sent and receive at the same time. A Full Duplexed (FDX) line or curcuit is capable of the simultaneous and independent transmission and reception between two points -- in both directions.

b. A "private" or leased-line network is a network reserved for the exclusive use of one customer. A "public" network is a network provided by a common carrier for use by many customers. "Dialup" refers to the use of a dial or push-button telephone to initiate a station-to-station telephone call. The IBM RAX system provides the capability of handling either leased or dialup connections between the terminal and the CPU.

c. Telpak: Broadband communication channels for transmitting data at rates greater than 60,000 characters.

Voice-grade: a circuit of sufficient bandwith to permit a data to transfer rate up to 2,400 bits per second, generally with a frequency range of 300 to 3,000 cycles per second.

WATS: Wide Area Telephone Service. "A service provided by telephone companies which permit a customer by use of an access line to make calls to telephones in a specific zone on a dial basis for a flat monthly charge. Monthly charges are based on the size of the area in which the calls are placed, not on the number or length of calls. Under the WATS arrangement, the U.S. is divided into six zones to be called on a fulltime or measured-time basis. (2)

d. Single-drop/Multidrop Lines In time-sharing systems, the word "station" and the word "terminal" may be used synonomously. A drop refers to a station or terminal, and a single-drop system is a system in which only one terminal or station is connected to each communications line. A multidrop system is one in which two or more stations or terminals are connected to a single line. Both the programming and the hardware features necessary to implement these two types of systems are quite different. In a system which has only single-drop facilities such as the IBM RAX System, the

RAX program need only concern itself with processing information from one terminal per line. It need not sort out transmitted information, or direct information over a single line to different terminals or stations on that line. On the other hand, a multidrop system, such as the IBM CCAP Message Switching System, must determine which terminal on the line sent a message, and/or which terminal on a line will receive a particular message. Most problem-solving time-sharing systems of today have single-drop capabilities only.

3. Modulation-Demodulation Techniques Modulation is the "process by which some characteristic of an electrical wave is varied in accordance with another wave or signal."(1) In Data Processing, modulation is used to make business machine signals compatible with communications facilities. In a specific case, it is the conversion from digital signals to audio signals for transmission over communication lines. Demodulation would then be the conversion from audio signals from the communication lines to digital signals for business machines. A data set is a device which performs this modulation-demodulation and the control functions necessary to provide compatibility between business machines and the communication facilities. Another word used for this device is modem, which is the contraction of modulator-demodulator.

4. Transmission Techniques

a. Asnychronous Transmission: In this type of transmission each information character is individually synchronized and is normally preceded by a "a start signal, which serves to prepare the receiving mechanism for the reception and registration of a character, and is followed by a stop signal, which serves to bring the receiving mechanism to rest in preparation for the reception of the next character."

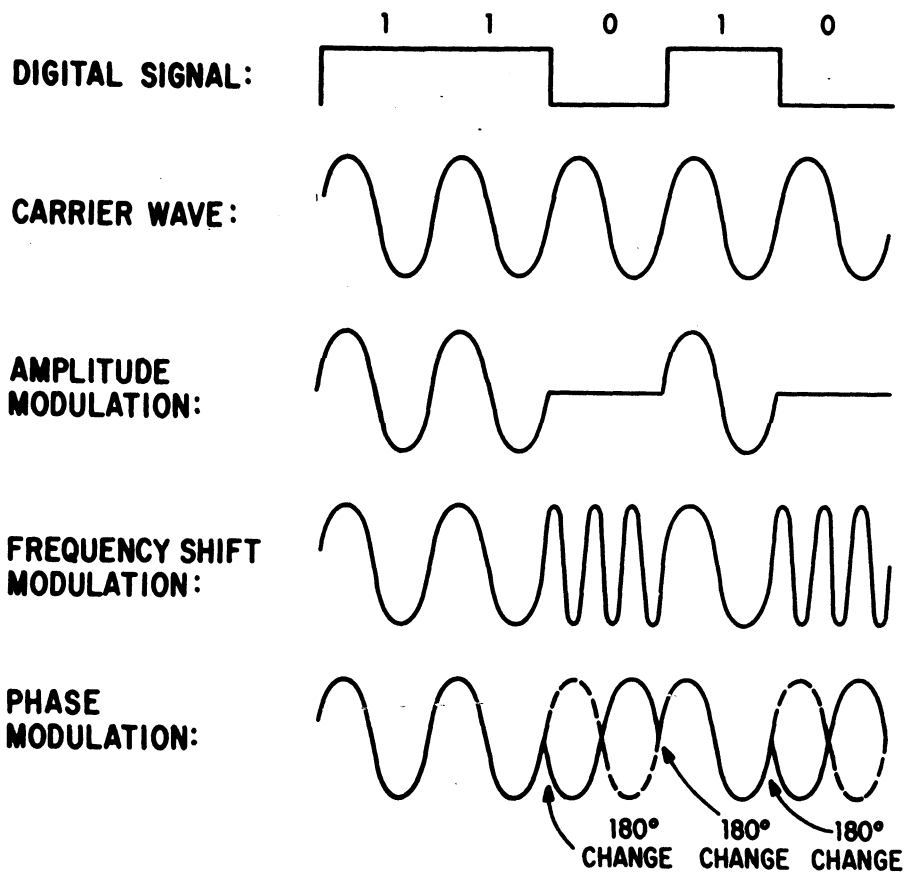
"Synchronizing each transmission independently permits multipoint line operation, with each station sending with its own bit and character phase."

b. Synchronous: "having a constant time interval between successive bits or characters. The term implies that all equipment in a system is in step."

---

1. Op. Cit., C20-1666-1, page 15.

MODULATION TECHNIQUES



Synchronous transmission is when the sending and receiving devices are operating at the same frequency, and are maintained in step with each other. Bit and character synchronization are established at the beginning of each transmission by transmitting a synchronizing pattern. This synchronization (sometimes called character phase) is then maintained until the transmission stops, signalled by line control characters. There are no "stop" and "start" bits. The synchronization is controlled by a clock located either within the station or the modem.

1) One group of IBM terminals use the fixed-count, four of eight code in synchronous mode. These terminals are called STR for Synchronous Transmitter/Receiver. The IBM 1013, 7702, 1978, 1130, and S/360 Model 20 are included in this class.

2) Binary Synchronous Communications (BSC)
BSC transmission or BISYNC provides for the transmission of a variety of codes in the synchronous mode. EBCDIC USASC II, Six-Bit Trancode are transmitted serial by character and serial by bit over half duplex lines. A transparency feature permits transmission of any combination of binary bits. The IBM 2780, 1130, S/360 Model 20 and any S/360 with the appropriate 2701 or 2703 are BSC devices.

C. Modems

1. Data processing equipment is connected to a communications network by the use of a device that goes by many names, including modulator and demodulator unit, mod/demod/ modem, subset, and data set. Certain trade names such as AT&T's Data-Phone (R), are also used. Whatever the name of the device, its purpose is to provide an "interface" or common boundary between data processing equipment and communications equipment. In this publication, the term "modem" is used.

Modems vary considerably according to the types of networks, data rates, and forms of signalling employed. However, they all are designed to perform conversions between the binary signals of business machines and the transmission frequencies of communications equipment.

2. Classes of Modems

Model	Range	Type	Timing
100's	Low Speed	Serial	Start-Stop
200's	Medium Speed	Serial	Synchronous
300's	High Speed	Serial	Synchronous
400's	Low & Medium Speed	Parallel	-
500's	High Speed	Parallel	-
600's	Voice Band	Analog	-
700's	Wide Band	Analog	-
800's	Auto-Dial Units		

Western Union model numbers are usually four digits but the first signifies range just as numbers shown above.

3. Rates for Modems

103		$25/Month		
201	2400B	$70/Month		
202	1200B	$40/Month		
301	40.8KB	$250/Month		
401A	TRANS	20 Char./Sec.	$5/Month	16 Char.
401B	RECVR	20 Char./Sec.	$40/Month	16 Char.
401E	TRANS	20 Char./Sec.	$7/Month	99 Char.
401F	RECVR	20 Char./Sec.	$30/Month	99 Char.
500's	Nothing Available Yet			
602A	T/R Low Speed Fax.		$30/Month	
700's	Nothing Available			
801	10 Pulses/Sec.	No Prices Available		

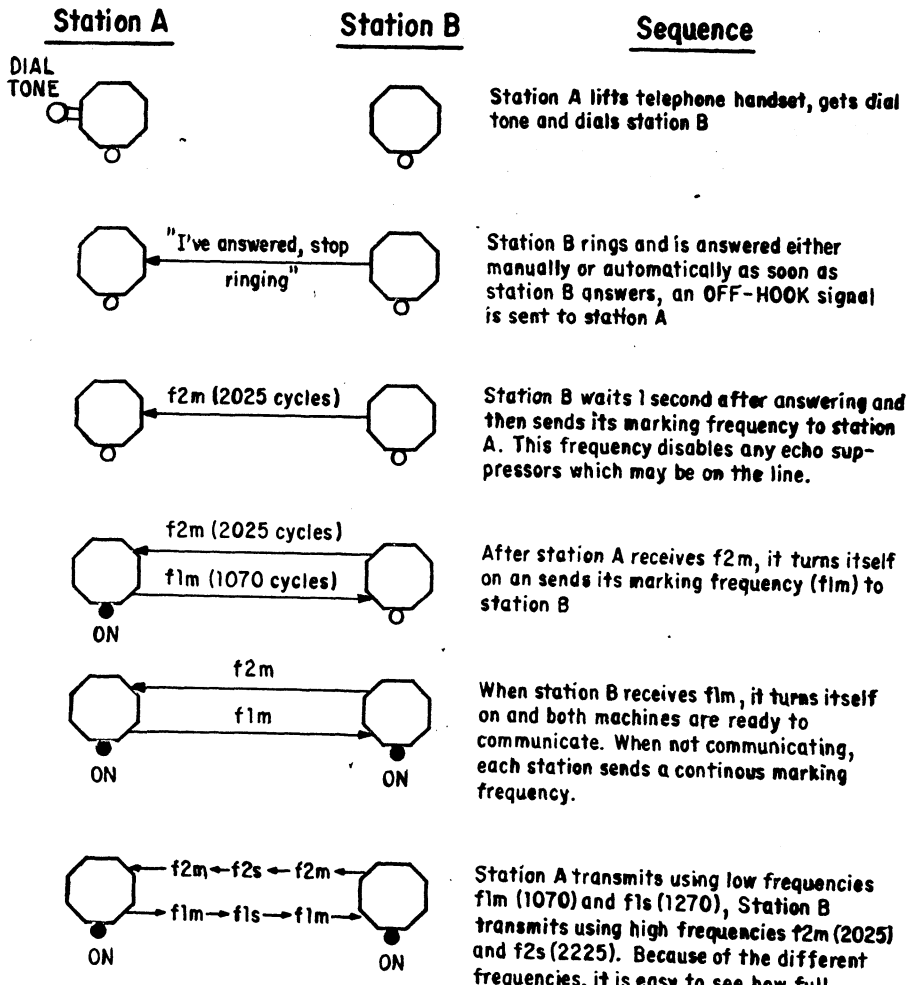
Note: Prices may vary with different operating companies.

4. Typical MODEM Operation

a. The 100 series AT&T Modem is frequently encountered in time-sharing communications. Specifically, the 103F is used on a leased line, and the 103A is used on the switched network.

This modem is compatible with low-speed terminals such as 2741 and 1050 that operate in the asynchronous mode - i.e., with start/stop technology. It provides for transmission of binary serial data up to 200 baud in either or both directions, simultaneously if desired. Two different frequency bands are used, each carrying data in only one direction. Each band carries a single carrier tone that is shifted to one of two frequencies, one representing the MARK state (a binary one) and one the SPACE state (binary zero). This technique is called Frequency Shift Keying (FSK).

DATA SET HANDSHAKING



b. Prior to the transmission of data, the two data sets must be placed on the data mode and an exchange of carrier tones called "data set handshaking" performed. This process follows a strict protocol and is performed each time the modems enter the data mode. The attached figure illustrates the "handshaking process."

c. After the channel is established, data transmission can take place in the asynchronous mode. The attached figure illustrates the transmission of one character, representing Carrier Return (CR), from the IBM 2741 to a S/360. After the CR key is depressed, the terminal sends a sequence of bits to the modem or data set. The modem, in turn converts (that is modulates) it to an equivalent sequence of transmission signals by frequency shift keying. At the receiving computer site, another modem reconverts (demodulates) this signal sequence to the same bit sequence originally sent by the terminal.

d. It should be noted that the modulation/demodulation process is "transparent" to the telecommunications equipment. It permits the digital signals from data processing equipment to be superimposed upon a system designed for voice communication.

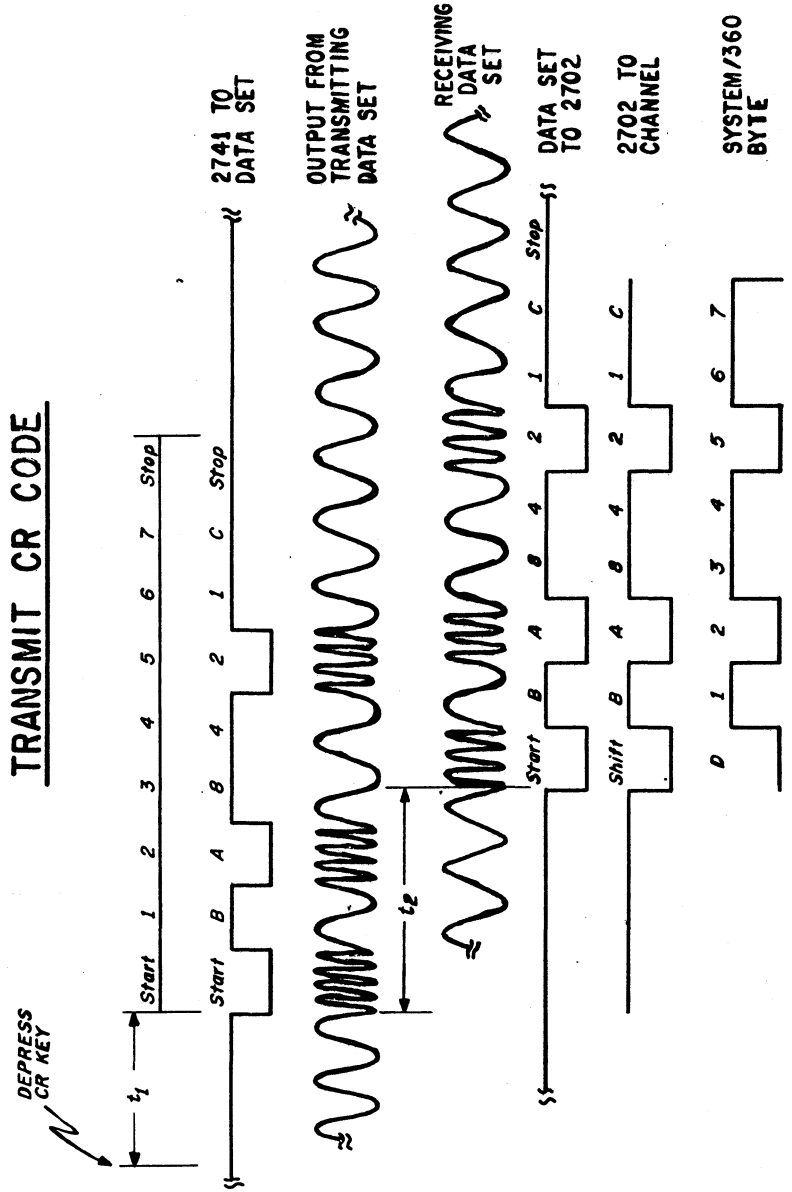
D. Terminals

1. IBM time-sharing systems support a variety of terminals for interactive computing.

   a. IBM 2741
   b. IBM 1050
   c. IBM 2260
   d. IBM 2740
   e. TTY 33 & 35

2. The IBM 2741 was specifically designed for time-sharing applications. It operates only in one-terminal-per line mode. It cannot be polled or addressed or otherwise operate in a multi-drop environment. An interrupt feature allows the attention key to be used by the terminal operator to halt transmission from the processor.

3. Although the IBM sales manual indicates there is only one model of the 2741, in effect, there are two types. There is a correspondence "type" and a BCD "type". The typing element specified determines the keyboard arrangement and the byte structure transmitted. Separate translation tables are required in the time-sharing software to support both types of 2741. All IBM time-sharing systems do not support both types. The following table indicates the support and the part number of the typing element required. The last three digits of the part number are imprinted on the typing element.



	2741 Correspondence	2741 EBCD
APL	RPQ E62267	RPQ F24235
ATS	Part No. 1167043 (without Interrupt Feature)	Not supported
CALL/360: BASIC	Part No. 1167087	Not supported
CALL/360: DATATEXT	Part No. 1167043	Not supported
CPS	Part No. 1167015 (translation req'd. 5 char.)	Part No. 1167963
CP67	Part No. 1167015 (RPQ 40681) req'd.	Part No. 1167963 (RPQ 40681) req'd.
RAX	Part No. 1167015 (announced for Version IV)	Not supported
TSS	Part No. 1167015 (support available)	Part No. 1167963 (RPQ 40681) req'd.

With the 1167015 typing element on the 2741 correspondence, type upper case 1 for > (greater than); type upper case 6 for < (less than); type upper case 7 for AND symbol; type upper case (to the right of P on second row) for ¬ (NOT symbol); type lower case (to the right of P on second row) for | (OR symbol).

4. There is a paper by Nat Rochester of the IBM FSD entitled, Type 2741 Typewriter Terminal Accessories for Time Sharing. This report contains simple design specifications for a working surface, a paper supply and stacker, and wheels to improve the utility of the IBM 2741.

5. From a human factors standpoint, the IBM 2741 has advantages in appearance and simplicity of operation. Lack of paper tape or card input is a handicap in some time-sharing/RJE applications. Some additional nitty gritties for the IBM 2741 are contained in the attached two pages.

MODES OF OPERATION

The 2741 Communications Terminal has two modes of operation: communicate mode and local mode. The mode of the terminal is controlled by terminal mode switch which is located on the left side of the typewriter stand (see Figure 3).

When in local mode, the terminal is disconnected from the communication line to the computer. The terminal can be used for typing, just as any other Selectric typewriter. Nothing can be transmitted or received.

When switched to communicate mode, the terminal is in a control-receive state. Automatically, the print element is shifted to lower case, if necessary, and the terminal goes to the communicate-transmit state. A (D) code is sent to the computer and the keyboard unlocks. The operator may now type whatever requests and text are desired.

The basic indication of the terminal state (transmit or receive) is the keyboard. The keyboard is locked whenever the terminal is not in transmit state. Receive-control is a momentary state in which the keyboard is locked and the terminal is waiting for a (D) from the computer. The (D) code places the terminal in a receive state. An automatic lower case shift occurs in the receive-control state if required.

Attention

This key performs two different functions, depending upon the mode of the terminal. If the terminal is in local mode, the attention key is used to test the terminal. A separate test of each key may be made to be sure the transmission circuits are operable.

When in communicate mode, this key causes a (C) to be sent to the computer.

If the interrupt special feature is installed, the attention key has a third function. This function is active only in communicate mode and only in the receive state. The operation of the attention key under these conditions causes a 200 ms signal to be transmitted, indicating that the operator wishes to interrupt the computer. In the transmit state, this key retains its standard function, sending a (C) to the computer.

IBM 2741 LINE CONTROL

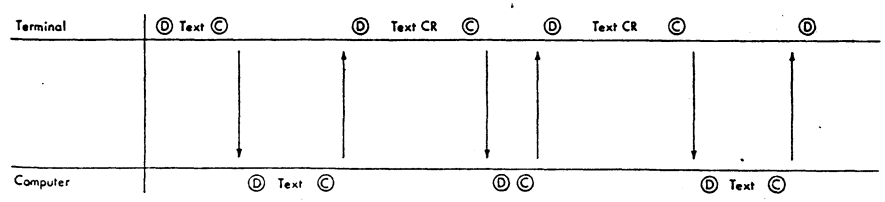
Line control becomes effective on the 2741 as soon as the terminal power switch is turned on and the terminal mode switch is set to communicate. The terminal is automatically put in the transmit state and a (D) code is sent. The operator may transmit by keying as on a typewriter.

Terminal transmission ends when the terminal transmits a (C). When the attention key is operated, a (C) is sent. When the carrier return key is pressed, a carrier return code followed by a (C) code is sent. The transmission of either code places the terminal in the receive-control state. The keyboard is locked.

When the computer transmits a (D) code, the terminal is placed in the receive state. Any valid character code received from the computer causes printing.

The receipt of a (C) code from the computer causes the terminal to switch to transmit mode. The keyboard is unlocked and the terminal automatically transmits a (D).

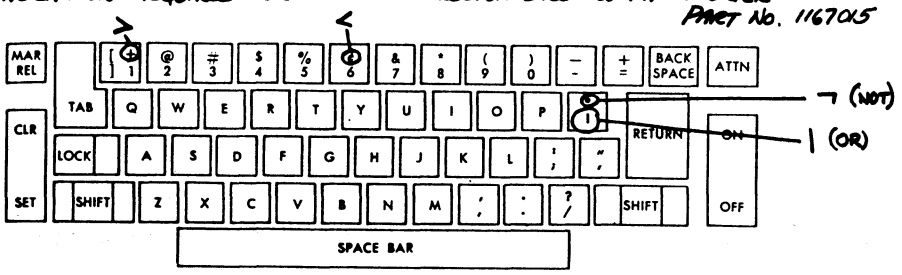
Figure 5 shows a typical line control sequence. The sequence may be ended only by the terminal. The operator terminates line control by switching to local mode or by turning the terminal power switch off.



IBM 2741 Line Control

CPS

TRANSLATION REQUIRED FOR 2741 CORRESPONDENCE WITH COURIER 72
PART No. 1167045



2741 Break Feature

The 2741 break feature modifies the IBM Terminal Control, Type I. It provides the control needed to operate the 2741 terminal in time-sharing applications. When this feature is present, it must be considered independent from the IBM Terminal Control, Type I for operation with 1050, 1060, 1070, 2740, and 2741 without the break feature. A different SAD command is required for 2741's with the break feature.

There are two modifications required to the IBM Terminal Control, Type I; they are:

1. Normally, the IBM Terminal Control, Type I, sets Channel End, Device End, and Unit Exception status upon receiving a © character during a Read or Inhibit command; with this feature, however, only Channel End and Device End will be set in this situation, allowing command chaining to occur.

2. The IBM Terminal Control, Type I, does not normally look at the Receive Data lead from Data Set or Modem while transmitting, but, when the 2741 Break Feature is present, the Receive Data lead is monitored for Space during a transmit operation. If a Space signal is detected at two successive stop-strobe times, the write-type command is ended with Channel End, Device End, and Unit Check status and the Intervention Required bit is set in the sense byte. The Timeout Complete sense bit in the LCW is used to store the fact that a Space was detected at the last stop-strobe time. With this bit on and the Receive Data lead still a Space at the next stop-strobe time, a Channel End, Device End, and Unit Check status is indicated. The Time-out Complete sense bit is reset before the ending status is presented.

PROGRAMMING NOTE:

After receiving Intervention Required during a Write command, the program may give a Prepare command, followed by a halt I/O. The 2702 presents Channel End and Device End status as soon as the line returns to Mark. Return of the line to Mark distinguishes the signal as Line Break and not as a nonoperational subset. If the line does not return to Mark, a nonoperational subset is indicated and the Prepare command ends immediately with Channel End, Device End, and Unit Check status and with the Intervention Required sense bit on.

6. The IBM 1050 is frequently supported in time-sharing systems although it was not designed specifically for the interactive or one terminal per line mode of operation. The 1050 system does provide the paper tape and card read/punching capability required in some time-sharing/RJE systems. The following features and RPQ's are generally used to modify the 1050 for the interactive environment of time-sharing.

   a. The Automatic EOB after Carrier Return. This RPQ (E28235) causes automatic transmission of an EOB character after the carrier return has finished returning when the Return Key is depressed.

   b. The Text Time-Out Suppression is provided to suppress the normal text time-out permitting an unlimited period of time for transmitting. The normal 9 - 18 seconds text time-out is undesirable when the 1050 is used as a time-sharing terminal.

   c. Receive Interrupt Control. This RPQ (E27428) provides the ability to send a 150 - 200 millisecond space signal to the 2702 to stop transmission. It is activated by depressing the Reset Key when terminal is receiving. Equivalent to the Interrupt feature on the 2741.

   d. Transmit Interrupt Control. This RPQ (E26903) allows the 1051, while transmitting, to recognize a 100 - 200 millisecond space signal, stop transmitting and revert to control mode status. The time-sharing system has the ability to interrupt the terminal. This RPQ is supported by TSS and CP67.

APPENDIX A

TIME-SHARING ACRONYM AND REFERENCE LIST

ACME - ADVANCED COMPUTER FOR MEDICAL EXPERIMENTATION

Provides interactive PL/1 (sub set) compiler for on-line experiments at Stanford Medical School. S/360 50 with LCS, 2701, 2702, 1800. Written by customer under government contract.

ABC - ALLEN BABCOCK COMPUTING, INC.

Provides RUSH (REMOTE USERS SHARED HARDWARE) on S/360 50 with LCS. Conversational PL/1. Developed under contract with IBM. Now providing commercial service. Los Angeles, California.

APL - A PROGRAMMING LANGUAGE

APL/360 is a conversational implementation of Iverson's language on S/360. Implemented at IBM Watson Research Center on Model 50. Supports 1050 and 2741 terminals. Available as Type III program -- file number is 360D-03.3.007.

APPLIED LOGIC CORP.

Provides service from Princeton, New Jersey on PDP-6 and 2 PDP-8.

ARPA - ADVANCED RESEARCH PROJECTS AGENCY

Department of Defense Agency that funded much of time-sharing development at MAC, SDC and others. Sometimes called the t-s Sugar Daddy to the tune of 12-13 million dollars per year. Also asking those it has helped to form computer network of some 16-17 installations.

ATLAS

First computer system built with paging mechanism. Designed for multi-programming and not specifically time-sharing. Had 32 pages of 512 words each. Parallel page address register compare included (associative registers). Jointly evolved at University of Manchester and Ferranti, Ltd.

continue
APPENDIX A

ATS - ADMINISTRATIVE TERMINAL SYSTEM

Provides text processing including data entry. Type II program available for 1460 and under DOS for S/360. Also announced for use under OS-MFT. Externally compatible with IBM Datatext Service.

BASIC - BEGINNERS ALL-PURPOSE SYMBOLIC INSTRUCTION CODE

An algebraic language resembling simple Fortran II and Algol. Originally developed at Dartmouth College for GE 265. Subsequently marketed by G.E. and G.E. BULL (overseas). Language has been extended and offered on G.E. 420 and 635. Language also available on SDS 940, B5500. Running on CDC 6400 and promised for RCA Spectra 70/46.

BBN - BOLT BERANEK & NEWMAN, INC.

One of first time-sharing systems (1962). McCarthy and Licklider of MIT were principals. Used DEC hardware PDP-1. Now commercially available on latest PDP hardware. Service is called TELCOMP.

BEST - BAYLOR EXECUTIVE SYSTEM FOR T/P

A system developed at Baylor Medical School provides t-s under OS-MFT. Supports 2260 (local and remote) and 1050 and 2740 terminals.

BRUIN - BROWN UNIVERSITY INTERPRETER

An interpretive time-sharing system implemented at Brown University on S/360 Model 50, modeled after PIL. Runs under BRUTUS (BROWN UNIVERSITY TERMINAL SYSTEM). Supports 1050 Terminal. Language is an ALGOL dialect.

BTSS - BERKELEY TIME-SHARING SYSTEM

Software system developed at University of California, Berkeley for SDS 940. Offers six languages - CAL, BASIC, QED, SNOBOL, ALGOL, FORTRAN IV, plus DDT (Debugging System)

CAI - COMPUTER ASSISTED INSTRUCTION

General term for on-line computer assisted learning.

CAL - CONVERSATIONAL ALGEBRAIC LANGUAGE

An interpretive or incremental compiler system implementing a JOSS-like language in the BTSS (SDS 940 TSS) at the University of California, Berkeley.

CALL/360: BASIC

IBM Information Marketing Depatment time-sharing service providing BASIC. Implemented on Model 50 S/360 with 512 k. Supports 2741 (Correspondence) and TTY 33/35. Known as Remote Terminal System (RTS) prior to announcement.

CCS - CONVERSATIONAL COMPILER SYSTEM

A subsystem of the TS monitor for SDS 940 developed at University of California, Berkeley. Interpretive or incremental system for ALGOL and FORTRAN IV. CCS and its compilers were developed jointly by Com-Share, Inc., Tym Share, Inc., and SDS.

COMNET - COMPUTER NETWORK CORPORATION

Provides commercial service on B5500 in Washington, D. C. Languages are FORTRAN IV, ALGOL, COBOL, and BASIC. TTY Terminals.

COM-SHARE, INC.

Provides commercial service in midwest on SDS 940 via headquarters in Ann Arbor, Michigan. Outgrowth of University of California, Berkeley TS System.

CP40 - CONTROL PROGRAM 40

Experimental system developed at the IBM Scientific Center, Cambridge, Mass. with special modifications to S/360 Model 40 hardware. Forerunner of CP67.

CP67 - CONTROL PROGRAM 67

A time-sharing system developed at the Cambridge Scientific Center for the S/360 Model 67. CMS (Cambridge Monitor System) provides the conversational capability in the framework of CP. Any S/360 Operating System could work in this framework, i.e. OS, DOS, etc. of the virtual system provided by CP67. Type III program number is 360D - 05.2.005.

CMS - CAMBRIDGE MONITOR SYSTEM

Conversational subsystem of CP67.

CPS - CONVERSATIONAL PROGRAMMING SYSTEM

A Type III Program (360 D - 3.4.016) available from PID to provide Conversational PL/1 in a partition of OS-MFT. Interpretive System in Segmented Core. RJE capability included. Also see Allen-Babcock Computing, Inc.

CRBE - CONVERSATIONAL REMOTE BATCH ENTRY

This package provides low speed (1050, 2740) remote access to OS-MVT for file creation and modification, submission to batch, etc. FORTRAN syntax checking is optional.

CS-I - CONVERSATIONAL SYSTEM/I

An experimental system providing PL/1 language on S/360 50. Developed at the IBM Boston Programming Center.

CTSS - COMPATIBLE TIME-SHARING SYSTEM

MIT's System - generally conceded to be the first of the time-sharing systems and most significant. Still in operation - two systems 7090/7750 at MAC & MIT Computer Center. Languages - FAP, MAD and FORTRAN and many others.

DATATEXT

A terminal service provided via contract with IBM Information Marketing Department (IMD). Text processing capability (externally compatible with ATS) from central computer.

DDT - DYNAMIC DEBUGGING TOOL

On-line debugging routine that permits modifications at the symbolic language level for BTSS (SDS 940 TSS). Acronym also used for debugging system on earlier systems.

DEC - DIGITAL EQUIPMENT CORP.

Computer manufacturer in Maynard, Mass. that has developed and sold the PDP series.

DIA MAG - MATHMATICAL APPLICATION GRENOBLE

ALGOL conversational compiler on 7044/PDP-8 hardware configuration. Supports TTY. Implemented at Grenoble Institute of Mathematics. Grenoble, France. Working for two years.

DISPLAYTRAN

A graphic (2250) system using interpretive FORTRAN under OS-MFT. Developed jointly by IBM and U.S. Navy Weapons Laboratory at Dahlgren, Va. Likened to QUIKTRAN with special command language. TIE paper (Z77-7176).

DROS - DISK REMOTE OPERATING SYSTEM

Type III program (360 D-05.1.011) providing Remote Job Entry under DOS/360 using 32 k foreground and BTAM and STRAM to support 1050, 1978, and 1130 terminals. TIE Paper (Z77-7286). Program file order number is 360D-05.1.001.

GENIE

Project GENIE is the name applied to the time-sharing development at University of California, Berkeley involving the SDS 940 and its TS monitor. Supported by ARPA funds.

HASP - HOUSTON AUTOMATIC SPOOLING PRIORITY SYSTEM

An automatic SYS IN/SYSOUT/SYSPUNCH Spooling package that performs peripheral functions with extensive buffering and blocking to enhance performance of OS/360 MFT. Includes RJE support of STR devices - 1978 and Model 20 with Communication Adapter feature.

HSRJE - HIGH SPEED REMOTE JOB ENTRY

Remote entry to on-line batch via synchronous communications systems, i.e. STR or BSC above 600 baud.

IBM - TSS - IBM SYSTEM/360 MODEL 67 TIME SHARING SYSTEM

General purpose time-sharing system providing FORTRAN IV and Assembler languages, full data management capability, and debugging system (PCS). Virtual memory and paging implemented with Model 67 hardware - dynamic address translation (DAT). Supports 1050, 2741 (BCD), and TTY terminals. Proposed or sold on a "controlled marketing" basis only. Master index (C28-2023).

IPL-V - INFORMATION PROCESSING LANGUAGE V

Language capability available on the SDC-TSS for character string manipulation. Makes SDC-TSS "general purpose".

ITSS - INTERIM TIME-SHARING SYSTEM

A special system jointly developed by IBM and General Motors Research under OS/PCP to provide multiple time-sliced partitions for 2250 graphics support. Uses Model 67 DAT with RPQ to retain S/360 normal PSW mode.

JOSS - JOHNNIAC OPEN-SHOP SYSTEM

Early system on Johnniac hardware at RAND Corp. First with interpretive technique. Language is algebraic, an ALGOL dialect and forerunner of BASIC. ARPA sponsored.

JOVIAL - JULES' OWN VERSION OF AN INTERNATIONAL ALGEBRAIC LANGUAGE

An ALGOL dialect developed at Systems Development Corp. (SDC) by Jules Schwartz. Essentially aimed at Command and Control applications. TINT is time-sharing version of JOVIAL at SDC.

KEYDATA, INC.

First completely commercial TS service out of Cambridge, Mass. On SR 491 hardware now after start on PDP equipment.

LAFFF - LANGUAGE FOR THE AID OF FINANCIAL FACT FINDERS

A subsystem of the Dartmouth Time-Sharing system used in TUCK School (Business Administration) for analyzing data on publicly traded companies. Data bank drawn from Standard Statistics Corporation's COMPUSTAT tape. Price-earnings ratios, dividends, closing price, etc.

LISP - LIST PROCESSING LANGUAGE

A list processing language modified from its batch-processing counterpart to run on the CTSS/MAC system. Other LISP implementations for time-sharing subsequently made at SDC, BERKELEY, and BBN system.

LSRJE - LOW SPEED REMOTE JOB ENTRY

Remote job entry with low speed terminals, i.e., TTY, 1050, 2741.

M44

An experimental system developed at the IBM Watson Research Center to study virtual machines and paging. Hardware is highly modified 7044. MOS is Modular Operating System. Virtual machines if 44X. Forerunner of Model 67 to certain extent.

MAC - MULTIPLE ACCESS COMPUTER, MACHINE AIDED COGNITION

Project implemented upon second 7094 at MIT. See CTSS. Funded by ARPA. $3 million budget. Presently installing MULTICS.

MAD - MICHIGAN ALGORITHM DECODER

Language developed at University of Michigan based on ALGOL for batch system and modified for CTSS at MIT.

MEDINET

A hospital and medical information system designed by Bolt Beranek and Newman, Inc. and G.E. G.E. hardware.

MTS - MICHIGAN TIME-SHARING

Interim system implemented for Model 67 at University of Michigan using relocation hardware. OS compatible, FORTRAN G, ASSEMBLER F, File Maintenance, PIL, SNOBOL IV and five other languages. Supports 1050, 2250, 2741, TTY and small computers on campus. Background 1-3 streams. Earlier MTS version ran on Model 50 without relocation hardware.

MULTICS - MULTIPLEXED INFORMATION AND COMPUTING SERVICE

General purpose time-sharing system being developed jointly by MIT (Project MAC), G.E., and Bell Labs, Murray Hill using the G.E. 645 computer.

PHONETRAN

An experimental calculator system designed in IBM ASDD lab using a touch-tone pad for input and voice answerback for output.

PIL/L - PITTSBURGH INTERPRETIVE LANGUAGE

A conversational system on S/360 50 at University of Pittsburgh. A JOSS-like language on a dedicated system. Interpretive.

QED

A text editor with line organization and content addressing characteristics in the Berkeley (University of California) TS system on the SDS 940 using TTY terminals. Allows generation and modification of coding at the symbolic language level.

QUIKTRAN

Terminal service provided by contract with IBM Information Marketing Department. Hardware is 7740/7044. Supports 1050, 2741 (Courier Correspondence) and TTY. Interpretive system. QUIKTRAN[2] is current version with greatly improved execution speeds over QUIKTRAN[1].

RAX - REMOTE ACCESS COMPUTING SYSTEM

A Type II program providing Basic FORTRAN IV and Basic Assembler language in TS environment. Support 1050 and 2260 (local) on Models 30, 40 and 50. Users Manual (H20-0354). TIE paper (Z77-7222). Applications Description (H20-0353).

RITS - REMOTE INPUT TERMINAL SYSTEM

A low speed remote input package originally written by SBC for NASA GODDARD under contract from GEMRO. Interfaces to OS and used in the ASP configuration. Presently used by Columbia University, Westinghouse and Combustion Engineering. Forerunner of CRBE.

RTS - REACTIVE TERMINAL SYSTEM

Service provided by ITT data services on S/360 Model 50. Modified RAX with OS FORTRAN-G Compiler to provide full FORTRAN. BASIC language to be added. Supports variety of terminals.

RUSH - REMOTE USERS SHARED HARDWARE

System commercially available from Allen-Babcock Computing, Inc. Conversational PL/1. Similar to CPS.

SABRE

American Airlines reservation system developed jointly by AA and IBM. System runs on duplexed 7090's and supports over 1,000 special terminals all over U. S.

SDC - TSS -- SYSTEM DEVELOPMENT CORPORATION - TIME-SHARING SYSTEM

One of first TS systems. Developed at SDC with ARPA funding on AN/FSQ 32 (transistorized SAGE computer) with PDP-1 handling communications. Languages available are TINT and IPL-V.

SDS 940 - SCIENTIFIC DATA SYSTEMS 940

A modified SDS 930, including a paging scheme, privileged instructions, and overlapped I/O to secondary memory. Changes specified at University of California, Berkeley by Melvin Pirtle.

SHARER

A time-sharing system embedded within the SCOPE multi-processing batch system for the CDC 6600. Developed at NYU under AEC funding.

SNOBOL

A character string manipulative language for information retrieval, linguistics, and compiler functions. Available on BTSS (SDS 940 TSS).

TELCOMP

Name given to the commercial service provided by Bolt Beranek and Newman. Uses PDP-7 and 8 computers. Described as a "derivative of the JOSS language".

TINT - TELETYPE INTERPRETER

Language capability available on the SDC-TSS system. An interpretive system for JOVIAL (algebraic language developed at SDC). See SDC-TSS.

TORTOS - TERMINAL ORIENTED REAL TIME OPERATING SYSTEM

Generalized TS system to provide full resources of OS/360 plus some special functions to terminal users at UCLA. Four basic priority classes - batch, background, terminal, and real time are provided. RJE or conversational Syntax for FORTRAN, PL/1 and JCL. TIE paper (Z77-7169).

TSM - TIME SHARING MONITOR SYSTEM

Experimental system developed in IBM ASDD on 7090 hardware. Languages: FAP, FORTRAN, GPSS, and PAT (PERSONALIZED ARRAY TRANSLATOR) an interpretive sub-system. 7281-II Data Communications Channel for terminal multiplexing.

TUCC - TRIANGLE UNIVERSITIES COMPUTING CENTER

Non-profit organization at Raleigh, North Carolina set up by Duke, North Carolina, and North Carolina State to provide computer facilities to universities in North Carolina. Model 30's and other terminals on campus on-line to Model 75 at TUCC. Modified OS using LCS to provide batch service with RJE/RJO via STR access method. CPS used for interactive computing.

TYMSHARE, INC.

Offers TS service to subscribers in California on SDS 940 from headquarters in Palo Alto. Languages are CAL, BASIC, FORTRAN, ALGOL and QED.

UCC - UNIVERSITY COMPUTING COMPANY

UCC operates eleven computing centers providing batch services and RJE to 1108 and a variety of data processing services. Headquarters in Dallas, Texas. Recently purchased Benson-Lehner and attempted to acquire Western Union.

WATFOR - WATERLOO FORTRAN

Compiler developed for S/360 at the University of Waterloo, Waterloo, Ontario. Full FORTRAN IV, core resident, and very fast compile. Particularly suited to student jobs where fast compile is essential. Will run under OS. Program tape provided with maintenance for $300.

WESTIME 300

A time-sharing system offered for the IBM 1130 by Western Telematic, Inc. of Arcadia, California. Supports 8 terminals in 8K, single disc 1130. IBM 2741 and KSR 33 supported. Conversational FORTRAN. Price of Multiplexor and software not indicated in Datamation ad October 68.

WHITEWELD

New York based service company providing financial information for banks, trusts, and mutual funds via SDS 940 hardware system. First Financial Language (FFL) provides ability to manipulate data.

WRAP - WAYNE REMOTE ACCESS PROCESSOR

TS system developed at Wayne State University at Detroit, Michigan. It runs under OS/360 and makes available all OS language, i.e., PL/1, FORTRAN, COBOL, ASSEMBLER, RPG. Uses multiple partition roll-in/roll-out. Supports 1050, 2741 and 2260.

APPENDIX B

TIME-SHARING READING AND REFERENCE LIST

1. Arden, B.W., Galler, B.A., O'Brien, T.C., Westervelt, F. H., "Program and Addressing Structure in a Time-Sharing Environment", Journal of the ACM, (January, 1966).

2. Bairstow, J. N.,"Time-Sharing: A Computer for Everyone", Electronic Design, (April 25, 1968).

3. Bauer, W.F., Hill, R.H., "Economics of Time-Shared Computing Systems", Datamation, (Part I - November, 1967) (Part II - December, 1967).

4. Bleiweiss, L.S., et al, "CPS Terminal User's Manual", Technical Report TM 48.67.006, (September, 1967).

5. Castleman, P. A., "An Evolving Special-Purpose Time-Sharing System", Computers and Automation, (October, 1967).

6. Coleman, C. D., "CS/I Primer", Technical Report BPC 1, IBM Boston Programming Center.

7. Conover, J.W., "TORTOS (Terminal-Oriented Real Time Operating System)", Tie Paper Z77-7169, (July, 1967).

continue
APPENDIX B

8. Corbato, F.J., Daggett, M., Daley, R.C., "An Experimental Time-Sharing System", Proc. SJCC, (1962).

9. Corbato, F.J., et al, "The Compatible Time-Sharing System - A Programmers Guide", The M.I.T. Press, (1963).

10. Corbato, F.J., Vyssotsky, V.A., "Introduction and Overview of the Multics Systems", Proceedings, - Fall Joint Computer Conference, (1965).

11. Critchlow, A.J., "Generalized Multiprocessing and Multiprogramming Systems", Proceedings - Fall Joint Computer Conference,

12. Deutsch, L. P., Lampson, B. W., "An Online Editor", Communications of the ACM - Vol. 10, Number 12, (December, 1967).

13. Devonald, C.H., Fotheringham, J.A., "The Atlas Computer", Datamation, (May, 1961).

14. Dunn, T.M., Morrissey, J. H., "Remote Computing - An Experimental System Part I: External Specifications", Proc. SJCC, (1964).

continue
APPENDIX B

15. Enslein, K., et al, "The Rochester Direct-Access Time-Shared System", Computers and Automation, (October 1967).

16. Evans, T. G., Darley, D. L., "On-Line Debugging Techniques: A Survey", Proceedings-Fall Joint Computer Conf., (1966).

17. Falkoff, A. D., Iverson, K. E., "APL/360 User's Manual", (July, 1968).

18. Gagliano, F. W., "DISPLAYTRAN", Tie Paper Z77-7176, (May, 1967).

19. Gallenson, L., Weissman, C., "Time-sharing Systems: Real and Ideal", SDC Professional Paper SP-1872, (March, 1965).

20. Glaser, E. L., Couleur, J. F., Oliver, G. A., "System Design of a Computer for Time-Sharing Applications", Proceedings - Fall Joint Computer Conference, (1965).

21. Glauthier, T. J., "Computer Time-Sharing: Its Origins and Development", Computers and Automation, (October, 1967).

continue
APPENDIX B

22. Harrison, M. C., Schwartz, J. T., "SHARER, a Time-Sharing System for the CDC 6600", Communications of the ACM - Vol. 10, Number 10, (October, 1967).

23. Hittel, L. A., "Some Problems in Data Communications Between The User and the Computer", Proceedings, Fall Joint Computer Conf., (1966).

24. Hobbs, W. F., Levy, A. H., McBride, J., "The Baylor Medical School Teleprocessing System - Operational Time-Sharing on a System/360 Computer", Proc. SJCC, (1968).

25. Hyman, H., "The Time-Sharing Business", Datamation, (February, 1967).

26. Irwin, M. R., "Government Policy Implications in Data Management", Datamation, (June, 1968).

27. Keller, J. M., Strum, E. C., Young, G. H., "Remote Computing - An Experimental System Part 2: Internal Design", Proc. SJCC, (1964).

28. Kemeny, J. G., Kurtz, T. E., "The Dartmouth Time-Sharing Computing System," Final Report under NSF Grant GE-3864, (April, 1967).

continue
APPENDIX B

29. Kinslow, H. A., "The Time-Sharing Monitor System", Proceedings - Fall Joint Computer Conf., (1964).

30. Lichtenberger, W. W., Pirtle, M. W., "A Facility for Experimentation in Man-Machine Interaction", Proceedings - Fall Joint Computer Conference, (1965).

31. Licklider, J. C. R., "Man-Computer Symbiosis", IRE Transactions on Human Factors in Electronics, Vol. HFE-1, 4-11, (March, 1960).

32. Licklider, J. C. R., "Man-Computer Partnership", International Science and Technology, (May, 1965).

33. Main, J., "Computer Time-Sharing-Everyman at the Console", Fortune Magazine, (August, 1967).

34. McCarthy, J., "Time-Sharing Computer Systems", Computers and the World of the Future (ed) M. Greenberger, M.I.T. Press, (1962).

35. McCarthy, J., Boilen, S., Fredkin, E., Licklider, J. C. R., "A Time-Sharing Debugging System for a Small Computer", Proceedings-Spring Joint Computer Conf., (1963).

continue
APPENDIX B

36. McNamara, J. E., "Data Communications Systems for Time-Sharing Computers", Lincoln Lab Paper, (Date Unknown).

37. Mendelson, M. J., England, A. W., "The SDS Sigma 7: A Real-Time Time-Sharing Computer", Proceedings - Fall Joint Computer Conference, (1966).

38. Nelson, W. D., "How to Pick a Time-Sharing Service", Computer Design, (August, 1968).

39. No Author, "Adding Computers - Virtually", Computing Report, (March, 1967).

40. No Author, "An Introduction to CP-67/CMS", IBM Cambridge Scientific Center Paper, (August, 1968).

41. No Author, "Systems Development At TUCC: 1966-67", Triangle Universities Computation Center Paper, (February, 1966).

42. No Author, "Time-Sharing Market Put At $2.5 Billion in 1971 Software, Service Firms Riding Growth Boom", Electronic News, (April 29, 1968).

43. No Author, "Time-Sharing: Who Pays How Much For What?", Datamation, News Briefs, (February, 1967).

continue
APPENDIX B

44. No Author, "Wayne Remote Access Processor User's Guide", Computing and Data Processing Center, Wayne State University, (January, 1968).

45. Oestreicher, M. D., Bailey, M. J., Strauss, J. I., "George 3 - A General Purpose Time-Sharing and Operating System", Communications of the ACM - Vol. 10/Number 11, (November, 1967).

46. O'Neill, R. W., "Experience Using a Time-Shared Multi-Programming System with Dynamic Address Relocation Hardware", Spring Joint Computer Conference, (1967).

47. O'Rourke, T. J., "The Many New Uses of Time Sharing", Computers and Automation, (October, 1967).

48. O'Sullivan, T. C., "Terminal Networks For Time-Sharing", Datamation, (July, 1967).

49. Patrick, R. L., "Time-Sharing Tally Sheet", Datamation, (November, 1967).

50. Pointel, N., Cohen, D., "Computer Time Sharing - A Review", Computers and Automation, (October, 1967).

continue
APPENDIX B

51. Rosenberg, A. M., Colten, R., Myles, S. B., "How to Cash in on the Computer Utility - Parts I and II", Data Processing Magazine, (Part I - March, 1968, Part II - April, 1968).

52. Rosenberg, A. M., "Time-Sharing: A Status Report", Datamation, (1966).

53. Ryan, J. L., Crandall, R. L., Medwedeff, M. C., "A Conversational System for Incremental Compilation and Execution in a Time-Sharing Environment", Proceedings - Fall Joint Computer Conf., (1966).

54. Schatzoff, M., Tsao, R., Wiig, R., "An Experimental Comparison of Time-Sharing and Batch Processing", Communications of the ACM Vol. 10, Number 5, (May, 1967).

55. Schwab, B., "The Economics of Sharing Computers", Harvard Business Review, (September-October, 1968).

56. Schwartz, J. I., Coffman, E. G., Weissman, C., "A General Purpose Time-sharing System", Proceedings - Spring Joint Computer Conf., (1964).

57. Shaw, J. C., "JOSS: A Designer's View of an Experimental On-Line Computing System", Proceedings - Fall Joint Computer Conf., (1964).

58. Theis, D. J., Hobbs, L. C., "Low-Cost Remote CRT Terminals". Datamation, (June, 1968).

59. Tuttle, J. R., et al, "The Brown University Interactive Language", (September, 1968).

60. Wilkinson, B., "Some Problems with Time-Sharing", Datamation, (May, 1968).

APPENDIX C

GLOSSARY OF TIME-SHARING TERMS

The following definitions with some exceptions were extracted from the IBM Computer Description Manuals and arranged in alphabetic order to facilitate reference.

ASNYCHRONOUS TRANSMISSION is the type of transmission where each information character is individually synchronized and is normally preceded by a start signal, which serves to prepare the receiving mechanism for the reception and registration of a character, and is followed by a stop signal, which serves to bring the receiving mechanism to rest in preparation for the reception of the next character.

BAUD is the number of discrete conditions or signal events per second. If each signal represents only one bit condition, baud is the same as bits per second.

BATCH PROCESSING is the processing of a stream or batch of tasks in a generally sequential manner, by an operating system controlled by commands supplied by the task originator prior to the commencement of processing for the task.

BINARY SYNCHRONOUS COMMUNICATIONS (BSC) is transmission that is serial-by-character and serial-by-bit, over half duplex lines, and at a transmission rate which varies between medium and high (from 600 to 230,400 bits per second). Bit and character synchronization are established at the beginning of each transmission by transmitting a synchronizing pattern. This synchronization (sometimes called character phase) is then maintained

until the transmission stops, signalled by line control characters. The synchronization is controlled by a clock located either within the BSC station or the modem.

COMMUTATOR - a table or chained queue in which each entry represents a user logged-on the system. The scheduler steps around the commutator to give a time-slice to each active user.

CONVERSATIONAL I/O - is the ability to write to the terminal and read from the terminal during the execution of a user program. This facility is basic to time-sharing and is one factor that distinguishes time-sharing from remote batch entry.

A DATA SET is a device which performs the modulation-demodulation and the control functions necessary to provide compatibility between business machines and the communication facilities. Another word used for this device is modem, which is the contraction of modulator-demodulator.

"DEMAND PAGING" is the allocation of storage to a segment, page or procedure based upon the actual demand for storage space by that procedure, rather than the allocation of storage to a procedure based upon its anticipated or predicted demand. The user's virtual page is not called in to main memory until it is "demanded".

A DESK CALCULATOR facility offered by a time-sharing system is a simple command or set of commands by which the user may operate his terminal to perform those functions (such as addition, subtraction, multiplication and division) that a desk calculator performs.

"DIALUP" refers to the use of a dial or push-button telephone to initiate a station-to-station telephone call.

A DROP refers to a station or terminal.

DYNAMIC ADDRESS TRANSLATION is the process of converting virtual addresses into actual processor storage addresses during instruction execution. The translation feature is implemented in hardware in the processor.

HEIRARCHIAL STORAGE is the grouping of storage by class or rank. The ranking of such storage is normally determined by the access time associated with it. Therefore, the storage would be ranked in the following manner: internal storage; LCS (auxiliary storage); secondary storage such as tapes, drums and disks.

DYNAMIC PROGRAM RELOCATION is the relocation of a program before it has completed execution and without modification, to another part of storage, in a manner that permits subsequently resuming its

execution. This facility normally involves the use of a hardware feature called a relocation register.

FAIL SAFE: A system that can perform its entire workload in the presence of any single malfunction is said to be fail safe.

FAIL-SOFT: A fail-soft system performs only the essentials of its workload in the presence of malfunction.

FILE I/O is the accessibility to the terminal user of data files stored on devices such as drums, disks and tapes at the computer site, within a problem-solving environment. These data files may have been entered and saved by the terminal user, or created by an on-site computer user under the time-sharing system, or under another program operating independently of the time-sharing system.

A FULL DUPLEXED (FDX) line or circuit is capable of the simultaneous and independent transmission and reception between two points--in both directions.

A HALF-DUPLEXED (HDX) circuit or line is capable of transmitting and receiving in both directions, but is not capable of simultaneous and independent transmission and reception. It cannot send and receive at the same time.

A HIGH SPEED TERMINAL is a computer system or a group of components (reader, punch, printer) that are not keyboard driven.

An IN-HOUSE TIME-SHARING SYSTEM is a nonproprietary system which a customer may obtain from a vendor and install on his own equipment, in his own "house".

INTELLIGENT TERMINAL - a terminal device that incorporates stored-program logic such as the IBM 1130, Model 20, or other S/360 model.

An INTERPRETIVE compiler decodes user source statements into an intermediate code (pseudocode) that it can execute without translating the statement further into machine language.

A LOCAL terminal is a terminal which is connected directly to the computing system via its control unit.

LOW SPEED terminals will be the class of all keyboard-driven terminals. Other devices such as paper tape readers and card readers may be attached to the keyboard-driven device, but if the terminal may be keyboard-driven, it will be classed as a low speed terminal.

MODULATION is the process by which some characteristic of an electrical wave is varied in accordance with another wave or signal. In Data Processing, modulation is used to make business machine signals compatible

with communications facilities. In a specific case, it is conversion from digital signals to audio signals for transmission over communication lines.

A MULTIDROP system is one in which two or more stations or terminals are connected to a single line. A multidrop system must determine which terminal on the line sent a message, and/or which terminal on a line will receive a particular message.

MULTIPROGRAMMING is a technique by which a computing system can interleave the execution of two or more generally unrelated programs, parts of which are residing together in main storage.

MULTI-PROCESSOR: A system consisting of two or more CPU's, ALU's or processors that do communicate for the purpose of sharing a workload. (Although shared main memory is sometimes considered to be a pre-requisite, there are multi-processing systems without this facility.)

MULTIQUEUED EXPONENTIAL SCHEDULING is a technique where different queue levels are established such that every terminal request initially is placed on the first queue and is "pushed down" to lower queues if the request becomes more compute bound.

MULTISYSTEM: A multisystem is composed of two or more computing systems which may be the same or may be different models and which

are capable of intercommunication between them in some manner. A multiprocessing system may be said to a subset of a multisystem. The multisystem approach was implemented primarily to provide high availability to the installation.

OPERATIONAL CYCLE TIME - the time required to give one time slice to all active users being multi-programmed in a time-sharing system; the time required for one trip around the commutator.

PAGE: A program segment of fixed size. In the System/360 Model 67, a page is a set of 4096 consecutive bytes. The size of a page will vary between different systems.

PAGING or PAGE TURNING is a technique for locating/relocating and swapping program sections (pages) to and from main memory, utilizing both hardware and software assists.

A PRIVATE or leased-line network is a network reserved for the exclusive use of one customer.

A PUBLIC network is a network provided by a common carrier for use by many customers.

continue
APPENDIX C

PUSH-DOWN stack is a list of items where the last item entered becomes the first item of the list and relative positions of the other items are pushed back one.

QUANTUM - a specified unit of time allocated to a user program when it gains the CPU for execution. A time slice may consist of a quantum or multiple quanta.

RECURSION: the continued repetition of the same operation or group of operations. "Recursive" pertains to a process that is inherently repetitive. A recursive program or block of code can call or transfer control to one of its own entry points.

REFRESHEABLE CODE - a program module containing appropriate flag bits to indicate that a fresh copy of the code could be obtained from the library during an attempt to recover from an error condition.

A REENTRANT PROGRAM is one which may be interrupted during the execution of one user request, entered by another user and subsequently be reentered at the point of interruption by the first user, producing the desired results for all. Each user entering the code brings with him a pointer to his own parameter list. All information relating to a particular user is posted to (placed in) an external area in order for his processing to resume at the proper place.

continue
APPENDIX C

RELOCATION REGISTER - a single hardware register used to dynamically relocate code that has been partially executed. A relocation constant is loaded into the register and subsequently added to the addresses generated during program execution. The relocation constant is established as the code is relocated in core storage on each time slice.

A REMOTE terminal is a terminal which is connected to the computing system via communication lines.

ROLL-IN/ROLL-OUT - a memory management technique employed in time-sharing systems whereby user programs are written to secondary storage at time slice end and are retrieved from secondary storage before starting the next time slice.

ROUND ROBIN - a simple scheduling technique whereby users are served on a first come, first served basis.

SCHEDULING ALGORITHM - a scheduling algorithm is a series of rules and decisions used to determine how central processing unit time (as well as other system resources) is to be allocated among the tasks (jobs or programs) contending for it.

continue
APPENDIX C

SERIALLY REUSABLE PROGRAM is a reusable program which may be used more than once but which has the property that a user must enter the code, complete his requirements, and exit before another user may enter.

A SIMPLEX line is a circuit capable of one-way operations only. Communication may proceed in one direction only, with no capability for reversing the direction.

A SINGLE-DROP system is a system in which only one terminal or station is connected to each communications line. Most problem solving time-sharing systems of today have single-drop capabilities only.

SWAPPING - another name for ROLL-IN/ROLL-OUT technique of memory management.

TELPAK: Broadband communication channels for transmitting data at rates greater than 60,000 characters.

TIME SLICE - the period of CPU time allocated to a user program by the scheduler in a time-sharing system. A time slice can be a fixed time unit or multiples thereof. (See Quantum) or a work slice to execute a fixed number of program statements.

continue
APPENDIX C

VIRTUAL MEMORY - is a concept which describes an imaginary memory whose extent covers the entire addresseable range of the machine design and which appears to a programmer to have the characteristics of real core storage.

VOICE-GRADE: a circuit of sufficient bandwith to permit a data transfer rate up to 2,400 bits per second, generally with a frequency range of 300 to 3,000 cycles per second.

WATS: Wide Area Telephone Service is a service provided by telephone companies which permit a customer by use of an access line to make calls to telephones in a specific zone on a dial basis for a flat monthly charge. Monthly charges are based on the size of the area in which the calls are placed, not on the number or length of calls. Under the WATS arrangement, the U.S. is divided into six zones to be called on a fulltime or measured-time basis.

APPENDIX D

HYPERVISOR

Purpose & Usage

Hypervisor is a control program designed to provide the capability to operate the Remote Access Computing System (RAX) concurrently with the Operating System/360 (OS/360) in the background in a single-CPU system. In this mode of operation, the core storage is partitioned, by means of the program-controlled prefix switches provided by RPQ into two equal areas, i.e., high-core and low-core areas, and RAX operates in the high-core area while OS/360 operates in the low-core area. The two systems have separate work loads, and operate independently of each other sharing the CPU. Hypervisor provides the interface between RAX and OS/360 in order that the two autonomous systems may operate without interference from each other.

Description

One part of Hypervisor is resident in the RAX partition with the core-resident RAX programs, and another is in the OS/360 partition as an OS/360 task. When Hypervisor gives control of the CPU to RAX, it switches the CPU prefix to the high-core mode in which all core storage references are statically relocated by hardware so that the core storage area in the RAX domain, which is the upper half of the, say, 512K core storage, can be accessed by addresses 0 through 256K. Thus, RAX

is enabled to run without address modifications in the upper half of the core storage. Whenever the CPU becomes free while operating in the RAX mode, Hypervisor switches the CPU prefix to the low-core mode and transfers control of the CPU, across the partitioning, to OS/360. Hypervisor intercepts all I/O interrupts while the CPU is operating in the OS/360 mode, and returns control of the CPU to RAX upon an I/O interrupt from an I/O unit dedicated to RAX. In this way, any CPU idle-time in the RAX operations is utilized to run OS/360 without degradation of response time in the RAX operations.

The sequence of operations to start up a Hypervisor system is as follows: The operator IPLs OS/360 after manually setting the Bit 6 of the Prefix Control Register (PCR) to 1 to limit the OS/360 core storage area to one half of the total core storage in the system. (For a description of the PCR, see Hardware Requirements below.) Upon completion of the IPL, the Hypervisor Task, which contains the RAX IPL routine (described in Step 2) and I/O Interrupt routine (described in Step 4), is read into OS/360 as an OS/360 task. The Hypervisor routines perform the functions described below:

1. When Hypervisor Task receives control from OS/360, it invokes the OS/360 Type 1 SVC to put itself in the Supervisor state with interrupts masked off and storage protection key of zero. The Hypervisor Task saves OS/360's I/O new PSW and replaces it with the Hypervisor's; the instruction address of the I/O new PSW points to the Hypervisor's I/O Interrupt routine. It resets the PCR to all-zeros, by means of the Set Prefix and Branch (SPB) instruction (for detail, see Hardware

Requirements below), to put the system into the non-partitioned mode in which the CPU can access the entire core storage, and moves the RAX IPL routine into the upper core. The Hypervisor Task also sets up a return entry point at Location 8 of the OS/360 partition for the Hypervisor's Wait routine (Step 3) in the RAX partition. Then, it sets the PCR Bit 7 and those for the I/O channels dedicated to RAX to 1 and transfers control of the CPU to the RAX IPL routine in the high-core mode. (Subsequent SPB instructions alter the status of the PCR Bit 7 only, and these I/O channel mask bits of the PCR remain unchanged until the system is shut down.)

2. The RAX IPL routine simulates the IPL to load RAX containing the Hypervisor's Wait routine into the upper half of the core storage. The RAX IPL routine is overlayed: RAX, then, starts its operations in the high-core area with the I/O channels dedicated to OS/360 disabled.

3. When RAX encounters a wait or a forced I/O loop condition, the Hypervisor's Wait routine switches the CPU prefix to the low-core mode and transfers control of the CPU, across the partitioning, to the Hypervisor Task in the OS/360 domain (via Location 8). The return address in the Hypervisor's Wait routine (Step 5) in the RAX partition is passed to the Hypervisor's I/O Interrupt routine in the OS/360 partition in a general register as a parameter. The Hypervisor Task issues the wait-macro instruction causing OS/360 to start processing tasks in the OS/360 queue with all the I/O channels enabled.

4. The Hypervisor's I/O Interrupt routine intercepts all I/O interrupts while the CPU is operating in the OS/360 mode. It determines if the interrupt is caused by an I/O channel dedicated to RAX. If not, control

is relinquished to OS/360's I/O interrupt handling routine, if the I/O interruption is caused by an I/O channel dedicated to RAX, it switches the CPU prefix to the high-core mode after setting up the return entry point at Location 8 for the Hypervisor's Wait routine (Step 5), and returns control of the CPU to the Wait routine, which fakes the I/O old PSW in the RAX partition so that it appears as though an I/O interrupt occurred in the wait state in RAX, and transfers control of the CPU to RAX I/O interrupt processor.

5. When a wait or a forced I/O loop condition is encountered in the RAX mode, Hypervisor's Wait routine switches the CPU prefix to transfer control of the CPU to the Hypervisor's I/O Interrupt routine (via Location 8) in the low-core mode. The Hypervisor's I/O Interrupt routine, in turn, returns control of the CPU to OS/360 at the point of the last I/O interruption in the OS/360 domain. The procedure now repeats itself at Step 4.

Hardware Requirements

1. RPQ Storage Shared Program, which provides the Prefix Control Register and the Set Prefix and Branch instruction as described below:

   a. Prefix Control Register

      The Prefix Control Register (PCR) contains eight bits: Bits 0 to 6 of the PCR are assigned to the I/O channels 0 to 6 respectively, and Bit 7 of the PCR is assigned to the CPU. When a bit in the PCR is set to 1, all core storage requests by the associated unit are prefixed to the upper half of the core storage by hardware. Core storage accessible to a unit is limited to one half of the total

core storage of the system if any bit in the PCR is 1. For example, if an address generated by the CPU is greater than 256K and the PCR Bit 5 is 1, a program interrupt (addressing exception condition) will take place. (A system with 512K core storage is assumed for the convenience of exposition.)

b. Set Prefix and Branch Instruction

The Set Prefix and Branch (SPB) instruction is privileged and has the SI format with an op-code of A7. The SPB instruction causes the bit pattern of the immediate data, I_2, field to be moved into the PCR, and an unconditional branch to the address specified by the B1/D1 field of the instruction. The branch address is prefixed to the upper-core if the bit 7 of the immediate data field of the SPB instruction is 1. If the branch address specified by the instruction is greater than 256K and any bit in the immediate data field of the instruction is 1, a program interrupt will occur.

c. At the I/O initiation (SIO) time, the CAW is prefixed (256K + 72) if the PCR Bit 7 is 1, and CCW and data address are prefixed if the PCR bit assigned to the channel referenced by the SIO instruction is 1. Since a core storage request is made by the channel for each unit of data transferred to or from the core storage at the time of the data transfer, the status of the PCR bit assigned to a channel cannot be changed while a data transfer operation is in progress on the channel. Consequently, an I/O channel cannot be concurrently shared by RAX and OS/360.

At the I/O interrupt time, the CSW is prefixed (256K + 64) if the PCR bit associated with the channel is 1, and the I/O old PSW and I/O new PSW are prefixed if the PCR Bit 7 is 1.

2. Since no I/O channel can be concurrently shared by the two systems, the I/O configuration must contain sufficient I/O components to support RAX and OS/360 at the same time, a minimum I/O configuration includes one multiplexor channel, two operator's consoles, and two selector channels.
3. Two IBM 2365 Storage Units.

Comments

1. RAX and OS/360 will maintain separate timers.
2. The I/O channels dedicated to OS/360 will be so indicated to OS/360 at System Generation time. The remaining I/O channels will be marked non-existent in OS/360.
3. Recovery capabilities for a catastrophic error in either partition will not be provided.

Restrictions

1. The I/O channels dedicated to OS/360 remain disabled while the CPU is operating in the RAX domain, and the I/O channels dedicated to RAX remain enabled while the CPU is operating in the OS/360 domain.
2. No program in RAX and OS/360 executes the Set Prefix and Branch instruction in order that the partitioning may remain opaque to both systems. Control of the CPU is transferred across the partitioning from one system to another solely by Hypervisor.

APPENDIX D

3. The Hypervisor Task must be the first task of OS/360 after IPL.

References

1. RPQ Transmittal No. E48992, Storage Shared Program, dated June 5, 1967.

2. Functional Specification, RPQ E48992, Storage Shared Program, dated May 16, 1967, by Mr. Ray Newton.