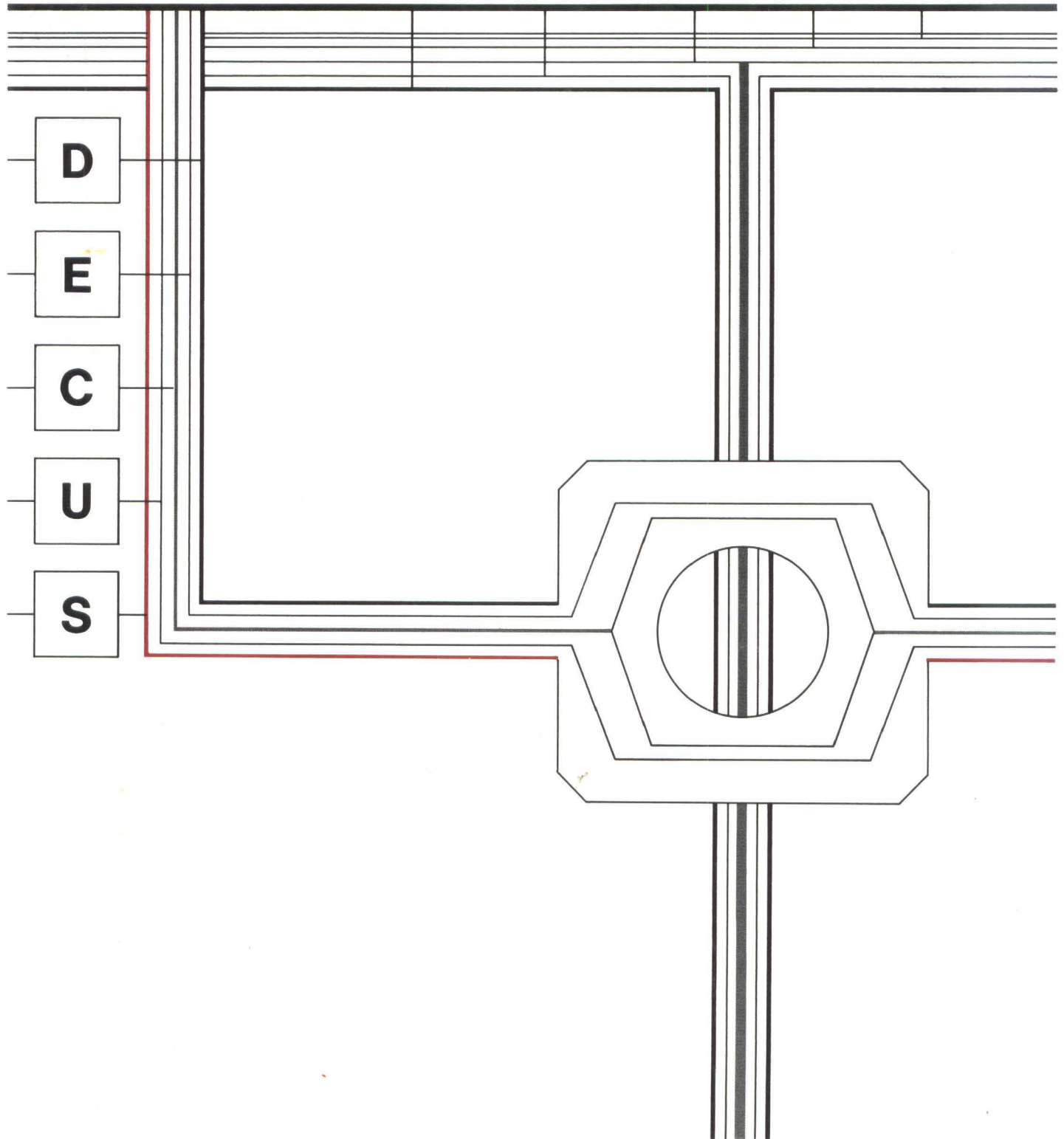


USA

1985 SPRING

PROCEEDINGS OF THE DIGITAL EQUIPMENT USERS SOCIETY





**PROCEEDINGS
OF THE
DIGITAL EQUIPMENT
COMPUTER USERS
SOCIETY**

**Presentation and Reports
USA Spring 1985**

**New Orleans, Louisiana
May 27 - 31, 1985**

"The Following are trademarks of Digital Equipment Corporation"

ALL-IN-1	Digital logo	RSTS
DEC	EduSystem	RSX
DECnet	IAS	RT
DECmate	MASSBUS	UNIBUS
DECsystem-10	PDP	VAX
DECSYSTEM-20	PDT	VMS
DECUS	P/OS	VT
DECwriter	Professional	Work Processor
DIBOL	Rainbow	

Copyright ©DECUS and Digital Equipment Corporation 1985
All Rights Reserved

POLICY NOTICE TO ALL ATTENDEES OR CONTRIBUTORS "DECUS PRESENTATIONS, PUBLICATIONS, PROGRAMS, OR ANY OTHER PRODUCT WILL NOT CONTAIN TECHNICAL DATA/INFORMATION THAT IS PROPRIETARY, CLASSIFIED UNDER U.S. GOVERNED BY THE U.S. DEPARTMENT OF STATE'S INTERNATIONAL TRAFFIC IN ARMS REGULATIONS (ITAR)."

DECUS and Digital Equipment Corporation make no representation that in the interconnection of products in the manner described herein will not infringe on any existing or future patent rights nor do the descriptions contained herein imply the granting of licenses to utilize any software so described or to make, use or sell equipment constructed in accordance with these descriptions.

The articles are the responsibility of the authors and therefore, DECUS and Digital Equipment Corporations, assume no responsibility or liability for articles or information appearing in the document.

The views herein expressed are those of the authors and do not necessarily express the views of DECUS or Digital Equipment Corporation.

Ada is a trademark of the U.S. Government, XEROX is a trademark of Xerox Corporation, IBM, PROFFS are trademarks of International Business Machines Corporation, UNIX is a trademark of AT&T Bell Laboratories, CP/M, PL/I are trademarks of Digital Research, Inc., MSDOS is a trademark of Microsoft Corporation, TSX-PLUS is a trademark of S&H Computer Systems Inc.

FOREWARD

This Proceedings is published by DECUS (Digital Equipment Computer Users Society), a world-wide society of users of computers, computer peripheral equipment and software manufactured by Digital Equipment Corporation. The U.S. Chapter of DECUS has approximately 51,000 active members.

DECUS maintains a library of programs for exchange among members and organizes meetings on local, national and international levels to fulfill its primary functions of advancing the art of computation and providing a means of interchange of information and ideas among members. Two major technical symposia are held annually in the United States.

For information on the availability of back issues of Proceedings as well as forthcoming DECUS symposia, contact the following:

**DECUS U.S. Chapter
Digital Equipment Corporation
219 Boston Post Road, BP02
Marlboro, MA 01752**

All issues of past Proceedings are available on microfilm from:

**University Microfilms International
300 North Zeeb Road
Ann Arbor, MI 48106**



This volume of the Proceedings contains papers which were presented at the Spring 1985 Symposium of the Digital Equipment Computer Users Society.

The Spring 1985 Symposium was held at the Convention Center and Riverside Hilton in New Orleans, Louisiana. Five thousand two hundred and eighty seven members of DECUS (out of a membership of approximately 51,000) attended over the week of May 27 to May 31, 1985. One thousand two hundred fifty also attended 48 Pre-Symposium Seminars on Sunday, May 26. They attended 71 birds-of-a-feather sessions, and over 915 regularly scheduled presentations. Add all of that to thousands upon thousands of hours of discussions in seminar rooms, hotel rooms, corridors, on the Mississippi and in the French Quarter to approximate the magnitude of the input source for this output document.

Avis, the car rental company, has a rather famous slogan — "We're Number Two. We Try Harder." They recently updated it slightly. "We Try Harder, Faster." Digital Equipment Corporation might well be following Avis closely, for they too are number two, and they too are trying harder faster, with the introduction of the new 8600 processor, the fastest Digital computer yet. And at DECUS, we follow Digital quite closely, trying harder to be faster, making use of new computer technology, decision support systems, and office automation to reduce costs and increase productivity of volunteers and staff alike. But there is one area where we're trying especially hard to be extremely fast, publications.

You will notice as you peruse this Proceedings that the papers are reprinted as received from their authors in camera-ready form. Why are not the Proceedings, and the Special Interest Group newsletters, entered into a Digital computer and typeset? The answer to this harkens directly back to Avis' new slogan. Faster. It is the goal of each editor of each publication to get information out to the reader as quickly as possible. Data entry and typesetting would add days and weeks to the time between submission and printing. Certainly, these delays could be reduced, but at a very high price, and low cost is one of the major goals of the DECUS Communications Committee in disseminating information to the DECUS membership.

At this point, the time between an author's submittal of a paper for consideration, acceptance, delivering of the paper at symposium, and receipt of the Proceedings by attendees is about nine months. Compare

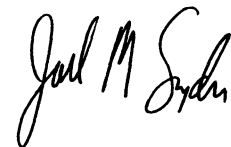
that to some ACM and IEEE journals, which can take 3 years between acceptance and publication. Thus, we sacrifice cosmetics in favor of speedy delivery of information in an environment where information is the most valuable commodity.

The Special Interest Group newsletters are also reorganizing themselves, partly to increase the timeliness of the information they distribute. Now, the worst case between submission and printing will be about sixty days, where before four months could pass. As a society devoted to disseminating knowledge and information, DECUS is getting better and better, and faster and faster.

One obvious problem is that papers accepted and published as quickly cannot go through a lengthy review and editing process. This reduces our credibility in the eyes of some observers, but the timeliness of DECUS publications simply does not allow a thirty-six month turnaround time. Digital has released and obsoleted entire operating systems in less time. As DECUS members, we simply cannot wait.

Even though the computer and information industries are now going through a highly publicized slump, the cost of equipment continues to drop, and as computerized typesetting equipment, optical character readers, and laser printers fall, you, the user, can expect to see the quality of the cosmetics of DECUS publications rise.

The symposium which produced this document was put together by hundreds of volunteers — the Symposium Committee, the DECUS staff, and all of the Special Interest and Local User Groups. But especially hard-working and visible are Jeff Jalbert and Dorothy Geiger. My thanks on behalf of all 5,300 attendees to both of them. The work of the meeting administrators in Marlboro, led by Nancy Wilga, is unparalleled in the world of user groups. Our appreciation for their time and energy is sincere. In communications, led by DECUS staff member Judy Arsenault and volunteer Mark Grundler, my greatest thanks must go to Cheryl Smith, my colleague, who publishes the Proceedings, for her efforts and assistance.



Proceedings Editor
DECUS U. S. Chapter Publications Committee



Table of Contents

	Page		Page
BUSINESS APPLICATIONS SIG		DATATRIEVE SIG	
Project Management in the New Mini/Micro World Raymond J. Doubleday	3	Use of Domain Tables to Connect Interactive and Batch DATATRIEVE Elliot F. Jaquith, Jr.	121
DATA ACQUISITION, ANALYSIS, RESEARCH, AND CONTROL SIG		DATATRIEVE Record Definition Workshop Bart Z. Lederman	127
Using the DRE-11 to Solve Real Time Data Acquisition Problems on a VAX Mark Silverstein	13	EDUSIG	
Developing a Multiprocessing Construct with a Structured Language Stevan Leonard	17	Developing an Applications Library on the VAX-- Some Observations John M. Anderson	145
Cephalometric Analysis of Facial Growth using the PDP-11/23 and VAX Mary Lou Naegele, Herbert J. Gould	27	Introduction to Microcomputers for Adults Richard L. Kopec	149
Investigation of Interrupt Response Times of PDP-11/ 44 and PDP-11/23 Computers Programming in FORTH for CAMAC Interfaces J. R. Birkelund, J. A. Abate, T. S. Lund	33	GRAPHICS APPLICATIONS SIG	
Laboratory Information Management System for Lubricant Analysis Andrew M. Wims, Ching Po Wang, Bernard E. Nagel	39	Low Cost Terminal Options for Digital Equipment Users Charles S. Janik	155
Enhancing the DTC11-EM Through Software Communication Jean M. Lareau	45	Use of an Interactive Videodisc-based Retrieval System for Archival Management, Computer-Based Instruction, and Public Information Patricia K. Mansfield, Michael K. Mansfield	169
BARS- A Behavioral Acquisition and Research System B. Johnson, M. Yochmowitz, G. Brown	49	HARDWARE AND MICRO SIG	
DATA MANAGEMENT SIG		The LA-100 as a Shared Resource Richard G. Fulton	175
Artificial Intelligence: What It Is, Where It Has Been, and Where It Is Going Terry C. Shannon	55	LANGUAGES AND TOOLS SIG	
Data Management for High Energy Laser Systems Ramon A. Tenorio, David Dayton	65	A Microprocessor Cross Development Environment Clifford J. Schornak, II	185
Encryption for Beginners Bart Z. Lederman	71	LARGE SYSTEMS SIG	
A Radiation Therapy Patient Information Management System Theodore J. Smith, Jill M. Baren, Robert F. Curley	83	TOPS-20 Question and Answer Besty Ramsey	201
Criteria for Selecting Your Relational Database Jeffrey S. Jalbert, Keith W. Hare	97	VMS for TOPS Users -- Program Development Jack Stevens	205
Bar Coding for Inventory Control Larry R. Creel	105	TOPS-20 System Directions Don DenTandt	211
Creating Menu-Driven Systems Using FMS and VAX DCL Brian D. Lockrey	111	TOPS-10 Novice Question and Answer Jack Stevens	213
		TOPS-10 Monitor Directions Susan M. Lamaestra	215
		Reading Foreign Tapes on a DECSYSTEM-20 Besty Ramsey	217
		TOPS-20 Utility Closet Steve Attaya	219

	Page
MG-10/MH-10 Memory Upgrade and Multiport Internal MOS Memory	
Donald A. Kassebaum	221
TOPS-20 Versions 6.1 User's Panel	
Peter B. Galvin	223
Managing a Large Multisystem Site -- A Case Study	
Michael D. Joy	225
TOPS-10/TOPS-20 and Integration Documentation Status	
Susan Porada	247
TOPS/VMS Performance Comparisons	
Thomas P. Blinn	249
TOPS-20 V5.1 to V6.1 Technical Comparison	
Peter B. Galvin	267
OFFICE AUTOMATION SIG	
ALL-IN-1/WPS-PLUS Documentation	
Directions	
Sue Ellen Franklin	271
Office Automation -- Beyond the Information Spectrum	
Myron K. Hayashida	277
Development of an In-House Training Program for ALL-IN-1	
Nancy R. Pflanz	287
MUMPS SIG	
A Walk Through the Forest -- How to Fix Your MUMPS Trees	
Denise Simon	297
MUMPS Programming Standards or How I Stopped Worrying and Learned to Love MUMPS	
Robert C. Richardson	313
NETWORKS SIG	
Data Interchange Between An IBM Mainframe and Digital Minicomputers	
Leonard J. Moriarty	323
A CAMAC-LSI Network	
R. Friesen, A. Simmons, J. Helton, R. Schell	329
Techniques for Protocol Validation	
William T. Kramer	331
PERSONAL COMPUTERS SIG	
The Professional-350 as an Intelligent Color Graphics Engine	
Arthur E. Downey	349
SITE MANAGEMENT AND TRAINING SIG	
Don't Get Burned! Computer Room Fire Protection	
Terry Shannon	361

	Page
Writing User-Friendly Documentation	
Terry Shannon	367
RT-11 SIG	
Multiprocessing and High Speed Data Communication	
Harry Haenen	375
The Disk Data Cache under RT-11	
Harry Haenen	379
Real Time Temperature Graphics Data Acquisition System Using DEC RT-11	
Donald J. Mandley	385
Experiences with Style in FORTRAN	
Robert Walraven, Ralston Bernard	393
Using an LSI-11/23 and RT-11 to Digitize Analog Tapes	
John N. Stewart	397
RSX-11 SIG	
Loadable Device Driver Data Bases in RSX-11M	
SYSGEN	
Carl T. Mickelson	405
RSX-11M Hexadecimal Command Line Numerics	
Carl T. Mickelson	411
A Real Time Multiprocessor Data Acquisition Network	
Mark Podany	419
Programming with Indirect Command Files	
Sharon Linnea Johnson	427
VAX SYSTEMS SIG	
TAE: Transportable Applications Executive, NASA's Front-end for Scientific/Engineering Programs	
Martha R. Szczur, Dorothy C. Perkins, David R. Howell	433
Mini-disaster Prevention Planning for the VMS System Manager	
Marisa Riviere	441
Advanced DCL Programming	
Richard H. Warner	445
Recovery of Lost Files from VAX/VMS Disk Structures: A Case Study	
Larry W. Ebinger	459
Tuning RMS Files -- A Case Study on VMS Indexed Files	
John W. Beyer	471
Results and Comparisons in Multiprocessing using VMS 4.0 and MA780	
Nancy E. Werner	487
Microcomputer Emulation on the VAX -- Implementation and Management of a Virtual Microcomputer System	
John J. Vasconcelos, Ali T. Diba	497
RMS Indexed File Performance	
Harold T. Glaser, Philip A. Naecker, Pamela A. Valentine, Gary Friedman	509

	Page
MACVAX Connection	
Bob Wilson	523
VAX/VMS Security Considerations	
Robert Wells	527
The Instruction Unit of the VAX 8600 -- A Pipeline Implementation of the VAX Architecture	
F. Osorio, S. Ching, M. Troiani, J. Bloem, N. Quaynor	535
In Search of the VAXINTOSH-- Customizing VMS V4.0 for DCL Windows	
James G. Downward	549
Designing Reliability into the VAX 8600 System	
William Bruckert, Ron Josephson	557

POSTER PAPER

Implementation of a Local Area Network at Los Alamos Meson Physics Facility	
Anthony M. Gonzales	565



**DATA ACQUISITION ANALYSIS
FOR RESEARCH AND
CONTROL SIG**



PROJECT MANAGEMENT IN THE NEW MICRO/MINI WORLD

Raymond J. Doubleday
Advanced Technology, Inc.
2 Union Plaza Suite 103
New London, Connecticut 06320

1.0 INTRODUCTION

There are over 40 Project Management software packages currently available on the market. These packages range from the very simple and inexpensive, capable of handling only 50 events at a cost of \$80, to the sophisticated, capable of planning the construction of a space shuttle at a cost of more than \$100,000. With this wide variety of features, functions, and capabilities, selecting the appropriate system for your needs would appear to be an overwhelming task.

The purpose of this paper is to focus on what these automated tools can do for you, the project manager; what to look for; how to define your requirements; and how to evaluate packages that might fulfill those requirements. I also hope to point out some of the gains you should expect from an automated project management system. Specifically, what I hope you get from this paper is:

- o An understanding of what you should look for in Project Management tools.
- o An understanding of whether or not you require automated project management tools.
- o An understanding of what features and tools you need to fill your specific requirements.

What you won't get from this paper is:

- o A tutorial on project management and project management techniques.
- o A recommendation of the "right" package for you.

2.0 BACKGROUND

2.1 History

Before beginning the main part of this paper, I would like to discuss how Project Management software has changed over the past years and what has happened in the marketplace to warrant a discussion such as presented in this paper.

We have been part of a revolution in computing power. We have gone from large mainframe computers to microcomputers and now, to what I would call super-micro or small mini-computers. Originally, Project Management software was developed on mainframe computers. These Project Management systems had enormous capacity for project management data and literally unlimited capacity for handling that information. These systems typically ran in a batch mode, which made them extremely slow in terms of user response. They required a "guru" to care and feed the system and to analyze the data that came out of it. The graphics capabilities of these early machines were limited, if available at all.

However, there was no meaningful limit to what these machines could do. ARTEMIS is an example of a typical project management system with this legacy, as is PSD from Cambridge, Massachusetts (see Figure 1).

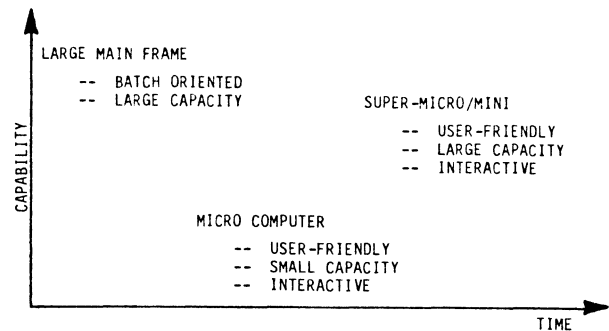


Figure 1. Automated Project Management Capabilities

However, with the advent of the microcomputer revolution (typified by machines such as the DEC Rainbow, Apple II, IBM PC, and others), we found a new kind of Project Management software. The capacity and capabilities of this software were limited; however, the packages were very friendly, easy to use, and provided immediate response for the project manager. There was no expert required to input data or interpret results; hence, the manager found a real-time decision support tool for his desktop. Typically, the graphics provided by these micros were of very poor quality (graphics were produced using either a dot-matrix or a line printer) but were sufficient to get the job done.

But, now, what do we have today? We have the super-micro, typified by machines such as the DEC Professional 350, the IBM PC XT/AT, and the MicroVAX I and II. Typically, these are the fast, powerful, single or few user machines with a large storage capacity built in. What has happened is that we have regained the data storage and speed of the mainframe computer.

Fortunately, current software has been able to maintain the user-friendliness of the micro machine. We now have real-time decision support software that is easy to use and has no realistic limitations to the quantity and complexity of data that can be handled.

The current systems are also able to generate high-quality graphics. Now we have the best of both worlds: we have a machine at the project manager's desk with the capacity of a mainframe and

can provide him with real-time, real world answers to his project management needs.

2.2 New Ideas

I would like to propose two themes for the evaluation of all tools and controls to be discussed in the remainder of the paper. These themes are abstraction and communication.

In everything that you do in a project, a software development program, or real life, it is important to be able to break the project into manageable, definable, understandable tasks (i.e., abstraction). Then, it is equally important to be able to meaningfully communicate that information.

There are three major features that should be part of the fundamental design of any Project Management package. These three features carry through the fundamental theme of abstraction and communication.

2.2.1. Abstraction. A package should support the concept of abstraction. By being able to abstract a project, you are able to take multiple-level views of your program (i.e., decomposition). Then, you can deal with it from the beginning (the Concept stage), through other successive levels of detail, down to the last possible level of detail (such as fabrication and assembly of a product).

2.2.2. Representation/communication. The choice of activities and milestones must be such that their representation on paper can be used as a means of communication. This is important because unless you can communicate the needs of the project to your staff, nothing can get done. Communication must be clear and unequivocal.

2.2.3. Manipulation. The automated tools must act on these representations of activities and milestones to ensure consistency, feasibility, and, most of all, achievability.

When looking at Project Management tools, you should look at the tools in the light of these themes as stated above.

3.0 PROJECT MANAGEMENT

This paper is not meant to be a tutorial on project management, but I would like to briefly go over what project management is to establish a common framework. The point of this paper is to highlight the benefits of automated project management and the gains that are achievable through the use of project management.

A project can be broken down into five major phases: Conception, Planning, Scheduling, Monitoring, and Action. Very often action involves the replanning and rescheduling of activities, as shown in Figure 2. We will look at each phase in detail from the point of view of Project Management systems.

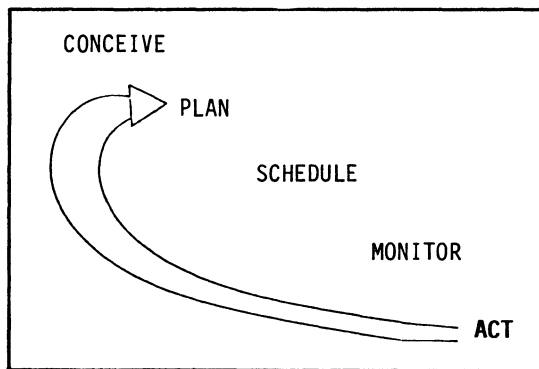


Figure 2. Project Phases

Project Management systems have two major functions. They can be used either as tools or as controls. As tools, they help you to organize, plan, and schedule; as controls, they monitor progress of the program (in terms of time and money). Tools help you to plan; controls tell you if your plan is working. If you are evaluating a feature of a Project Management system to be used as a tool, you should ask yourself how it will help you to plan your project; if you are evaluating a feature to be used as a control, you should ask yourself how it will help you to monitor your job.

3.1 Concept

The first phase of the project is the Conception phase. This is the definition of the program or the project and, in fact, becomes its charter. There are certainly no computers here; this is where insight, intuition, and depth of human understanding play a part in defining the project, its goals, and its requirements. This is where the goals of the project are established and the tempo of the program set.

3.2 Planning

3.2.1 Work Breakdown Structure. The planning stage is the decomposition of the project as conceived into its logical structure. In the initial planning stage, no schedule or resources have been assigned yet.

Top view planning. This is the first a computer Project Management package should be able to do something for you. First of all, it should support multiple views of the project and secondly, have the capacity to move down the project in detail. This is analogous to a top-down step wise refinement of the project. This is a place where the concept of being able to abstract a project or to push down the details of the project becomes very important because what you want as a project manager is to deal with a larger picture first and then to fill in the details of each phase. In essence, you are creating a management outline for your project managers to complete; and they in turn may provide the same sort of outline to their subordinates.

Let's look at what a typical software development project might look like as shown in Figure 3.

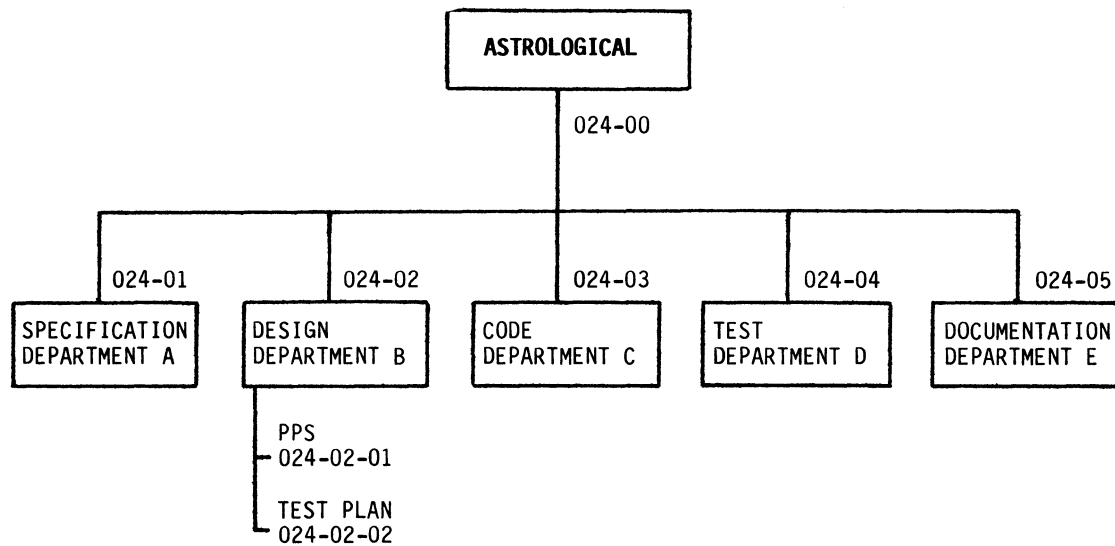


Figure 3. Astrological Organization

This is the development of a program called Astrological to analyze digitized images of the night sky. The product breaks down into typical software development components. The specification, design, coding, testing, and documentation. The package is meant for in-house use; therefore, the manufacturing and marketing functions are not included on this particular product. After the concept development, the next thing that the senior manager must do is to assign responsibility for each of these major phases to a person or department and then produce a rough schedule or goals for the project completion. Once this preliminary schedule and assignment have been achieved, the senior manager will ask the department managers to produce their own schedules, budgets, and resource requirements within the limits of their schedule.

How do you do this? You do this by having a project management package that supports various levels of hierarchy. One way to do this is through the use of work breakdown structure numbers, although there are a number of other schemes that may work equally well. Briefly, work breakdown structure is a hierarchical numbering system similar to the concept of a work outline where the order and the number that each work assignment has has meaning. Typically, a work breakdown structure number is associated with the concept of a work package, which is the smallest measurable unit of work. In our example, Figure 3, the Astrological analyzer is given the number 024. This code indicates that this particular software product is one of at least 24 different jobs that are taking place or have taken place within the organization. Looking underneath that, we see that the number 024-02 is the design function for Job 024. Looking at the preparation of the program performance specification is given the number 024-02-01. Development of the test plan is given the number 024-02-02. It is possible, of course, for this numbering scheme to continue down in more detail as required within each function and, of course, to go across to support more than the five functions shown here.

Why is this work breakdown structure important? It is important for two reasons. First of all, it

allows you to assign responsibility and a budget for a category of work such as the specifications 024-01 to Department A for completion. Secondly, it allows you to isolate your view of the project to the higher level. From now on, you as senior manager, will only be looking at things down to the second level; that is, you will be looking at tasks 024-01, 024-02, etc., leaving the specific details of the project to the managers of each of those particular departments. Your management, in turn, may look at Jobs 022, 023, and 024 to supervise the overall performance of the departments.

The next thing you should look for in a Project Management package is the ability to support various levels of hierarchy through the use of work breakdown number structuring or other means.

3.2.2 PERT/CPM. Now that we have established the major phases of the program, we need to go into more detail on how the Astrological program can be realized. The major tool you have available for this is network analysis. Network analysis is also known as either PERT (Program Evaluation and Review Technique) or CPM (Critical Path Method).

PERT/CPM are synonymous today; we will use the term network analysis to stand for a combination of PERT and CPM. The idea behind network analysis is to represent a complex project as a series of interconnected activities that must be performed. The description of the project is then used to analyze the project and answer the following questions.

- o How long will the project take?
- o Which jobs are most critical to the project?
- o How should the project be scheduled?

An activity is a time/resource consuming event in the project. I will use the arc in my discussion to represent an activity. A point in time corresponding to the start or completion of an activity is a milestone; they are represented by a triangle on the schedule. On a network drawing, they are represented as nodes or circles (note, however, that all nodes are not necessarily milestones).

Now that we have our network drawing, what other kinds of planning tools are available? Next is a Gantt chart, shown in Figure 7.

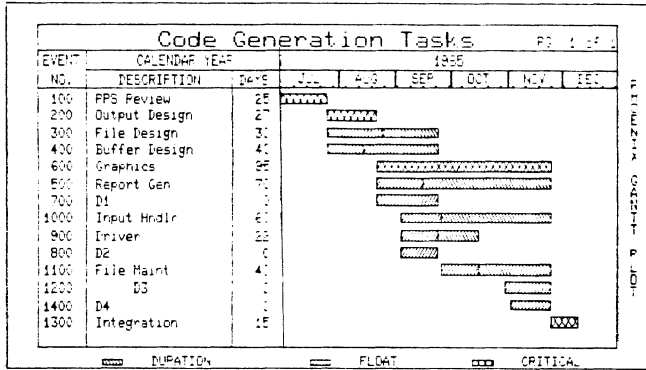


Figure 7. Gantt Plot

The Gantt chart is the first depiction of a schedule. The critical activities are shown in red on the Gantt chart as they were shown in red on the network drawing. Additionally, there should be a number of tabular reports provided with the network analysis to bring out the necessary detail in order to properly analyze the schedule. The kind of reports that you should expect to see again support the ideas of abstraction and decomposition, and are listed in Figure 8. There should be an executive summary, something that provides an overview of the time and resources consumed for the project, and a variety of reports getting down to a final detailed report showing for each of the activities the time estimates, the scheduled early start and late start, the early finish and late finish dates, as well as the float, the slack time, and the identification of the activities and resources that are critical to the time of completion of your task.

- o DATA SUMMARY
- o EXECUTIVE SUMMARY
- o DETAIL REPORTS
- o CRITICAL PATH REPORT

Figure 8. Network Analysis Management Reports

This in essence becomes the plan for your project. However, it is necessary now to generate a firm, fixed schedule or baseline.

3.3 Schedule

The Gantt plot is a candidate schedule. What you must do is use it to develop a firm, fixed schedule or baseline. The final schedule represents the plan of the Gantt Plot, with real-world constraints applied to the plan. This schedule is one that you will manage to and report on. All your progress will be measured against this baseline schedule. The schedule for the coding effort is shown in Figure 9.

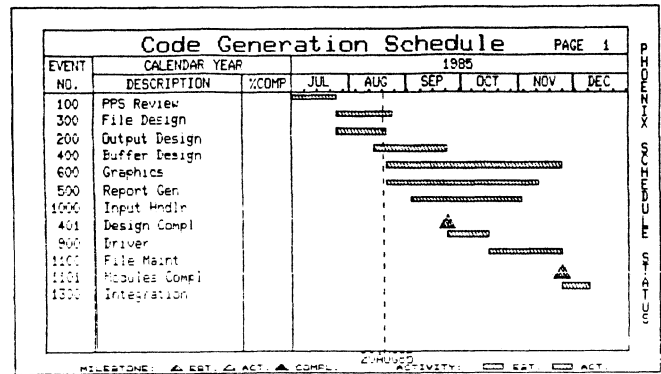


Figure 9. Schedule Plot

3.4 Monitor

3.4.1 Controls. You have passed the planning stage. Now that you have established a schedule, you need to have a number of automated project controls that will let you examine the schedule and examine the financials for your project to make sure you are both within budget and on schedule. Before we begin discussion of some controls you should look for in a project management package, let's take a look at our project.

First of all, the job spans five departments. The initial time estimates were that the job would take two years to complete, cost \$1.5 million dollars, and would be composed of approximately 5,000 separable and discrete activities. Given this size, how are you going to control it? Well, taking a step back, you have to look at why you are a project manager. Most likely, it is because of your ability to thoroughly understand your job and to almost be intuitive about the nature of the work you do. A project of the magnitude of Astrological would require a database so large it would negate your ability to be intuitive. What you need from a project management package is the ability to be dynamic in monitoring your project to be able to develop various views downward into the database until you can focus on the issues that are pertinent to the project. You need to be able to select or segment the database so that you can get an accurate, concise view of a limited segment of the database.

Again, this supports the concept of abstraction. You want to be able to look at the data in varying degrees of detail; only the detail necessary to give you the insights that you need to do your job. Your Project Management controls should provide unlimited query capability on the database.

3.4.1 Schedule Status. The first thing you should look at is the schedule. This is shown in Figure 10, which is a schedule with milestones for monitoring the progress of each task.

In this particular example we are showing a graphic depiction, one that is very important and gives a quick indication of how we are doing and where we should be today for the project. As you can see, immediately below the baseline schedule is the actual start and completion of each of the activities in the project as well as percentage complete. The percentage complete for each task is indicated by how much of the lower bar is filled in.

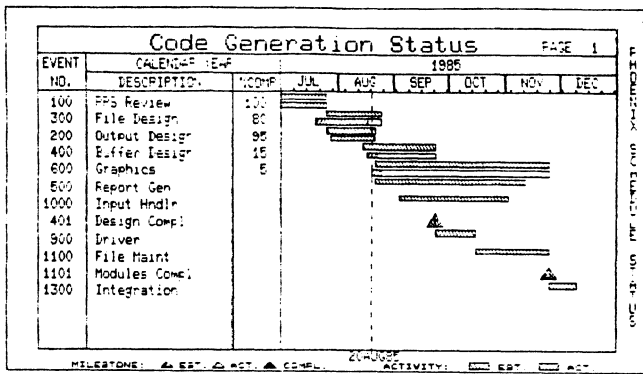


Figure 10. Schedule Status Plot

3.4.2 Completion Status. The second chart, Figure 11, is a Completion Status Plot which gives us another view of the data. It indicates which events are early, which events are late, percentage complete, and how many days remain until the completion of the job.

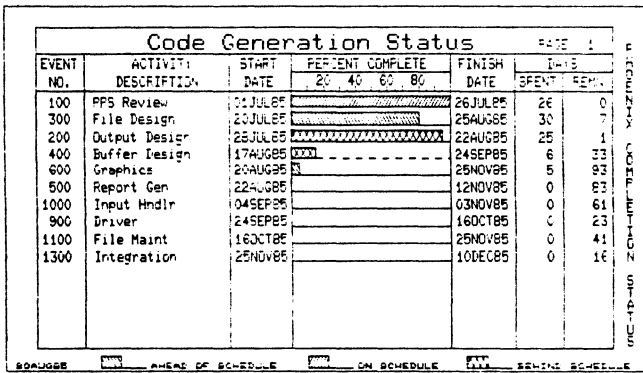


Figure 11. Completion Status Plot

3.4.3 Cost Status. Figure 12 shows a Cost Plot, which is a measure of the budget, the dollars spent, and the work achieved for those dollars spent.

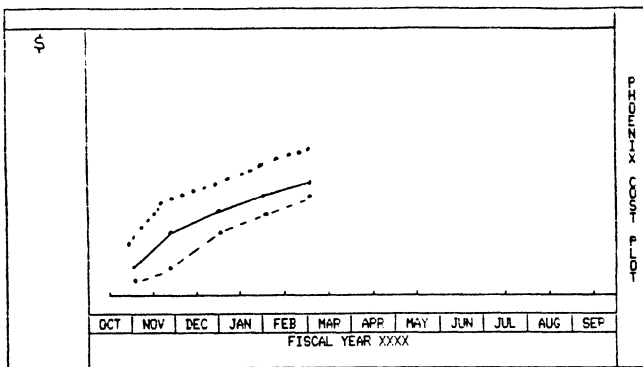


Figure 12. Cost Plot

This plot gives you a feel for the rate at which the funds of the project are being used and the amount of work that is actually being performed for your project. This brings up a number of ideas, such as the budgeted cost of the work scheduled, the budgeted cost of the work performed, and the actual cost of the work performed.

3.4.4 Cost Variance. Figure 13 shows a Cost Variance Plot. It is the difference plot of the data that was previously shown in Figure 12 and gives us a measure of how well we are progressing against the schedule. The closer these curves are to zero, the more accurate were our project predictions and the better our project performance.

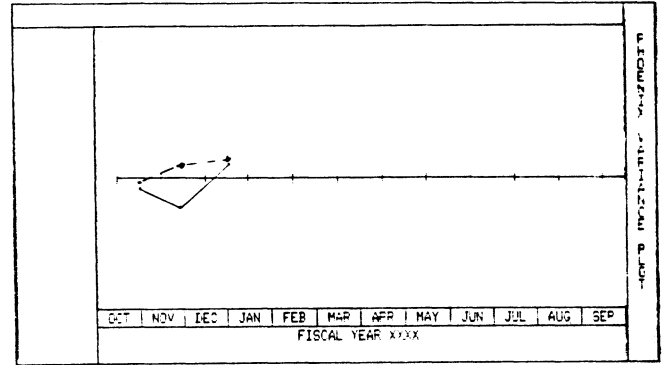


Figure 13. Cost Variance Plot

3.4.5 Communication. What is significant is that the previous four figures provide accurate and timely information that may be communicated easily. Large stacks of computer runs are not required, and it is not necessary to connect dots and asterisks because the plot was prepared on a printer. The control reports provided are presentation-quality graphics.

The monitoring tools that you should select should be suitable for all levels of management. In management reporting, you certainly don't want to have separate tools for different levels of management. What you should expect from your project management tools is that for high level meetings, briefings, and presentations, they should support full-color graphics with figures that are easy to read and understand. They should be crisp and to the point. For reports, figures should be done in black and white so they can be clearly reproduced by either printing or copying.

Your project controls should also support graphics with tabular reports which contain all necessary back-up data.

3.5 Act

Management must manage. Now that you have read the reports, reviewed the project data from your subordinates, you must identify the causes of any problems and act on them. Therefore, it is very important that the project management system you select be able to perform what-if analyses to aid in replanning and redefining the project as it progresses.

At this point, the idea of representation of the ideas and their automated manipulation becomes very important. You must be able to easily manipulate the parameters of your program and perform rapid what-if analyses until you have developed an adequate approach to your problem. You must then be able to modify schedules to accommodate this replanning just as easily. With replanning, the cycle begins again.

4.0 CONCLUSIONS

The latest generation of Project Management software has the power and capacity of main-frame type packages and the ease of use of micro-computer software.

Any Project Management system you select should:

- o Be easy to use.
- o Support multiple views of the database (abstraction).
- o Provide presentation-quality graphics (communication).
- o Provide real-time analysis (monitor).
- o Support rapid what-if analyses (plan and replan).

BASIC SIG



USING THE DRE-11 TO SOLVE REAL TIME
DATA ACQUISITION PROBLEMS ON A VAX

Mark Silverstein
Goddard Space Flight Center
Greenbelt, Maryland

ABSTRACT

When the rate of input data exceeds the worst possible interrupt latency times, a data acquisition system loses dependability. The DRE-11 alternate-buffered DMA interface can remove this difficulty. This paper enumerates the capabilities of the DRE-11, explains any difficulties in installing the driver, demonstrates a QIO instruction to utilize the device, and explains how the DRE-11 solved real time data problems in Goddard Space Flight Center's Laboratory for Extraterrestrial Physics.

THE ORIGINAL SYSTEM

I serve in the Laboratory for Extraterrestrial Physics's Information Analysis and Display Office. We provide computer support to the various groups in the lab. One such group is the Planetary Astrophysics Section which, among other things works on a project called SIRIS. The Stratospheric InfraRed Interferometer Spectrometer is a balloon experiment that is launched into the middle atmosphere to study the ozone layer and that which tends to decay ozone. The experiment needs real time computer support. Our processor serviced all the experiment's interrupts in a timely fashion. The displays flashed on the screen while the bytes traveled through the data line to deliver the next scan. Our Test and Formatting System served us well. The scientists smiled, and I knew peace.

This Test and Formatting System consisted of a MODCOMP II with a Diablo disk drive, an AMPEX tape drive, and a Teletype machine as its console terminal. After ten years of noble service, the system became unreliable and the scientists frowned. The company that made the Test and Formatting System no longer existed. When the tape drive went bad, we called AMPEX. They assured us the drive was fine and we had a MODCOMP problem. When we called MODCOMP, they assured us we had an AMPEX problem. We could not keep the system in working condition. I sought a replacement system.

We needed a system to which we could make an easy transition. Users would have to get on the machine without a great deal of retraining. Software, for the most part should not involve rewriting. Finally, the system had to be portable. After all, we needed the ability to test our experiment in the lab and in a high bay area while it sat in a gondola during flight preparation. We also had to be able to ship the machine easily to the balloon facility across the country where the system would be used to receive and analyze data during flight.

As so much of the lab's experience and software were VAX oriented, we decided the replacement system should be a VAX. We had a great deal of existing software we could use, and we would be familiar with the new system to which we would convert the Test and Formatting System's data acquisition and analysis programs. The VAX-11/730 fulfilled our requirement for portability, and with the floating point accelerator, compared well with our MODCOMP II.

REAL TIME PROBLEMS

Then the bubble burst. Rumors flew. A 730 can't handle multiple processes in a real time environment. The interrupt latency time on a 730 is too great for your needs. It will never work.

The last objection I discounted. People were doing real time work on a 730. The first objection I decided to worry about later. I didn't have multiple processes on the Test and Formatting System. I wanted them on the 730, but if I couldn't get them, I would lose nothing. The

middle objection stumped me. A word came from the experiment's output line every 800 microseconds. That word had to be examined and needed to have its bits juggled before the next word came. I had heard results of tests that showed rare, worst case latency times of hardware interrupts to the VAX-11/730 that were greater than my 800 microseconds. Even though such a thing would rarely occur, once would be enough to cause me to lose data. The latency time was unacceptable.

I talked over my problem with a fellow in another lab. His hardware people were putting the finishing touches on a buffer they had built to help their VAX-11/750 keep up with data that was coming too fast for it. I did not want our hardware people to have to design and build a special buffer for this application. In the course of discussion the fellow mentioned to me that someone had told him of a new product Digital Equipment Corporation had come out with that was featured in that month's INSIGHT magazine. It was a double buffered direct memory access high speed interface that sounded as if it would get around my latency time problem. I read the article.

THE DRE-11

The article made the DRE-11 sound as if it were the solution to my problem. It was alternate buffered. I wouldn't even have to tell it when I was done with one buffer to start filling the next. It would do it automatically. This device would put my incoming data into memory at high speeds. It would take my data in 16 bit parallel form. Wonderful! I immediately sent in the card to receive more information (see fig. 1).

When I believed the DRE-11 was what I wanted, I contacted our DEC sales representative for price, ordering information, driver information, etc. Her response was simply that she didn't know what I was talking about. She'd never heard of a DRE-11. It was such a new device that the sales representatives had no information on it. Our sales rep was able to research it and came up with enough information for us to order the device, except for one minor detail. INSIGHT claimed the driver would be found in VMS. Not so. Mere weeks before I was due to begin testing the experiment on the new system, I found myself with an interface device I couldn't talk to. An emergency procurement brought me the driver quicker than I'd thought possible.

At the time I was a very novice system manager. My sole experience consisted of 1 week of system manager class and 2 months as substitute system manager on a VAX-11/780 while the real system manager designed a major software project. Thus, I shuddered at the thought of installing a device driver. It must have taken me all of 20 minutes. The installation instructions were crystal clear. I had the driver installed and tested. This gave me the ability to code several functions of the DRE-11 (see fig 2) with simple QIOs.

THE FLOW OF DATA

The READ VIRTUAL BLOCK function IO\$.READVBLK fulfilled my requirements. If it worked as

DRE-11 SPECIFICATIONS

DATA TRANSFER

16 BIT PARALLEL WORDS

DMA BLOCK OR BURST MODE

128 K WORD ADDRESSING

64K WORD MAX BLOCK SIZE

MAXIMUM TRANSFER RATE

600K WORDS PER SECOND

700K WORDS PER SECOND

BURST MODE

4 ADDRESS REGISTERS

(FIG 1)

FUNCTIONS

CANCEL I/O REQUEST

WRITE BLOCK

READ BLOCK

SET DIRECT DATA PATH

SET BUFFERED DATA PATH

READ WORD

WRITE WORD

SET FUNCTION BITS

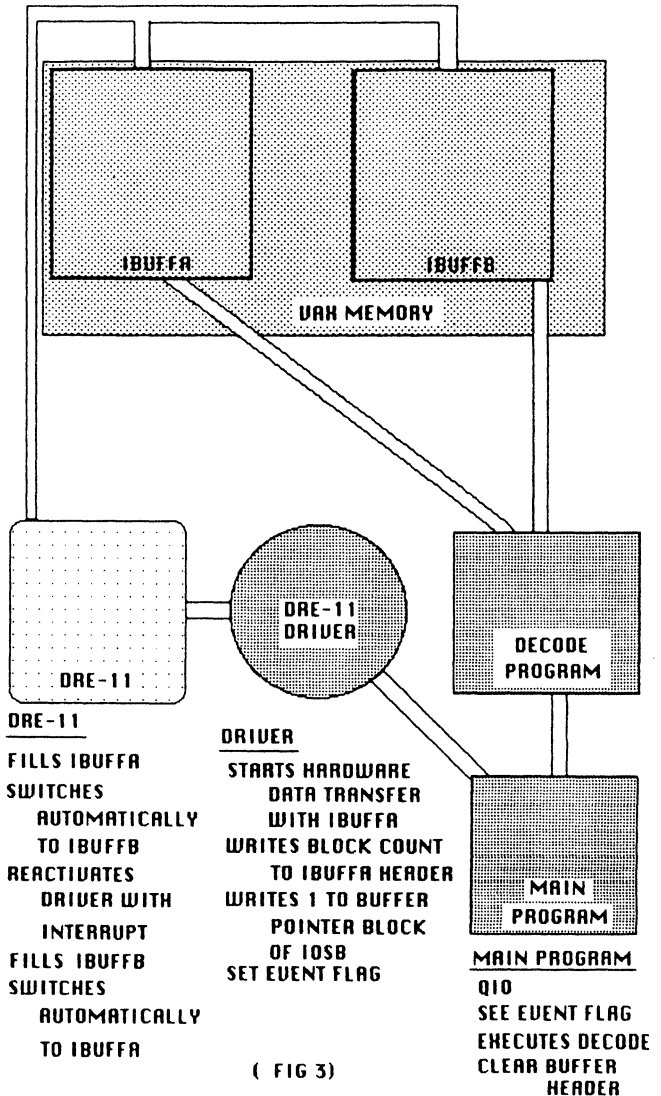
READ STATUS BITS

CONNECT TO UNSOLICITED

INTERRUPT

(FIG 2)

advertised, it would fill a buffer in memory with incoming data and immediately begin to fill another buffer while my software could play with the data in the first buffer. Best of all was that it claimed to do this with little more than one QIO instruction. Let me show you (see fig. 3) what goes on when you execute this QIO. The driver begins the DRE-11's data transfer by ordering the DRE-11 to fill my IBUFFA, where IBUFFA is my first buffer in memory. My next instruction after the QIO is a wait for event flag (see fig 4). I know real time programmers hate waiting for anything, but while you are waiting, the VAX's memory is actually acquiring the data you want. When the DRE-11 has filled IBUFFA it automatically begins to fill IBUFFB (my second buffer in memory) and issues an interrupt to



reactivate the driver. Now that I am dealing with time to spare on the order of 6 and a half seconds instead of 800 microseconds, I don't care if I have a little latency time here in reactivating the driver. The driver puts the current block count, (how many blocks or buffers I've filled - 1 in this case) in the buffer's header, writes a 1 into the buffer pointer part of the IOSB (indicating which of the 2 buffers I've most recently filled and sets the event flag. I've been waiting for that event flag. It tells me that I have a buffer full of data in IBUFFA. I clear the event flag, and go off in my program (see fig. 5.) to decode the data and store it on disk so it can be analyzed and displayed by another process. When I've decoded every byte

that was in IBUFFA. I clear the buffer header, as a signal that IBUFFA may be filled again. After doing that, I again wait for the event flag. Meanwhile, the DRE-11, has been busy filling IBUFFB. When IBUFFB is filled, the DRE-11 immediately begins to fill IBUFFA again. The DRE-11 once more reactivates the driver with an interrupt. The driver writes the current block count (2) into IBUFFB's header (I've filled 2 buffers), it sticks a 2 in the buffer pointer part of the IOSB (I've just filled buffer number 2) and sets the event flag I've been waiting for. I clear the event flag, and proceed to decode the contents of IBUFFB. On returning from my subroutine I clear the header of IBUFFB to allow it to be filled again, and once more wait for event flag. The scheme repeats from the point IBUFFA was filled to this point until an error occurs or until the program is terminated.

SOFTWARE COMMAND THROUGH DRIVER

One QIO commanded the DRE-11 to do all of the above. Let's take a look at it.

```
I=SYSSQIO(%VAL(1),%VAL(CHAN),
1 %VAL(IOS_READVBLK),IOSB,,
2 IBUFFA,%VAL(16384),IBUFFB,
3 %VAL(16384),%VAL(0),
4 %VAL(6.536))
```

where:

first argument, %VAL(1), indicates the number of an event flag to be set when either buffer is filled.

second argument, CHAN, is the number of the I/O channel assigned to the DRE-11.

third argument, IOS_READVBLK, is the function code to transfer blocks of data from my external experiment to the VAX's memory.

fourth argument, IOSB, is the address of a quadword I/O status block that receives the return status, an indicator telling which buffer was most recently filled, and the byte count of the buffer being transferred up to the point of termination.

first function parameter is IBUFFA, the address of my first buffer.

second function parameter is the length of that buffer in bytes, excluding a 4 byte header which is used to synchronize the program with the alternate buffering.

third function parameter is the address of the second buffer.

fourth function parameter is the length of that second buffer, again less the 4 byte header.

fifth function parameter is the number of blocks of data to be transferred. Note that I have this as a zero. A zero indicates that I want to keep on sending data until an error occurs or until the program terminates.

sixth function parameter is the maximum time allowed for each buffer to be filled. Here I told it that it should end with a timeout if it ever takes the buffer longer than 6.536 seconds to fill.

This is really a much simpler QIO to set up than those I remember being taught in system services class, and it gives you tremendous power.

WHAT DID THIS GET ME?

I began with a situation in which my VAX was choking with data, and ended up in a situation

```
IF(.NOT.I)CALL LIB$STOP(XVAL(I))
I=SYSSASSIGN(UUNAM,CHAN,,)
IF(.NOT.I)CALL LIB$STOP(XVAL(I))
I=SYSSQIO(XVAL(1),XVAL(CHAN),XVAL(IOS_READVBLK),
*IOSB,,IBUFFA,XVAL(16384),IBUFFB,XVAL(16384),
*XVAL(2),XVAL(6.536))
IF(.NOT.I)CALL LIB$STOP(XVAL(I))
I=SYSSWAITFR(XVAL(1))
I=SYSSCLEAR(XVAL(1))
```

(FIG 4)

DECODE

- REMOVE HALF WORD OF DATA FROM EACH WORD OF INPUT
- PUT HALF WORDS TOGETHER
- COLLECT WORDS TO FORM BLOCKS OF 64 WORDS
- PERFORM SYNC CHECK
- IF BLOCK CONTAINS HOUSEKEEPING - MAKE AVAILABLE TO DISPLAY
- COLLECT RECORD OF 64 BLOCKS
- STORE RECORD ON TAPE AND DISK

(FIG 5)

during which I have plenty of time. Earlier, I stated that people told us you couldn't have multiple processes in a real time environment on the VAX-11/730. We're doing it. While the program I've just described is acquiring data, decoding it and storing it in a disk file, (and on tape), a scientist is sitting at a terminal running an analysis and display program, while a third process is displaying pressures and temperatures from the input stream on another terminal. For our next flight we plan to increase the amount of data being transferred, and also increase the capabilities of our displays. With the DRE-11, I'm betting our 730 can handle it.



DEVELOPING A MULTIPROCESSING CONSTRUCT
WITH A STRUCTURED LANGUAGE

Stevan Leonard
Exel Microelectronics
San Jose, California

ABSTRACT

Executing and controlling several overlapping processes is a challenge even with the aid of a multitasking operating system. Attacking this problem from within a program is straightforward with the use of the case statement construct. A simple example using PASCAL will illustrate the means of process initiation and control, and how to code processes.

What do you do when you have a single-job operating system to solve a multiprocessing problem? Wishing and hoping offers only temporary relief. The application of a polling method to check on various processes in a round robin fashion brings some relief. Then adding in interrupt processing to form a hybrid approach adds some flexibility and lessens system overhead. One critical element is still missing — a general structure applicable to any multiprocessing problem. The construct described in this paper is one method of defining a general multiprocessing structure within a program. From there, the programmer is concerned only that the functions such as reading from the user console or printing text are performed with no-wait input-output system calls. So wish no longer and read on.

THE MULTIPROCESSING APPROACH

Execution of multiple processes by a CPU is a function usually attributed by programmers to an operating system. In most cases multiprocessing (i.e. multitasking) ought to be delegated to the operating system. But for certain applications it may be desirable to maintain control of overlapping processes from within a single program unit. This is a necessity in the case of a single-job operating system, where overlapping system resources must be handled by the program.

A multiprocessing implementation is a straightforward approach to handling the asynchronous activities of several devices. If a separate process controls each device, a process needs to respond only when the device it is tied to requires service. Any process can use information about the status of other active processes for decision making independently of the processing flow of other devices, as well as of any global processes controlling multiprocessing activities.

EMBEDDED IN THE APPLICATION

Developing the control of multiple processes within an application program provides a direct means to access and update process information, and to tailor process control to the application rather than be constrained to the general format provided

by an operating system. Even so, on a system that supports multiprocessing, embedded multiprocessing in the application program would usually be unnecessary without special requirements.

Applications written for single-job operating systems might employ a multiprocessing construct to provide a user simultaneous access to system resources. Printing documents and data capture through a modem can be time consuming. Overlapping print and data capture functions could cut the real time consumed up to 50%. But aside from efficiency, the greatest advantage to overlapped processes is that the user can continue to interact with the system to maintain control of the processing.

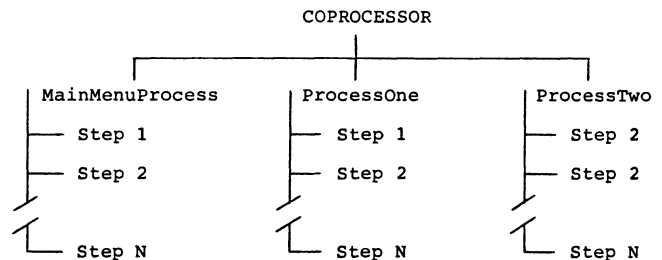


Figure 1: Code Structure in Program MULTIPROCESS

MULTIPROCESSING PROGRAM SKELETON

At the end of this paper is a listing entitled "Multiprocessing Program Skeleton" that includes the essential elements of a program that handles multiple processes. Pascal was used for this example since the construct was originally implemented in it. Structured languages that include a case statement would be equally suitable to code this construct. Though most any language can approximate the code in program MULTIPROCESS, the case statement is crucial to program legibility and the generation of uncomplicated code.

Figure 1 illustrates the code structure in program MULTIPROCESS. The coprocessor successively transfers control to active processes. These processes are defined as procedures, and broken up into steps that are activated through a case statement. The effect is that of segments of code in separate processes executing on a time-sharing basis.

COROUTINE IMPLEMENTATION

The concept of coroutines best fits this implementation of multiprocessing. Two coroutines operate by transferring control back and forth to one another at planned intervals in their sequence of code. Whenever one calls the other, processing continues wherever the other was last called. In contrast, subroutines are always entered at the top.

Variables coprocessStatus and nextCoprocess control the action of the coprocessor. The case statement selection via index nextCoprocess is included to show that processes don't necessarily have to be executed successively. If processes have varying importance, a main menu option might allow the user to establish process priorities.

Variable coprocessStatus contains the current status of each process. MainMenuProcess is always active to monitor input from the keyboard. Status of other processes is controlled by the application. The possible process statuses are defined by coprocessStatuses. Only when a process is "active" will it be entered. Process status can be altered by MainMenuProcess or another active process.

Step MMrestart in MainMenuProcess is of special importance to the functioning of the construct. If one or more of the steps in MainMenuProcess is a subprocess with its own case index, then upon completion of the subprocess, the procedure ReinitMM can be used to reset each case index associated with MainMenuProcess to its initial entry value. The MMrestart step might also be set after an error is processed. Another function of ReinitMM is to reinitialize global variables used by main menu subprocesses and global variables that depend upon calculations made in main menu subprocesses, such as variables used for timing.

MODEM PROGRAM EXAMPLE

On a system with two or more modems, it would be handy to be able to control sending and receiving files on more than one modem simultaneously. The modem program example at the end of this paper is a complete Pascal program that operates off the following main menu:

- ```

 Main Menu
1 Connect/disconnect modem channel
2 Start/stop file receive
3 Start/stop file transmit
4 View a receive/transmit process
5 Dumb terminal mode
6 End

```

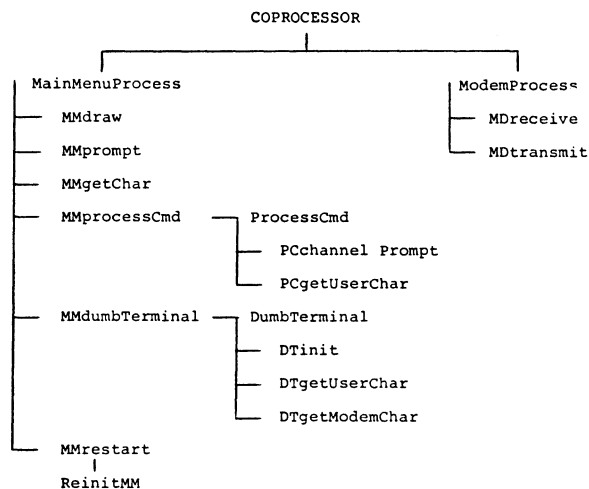


Figure 2: Code Structure in Program MODEM

The user is informed of active channels, then prompted to enter an option number. In order to start a receive or transmit process, the modem to be used must be connected. The view option causes the characters being receive/transmitted to be echoed on the user console. Dumb terminal mode can be entered on a connected modem, but not one that is performing file receive/transmit.

The code structure in program MODEM is shown in figure 2.

#### REGARDING INDIVIDUAL ROUTINES

In the coprocessor, the program name is written to the console, then a few global variables are initialized. From there on, the coprocessor looks very much like the one in the program skeleton.

MainMenuProcess has the same format as the program skeleton, but now has two subprocesses -- ProcessCommand and DumbTerminal. Note that while one of these subprocesses is running that the main menu is inaccessible to the user. The small procedure ReinitMM resets MainMenuProcess and DumbTerminal to their initial steps. ProcessCommand resets to its initial step internally.

ProcessCommand could have been implemented as two steps in MainMenuProcess, but was broken apart as a subroutine would be in a hierarchical program structure. Making PCgetUserChar a separate step was necessary to allow the coprocessor to continue executing other processes while waiting for a character from the terminal.

If more than one character was expected, then the format used in MainMenuProcess to execute ProcessCommand would be used, but the characters would be buffered in PCgetUserChar until a terminating character is received.

The DumbTerminal subprocess flips back and forth between console input and modem input. Since the user input character is written to the terminal, this shows a half duplex configuration of the modem.

Other than the argument `processNumber`, `ModemProcess` is essentially the same in format as the processes in the program skeleton. Use of `processNumber` as an index into `processStep.MD` makes `ModemProcess` into a multiple process itself. To support three modem processes would require changes to global variables to handle the third modem and adding another step to the coprocess loop, but would not affect `ModemProcess`. The process either receives characters from the modem and writes them to `modemFile`, or reads characters from `modemFile` and transmits them to the modem, until an end of file condition is encountered. When the view option is selected, each character being manipulated is written to the user console.

#### MISSING ROUTINES

The following routines are missing from the `MODEM` program example because they are system dependent. Other than the modem device handler, all of them should be fairly easy to code.

##### AttachModem

Performs the necessary functions to activate the modem channel, dial a number and establish connection. Note: This routine would have to be written as a subprocess, similar to `ProcessCommand`, in order to allow overlapping of waiting for user input characters and other processes.

##### DetachModem

Performs the necessary functions to deactivate the modem channel. See note for `AttachModem` if any user interaction is required.

##### GetModemChar

Returns the next character from the modem input buffer. Sets itself to true if a character was available; otherwise, to false. Must be no-wait.

##### Modem Interrupt Routine/Device Handler

An interrupt-driven routine to get and put characters to the modem. Must buffer input characters for `GetModemChar`.

#### NOW THAT YOU'VE SEEN IT

Implementation of this multiprocessing construct is not simple, yet the necessity to break down the application into small steps has the benefit of being self-documenting, and the code being more likely to be functionally correct the first time around. With a little practice, you'll be developing multiprocessing applications where multiprocessing never existed before.



```
0001 Program MULTIPROCESS (input,output);
0002
0003 Type
0004 coprocesses = (mainMenu,process1,process2);
0005 coprocessStatuses = (inactive,active,hold,aborted,complete);
0006 mainMenuSteps = (MMdraw,MMprompt,MMgetChar,MMcheckCmd,MMuserError,
0007 MMrestart);
0008 process1Steps = (P1step1,P1step2,P1step3);
0009 process2Steps = (P2step1,P2step2,P2step3);
0010 Var
0011 stopProcessing: boolean;
0012 coprocessStatus: array [mainMenu..process2] of coprocessStatuses;
0013 nextCoprocess: coprocesses;
0014 processStep: record
0015 MM: mainMenuSteps;
0016 P1: process1Steps;
0017 P2: process2Steps;
0018 end;
0019
0020 procedure ReinitMM;
0021 begin
0022 with processStep do
0023 MM := MMdraw;
0024 end; { ReinitMM }
0025
0026 procedure MainMenuProcess;
0027 begin
0028 case processStep.MM of
0029 MMdraw: ; { display menu }
0030 Mmprompt: ; { display prompt }
0031 MMgetChar: ; { get character from console }
0032 MMcheckCmd: ; { check for valid input }
0033 MMuserError: ; { display error message }
0034 MMrestart: ReinitMM;
0035 end; { case }
0036 end; { MainMenuProcess }
0037
0038 procedure ProcessOne;
0039 begin
0040 case processStep.P1 of
0041 P1step1: ; { code for step 1 }
0042 P1step2: ; { code for step 2 }
0043 P1step3: ; { code for step 3 }
0044 end; { case }
0045 end; { ProcessOne }
0046
0047 procedure ProcessTwo;
0048 begin
0049 case processStep.P2 of
0050 P2step1: ; { code for step 1 }
0051 P2step2: ; { code for step 2 }
0052 P2step3: ; { code for step 3 }
0053 end; { case }
0054 end; { ProcessOne }
0055
0056
0057
```

MULTIPROCESS  
01

Source Listing

15-May-1985 16:43:09  
15-May-1985 16:43:00

```
0058 (
0059 + + + CQPROCESSOR + + +
0060)
0061 begin
0062 stopProcessing := false;
0063 coprocessStatus[mainMenu] := active;
0064 coprocessStatus[process1] := inactive;
0065 coprocessStatus[process2] := inactive;
0066 nextCoproces := mainMenu;
0067 repeat
0068 begin
0069 case nextCoproces of
0070 mainMenu: if (coprocessStatus[mainMenu] = active) then
0071 MainMenuProcess;
0072 process1: if (coprocessStatus[process1] = active) then
0073 ProcessOne;
0074 process2: if (coprocessStatus[process2] = active) then
0075 ProcessTwo;
0076 end; { case }
0077 if (nextCoproces = process2) then
0078 nextCoproces := mainMenu
0079 else
0080 nextCoproces := succ(nextCoproces);
0081 end { repeat }
0082 until stopProcessing;
0083 end. { MULTIPROCESS }
```

Multiprocessing Program Skeleton (2 of 2)

```

0001 Program MODEM (input,output);
0002
0003 Const
0004 EOFchar = 26; EOTchar = 4;
0005 Type
0006 coprocesses = (mainMenu,modem1,modem2);
0007 coprocessStatuses = (inactive,active,stopped,complete);
0008 MainMenuSteps = (MMdraw,MMprompt,MMgetChar,MMcheckCmd,MMuserError,
0009 MMrestart,MMdumbTerminal,MMprocessCmd);
0010 dumbTerminalSteps = (DTinit,DTgetUserChar,DTgetModemChar);
0011 processCmdSteps = (PCchannelPrompt,PCgetUserChar);
0012 modemSteps = (MDreceive,MDtransmit);
0013 menuOptions = (connectORdisconnect,startORstopReceive,
0014 startORstopTransmit,viewProcess,dumbTerminalMode);
0015 Var
0016 stopProcessing: boolean;
0017 coprocessStatus: array [mainMenu..modem2] of coprocessStatuses;
0018 nextCoproces: coprocesses;
0019 processStep: record
0020 MM: mainMenuSteps;
0021 DT: dumbTerminalSteps;
0022 MD: array [modem1..modem2] of modemSteps;
0023 PC: processCmdSteps;
0024 end;
0025 view: array [modem1..modem2] of boolean;
0026 connected: array [modem1..modem2] of boolean;
0027 modemFile: array [modem1..modem2] of text;
0028 menuOption: menuOptions;
0029 DTmodemNumber: modem1..modem2;
0030
0031 procedure AttachModem(
0032 modemNumber: coprocesses); EXTERNAL;
0033
0034 procedure DetachModem(
0035 modemNumber: coprocesses); EXTERNAL;
0036
0037 function GetModemChar(
0038 modemNumber: coprocesses;
0039 var MDchar: char): boolean; EXTERNAL;
0040
0041 procedure PutModemChar(
0042 modemNumber: coprocesses;
0043 MDchar: char); EXTERNAL;
0044
0045 function GetUserChar(
0046 var userChar: char): boolean; EXTERNAL;
0047
0048 procedure ModemProcess(
0049 processNumber: coprocesses);
0050 Var
0051 MDchar: char;
0052 begin
0053 case processStep.MD[processNumber] of
0054 MDreceive: if (GetModemChar(processNumber,MDchar)) then
0055 begin
0056 if (view[processNumber]) then
0057 write(MDchar);

```

MODEM  
01

Source Listing

15-May-1985 13:38:45  
15-May-1985 13:38:31

```
0058 write(modemFile[processNumber],MDchar);
0059 if (MDchar = chr(EQFchar)) then
0060 coprocessStatus[processNumber] := complete;
0061 end;
0062 MDtransmit: if (eof(modemFile[processNumber])) then
0063 coprocessStatus[processNumber] := complete
0064 else
0065 begin
0066 read(modemFile[processNumber],MDchar);
0067 if (view[processNumber]) then
0068 write(MDchar);
0069 PutModemChar(processNumber,MDchar);
0070 end;
0071 end; { case }
0072 end; { ModemProcess }
0073
0074 procedure ReinitMM;
0075 begin
0076 with processStep do
0077 begin
0078 MM := MMdraw;
0079 DT := DTinit;
0080 end;
0081 end; { ReinitMM }
0082
0083 procedure DumbTerminal;
0084 Var
0085 DTchar: char;
0086 begin
0087 case processStep.DT of
0088 DTinit: begin
0089 writeln;
0090 processStep.DT := DTgetUserChar;
0091 end;
0092 DTgetUserChar:
0093 begin
0094 if (GetUserChar(DTchar)) then
0095 if (DTchar = chr(EDTchar)) then
0096 processStep.MM := MMrestart
0097 else
0098 begin
0099 write(DTchar);
0100 PutModemChar(DTmodemNumber,DTchar);
0101 end;
0102 processStep.DT := DTgetModemChar;
0103 end;
0104 DTgetModemChar:
0105 begin
0106 if (GetModemChar(DTmodemNumber,DTchar)) then
0107 write(DTchar);
0108 processStep.DT := DTgetUserChar;
0109 end;
0110 end; { case }
0111 end; { DumbTerminal }
0112
0113 procedure ProcessCommand;
0114 Var
```

modem Program Example (2 of 5)

MODEM  
01

Source Listing

15-May-1985 13:38:45  
15-May-1985 13:38:31

```
0115 modemNumber: coprocesses;
0116 PCchar: char;
0117 begin
0118 case processStep.PC of
0119 PCchannelPrompt:
0120 begin
0121 write('Enter modem channel (1 or 2): ');
0122 processStep.PC := PCgetUserChar;
0123 end;
0124 PCgetUserChar:
0125 if (GetUserChar(PCchar)) then
0126 begin
0127 processStep.PC := PCchannelPrompt;
0128 if (PCchar = '1') or (PCchar = '2') then
0129 begin
0130 processStep.MM := MMprompt;
0131 case ord(PCchar)-ord('0') of
0132 1: modemNumber := modem1;
0133 2: modemNumber := modem2;
0134 end;
0135 if (menuOption = connectORdisconnect) then
0136 if (connected[modemNumber]) then
0137 begin
0138 connected[modemNumber] := false;
0139 DetachModem(modemNumber);
0140 end
0141 else
0142 begin
0143 connected[modemNumber] := true;
0144 AttachModem(modemNumber);
0145 end;
0146 if (menuOption = startORstopReceive) then
0147 if (coprocessStatus[modemNumber] = active) then
0148 coprocessStatus[modemNumber] := stopped
0149 else
0150 begin
0151 if (connected[modemNumber]) then
0152 begin
0153 coprocessStatus[modemNumber] := active;
0154 rewrite(modemFile[modemNumber]);
0155 end
0156 else
0157 writeln('* * * Modem channel not connected');
0158 end;
0159 if (menuOption = startORstopTransmit) then
0160 if (coprocessStatus[modemNumber] = active) then
0161 coprocessStatus[modemNumber] := stopped
0162 else
0163 begin
0164 if (connected[modemNumber]) then
0165 begin
0166 coprocessStatus[modemNumber] := active;
0167 reset(modemFile[modemNumber]);
0168 end
0169 else
0170 writeln('* * * Modem channel not connected');
0171 end;
```

Modem Program Example (3 of 5)

MODEM  
01

Source Listing

15-May-1985 13:38:45  
15-May-1985 13:38:31

```
0172 if (menuOption = viewProcess) then
0173 if (coprocessStatus[modemNumber] = active) then
0174 begin
0175 view[modemNumber] := true;
0176 processStep.MM := MMgetChar;
0177 end
0178 else
0179 writeln('* * * Process not active');
0180 if (menuOption = dumbTerminalMode) then
0181 if (coprocessStatus[modemNumber] = active) then
0182 writeln('* * * Modem channel active')
0183 else
0184 begin
0185 if (connected[modemNumber]) then
0186 begin
0187 DTmodemNumber := modemNumber;
0188 processStep.MM := MMdumbTerminal;
0189 end
0190 else
0191 writeln('* * * Modem channel not connected');
0192 end;
0193 end
0194 else
0195 writeln('* * * Modem channel out of range (1-2)');
0196 end;
0197 end; { case }
0198 end; { ProcessCommand }
0199
0200 procedure MainMenuProcess;
0201 Var
0202 MMchar: char;
0203 begin
0204 case processStep.MM of
0205 MMdraw: begin
0206 writeln;
0207 writeln(' Main Menu');
0208 writeln('1 Connect/disconnect modem channel');
0209 writeln('2 Start/stop file receive');
0210 writeln('3 Start/stop file transmit');
0211 writeln('4 View a receive/transmit process');
0212 writeln('5 Dumb terminal mode');
0213 writeln('6 End');
0214 processStep.MM := MMprompt;
0215 end;
0216 MMprompt: begin
0217 writeln;
0218 if (coprocessStatus[modem1] = active) then
0219 writeln('----- Channel 1 active -----');
0220 if (coprocessStatus[modem2] = active) then
0221 writeln('----- Channel 2 active -----');
0222 if (coprocessStatus[modem1] <> active) and
0223 (coprocessStatus[modem2] <> active) then
0224 writeln('----- No active processes -----');
0225 write('Enter option number: ');
0226 processStep.MM := MMgetChar;
0227 end;
0228 MMgetChar: if (GetUserChar(MMchar)) then
```

Modem Program Example (4 of 5)

MODEM  
01

Source Listing

15-May-1985 13:38:45  
15-May-1985 13:38:31

```
0229 if (MMchar < '1') or (MMchar > '6') then
0230 begin
0231 writeln('* * * Option number out of range (1-6)');
0232 processStep.MM := MMprompt;
0233 end
0234 else
0235 begin
0236 case ord(MMchar) - ord('0') - 1 of
0237 0: menuOption := connectORdisconnect;
0238 1: menuOption := startORstopReceive;
0239 2: menuOption := startORstopTransmit;
0240 3: menuOption := viewProcess;
0241 4: menuOption := dumbTerminalMode;
0242 5: stopProcessing := true;
0243 end;
0244 processStep.MM := MMprocessCmd;
0245 end;
0246 MMprocessCmd: ProcessCommand;
0247 MMdumbTerminal:
0248 DumbTerminal;
0249 MMrestart: ReinitMM;
0250 end; { case }
0251 end; { MainMenuProcess }
0252 {
0253 + + + COPROCESSOR + + +
0254 }
0255 begin
0256 writeln('MODEM COMMUNICATION PROCESSOR');
0257 stopProcessing := false;
0258 coprocessStatus[mainMenu] := active;
0259 coprocessStatus[modem1] := inactive;
0260 coprocessStatus[modem2] := inactive;
0261 nextCoproprocess := mainMenu;
0262 processStep.MM := MMrestart;
0263 view[modem1] := false;
0264 view[modem2] := false;
0265 connected[modem1] := false;
0266 connected[modem2] := false;
0267 repeat
0268 begin
0269 case nextCoproprocess of
0270 mainMenu: if (coprocessStatus[mainMenu] = active) then
0271 MainMenuProcess;
0272 modem1: if (coprocessStatus[modem1] = active) then
0273 ModemProcess(modem1);
0274 modem2: if (coprocessStatus[modem2] = active) then
0275 ModemProcess(modem2);
0276 end;
0277 if (nextCoproprocess = modem2) then
0278 nextCoproprocess := mainMenu
0279 else
0280 nextCoproprocess := succ(nextCoproprocess);
0281 end
0282 until stopProcessing;
0283 end.
```

Modem Program Example (5 of 5)

M. L. Naegele and H. J. Gould, PhD  
Center for Craniofacial Anomalies  
University of Illinois Medical Center  
Chicago, Illinois

#### ABSTRACT

This paper presents a three step system designed to input, measure and analyse cephalograms. The inputs to the system are xrays digitized with a variable number of points and instructions to perform measurements. The output of the system provides hard copy plots representing the digitized points, constructed points from existing points such as the intersection of two lines, magnification corrections between inconsistent xrays, and measured variables in a format ready for statistical analysis.

#### HISTORY

The Center for Craniofacial Anomalies, University of Illinois, is a unique facility which started in 1949. The Center has grown into one of the largest multidisciplinary centers in the world for treatment, research, and education in the field of congenital and acquired deformities of the head and neck. A multidisciplinary staff of more than 50 members come to the Center from the six Chicago metropolitan area hospitals in addition to the University of Illinois. The Center's faculty is a group of professionals whose training and expertise cover all areas of dentistry and medicine which find significance in craniofacial biology. Meticulous records which document findings and service from the multiple specialists have been maintained in a longitudinal manner. Most patients are followed for a period of treatment and active observation which encompasses at least 10 to 20 years. The Center is automating procedures using a VAX 11/750 computer. Several studies have been performed using the system described in this paper for cephalometric analysis.

take approximately 1.5 hours per cephalogram, have included up to 75 unique variables and magnification corrections. The measured variables are defined distances, angles, or ratios of distances. Magnification corrections are required whenever a cephalogram's magnification varies from the reference magnification. Statistical analysis of these results is laboriously performed by using calculators. The above lengthy and tedious process is a perfect candidate for a computer solution.

#### SOFTWARE/HARDWARE SOLUTION

A three step system was designed to improve accuracy in the measurement and analysis of the cephalograms with the added benefit of reducing analysis time. The system shown in figure 1a demonstrates the three independent areas of the program, which input data, create and perform measurement instructions, and analyse results. This paper will discuss these three areas independently. Figure 1b depicts the hardware used for the analysis.

#### PROBLEM

The staff at the center analyse and measure cephalograms (head xrays) of craniofacial patients in order to aid in their diagnosis and research. The measurement of cephalograms is done by defining points, constructing points on the intersection of lines, measuring distances and angles on each cephalogram with rulers and protractors. These measurements, which can



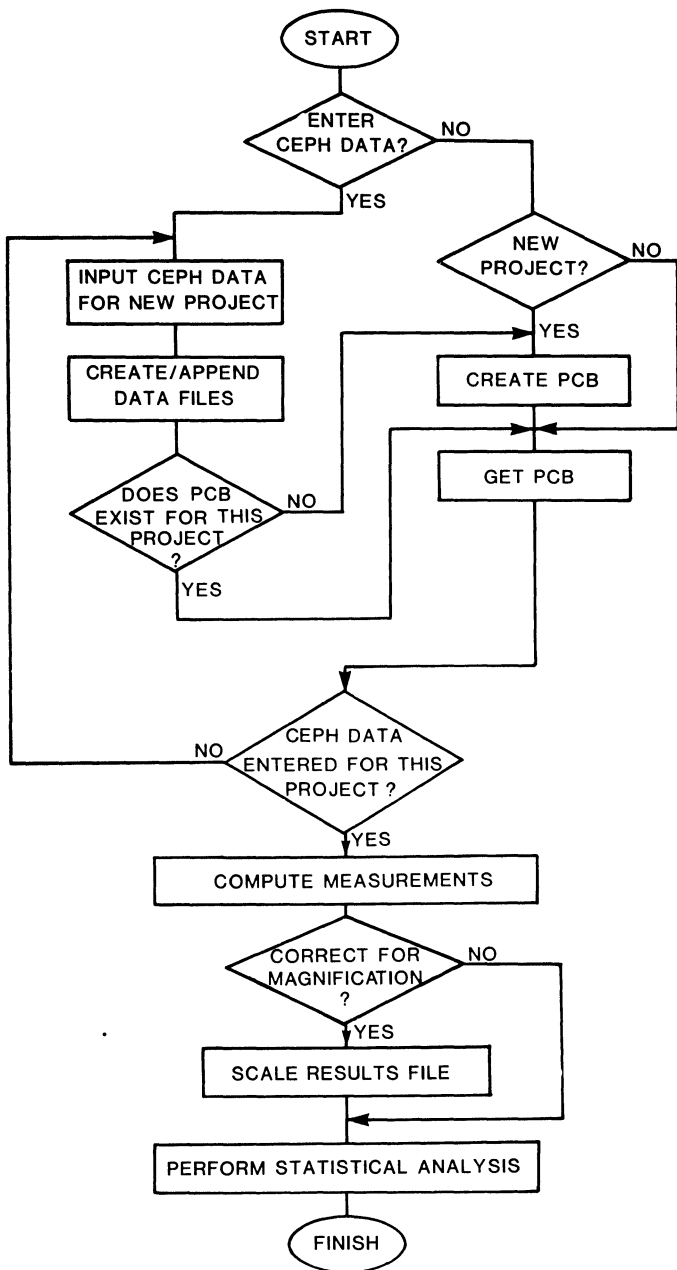


Fig. 1a. Flowchart of cephalometric analysis.

Step 1. Cephalogram traces (see Figure 2) are digitized using a Houston Instruments digitizing pad on a PDP 11/23 MINC system under RT-11. A variable number of points can be digitized and stored for each project. The software protects against file overwrites by checking the requested filename against the current directory under RT-11.

Once the filenames and the number of points to be digitized have been established

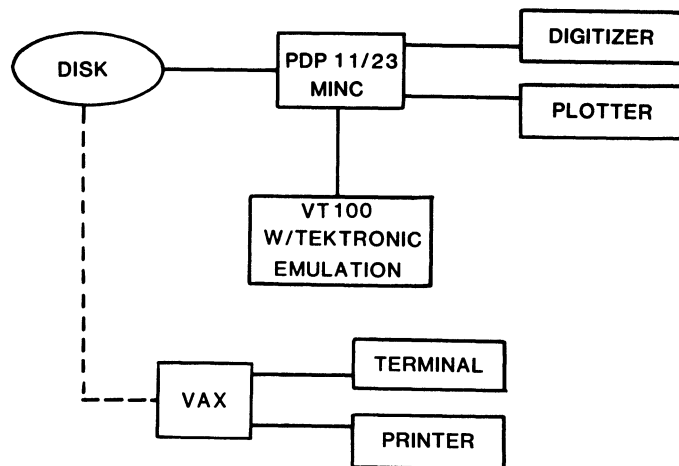


Fig. 1b. Hardware used for analysis.

digitization begins. The digitization routine displays all digitized points on a VT100 with 4010 emulation for immediate point verification. If a digitizing error is detected then one or all point(s) can be erased. Missing points on the cephalogram for whatever reason (hidden, unidentifiable) are marked missing by inserting a negative coordinate value. A safety against the wrong number of points digitized is built in by not permitting storage of data to disk if the point count is not the same as the one defined earlier by the user.

The points shown in figure 2 were digitized on each cephalogram. These points are stored in a sequential file. The file contains a patient identification header, xray magnification and in the case of the lateral cephalogram, 29 X and Y coordinates are appended. Note the two additional points added to the 27 lateral data points are used as registration points. This raw data file has a .LAT file extension. Figure 3 depicts patient 1044's frontal digitized points. Additional cephalograms are appended to the end of this file.

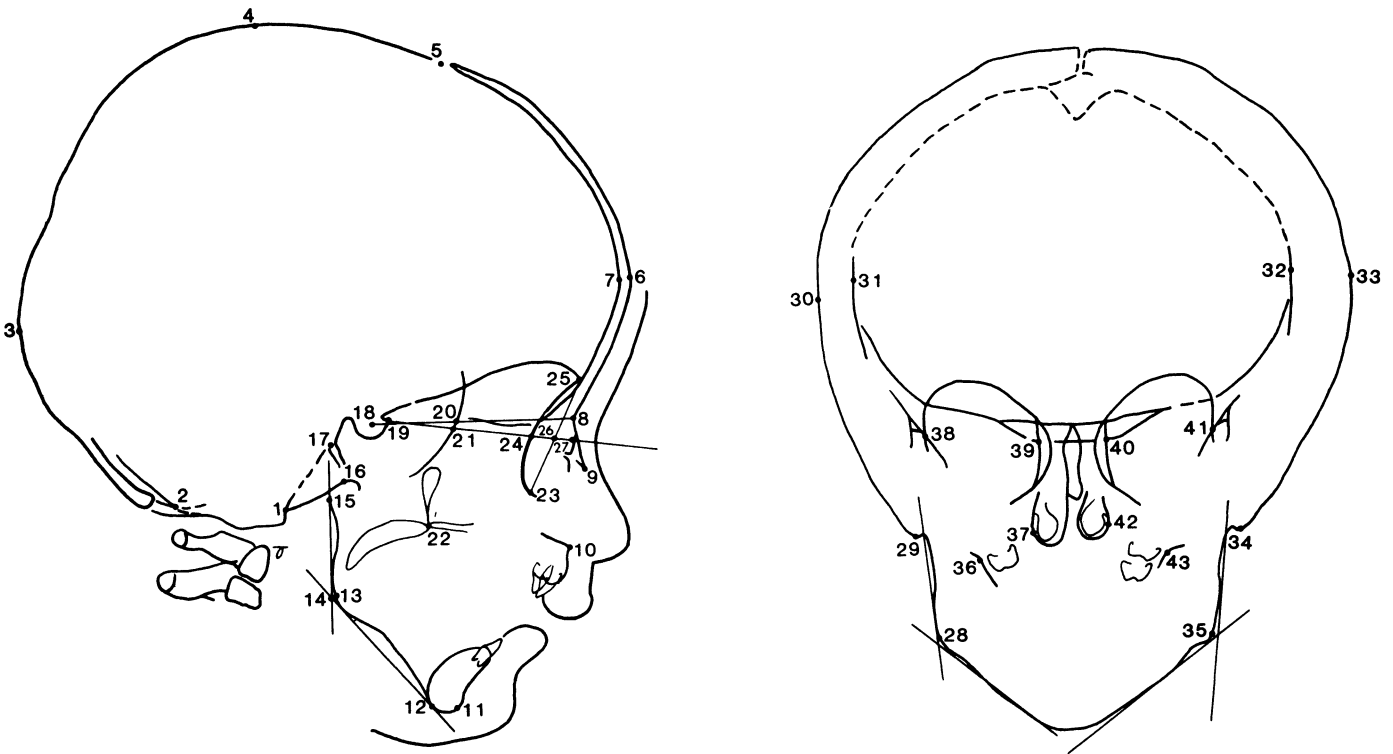


Fig. 2. Lateral and frontal cephalogram traces with digitized point markings.

|           |   |       |             |             |
|-----------|---|-------|-------------|-------------|
| header    | ➔ | 1044  | 8CM6 10.000 |             |
|           |   | 10.56 | 20.57       | ← digitized |
|           |   | 18.12 | 20.19       | ← reference |
|           |   | 6.60  | 2.76        | points      |
|           | ➔ | 5.66  | 4.86        |             |
|           |   | 3.17  | 13.05       |             |
|           |   | 4.47  | 11.64       |             |
|           |   | 14.97 | 11.95       |             |
|           |   | 15.91 | 13.71       |             |
| frontal   |   | 13.83 | 5.02        |             |
| digitized |   | 12.72 | 2.66        |             |
| point     |   | 7.39  | 5.10        |             |
| markings  |   | 8.87  | 6.15        |             |
| 28-43     |   | 6.07  | 8.22        |             |
|           |   | 8.99  | 8.11        |             |
|           |   | 10.79 | 8.17        |             |
|           |   | 13.62 | 8.49        |             |
|           |   | 10.87 | 5.91        |             |
|           | ➔ | 12.10 | 5.23        |             |

Fig. 3 Raw data file containing header and digitized points.

Hard copy plots showing all digitized points (see Figure 4) are made by accessing the raw data files. The plots are overlaid on the original cephalogram trace to insure proper digitization.

Step 2. Identify and compute desired variables to be measured using VAX.

In order to analyse the input cephalometric data we create what we term a process control block (PCB). The PCB defines the measurements wanted and gives instructions for constructing data points. Creation of a PCB can be performed in an asynchronous mode with data input (see Figure 1a). Variables

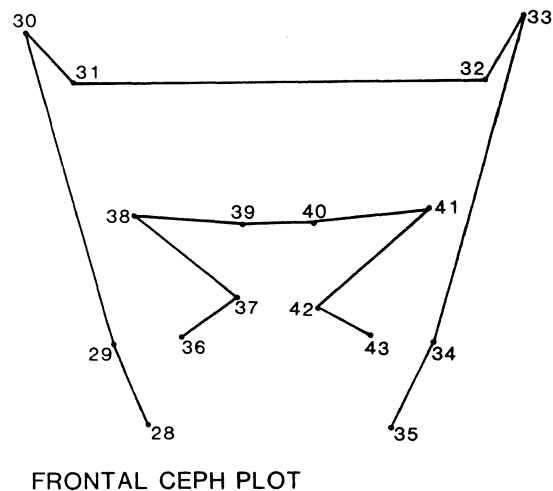
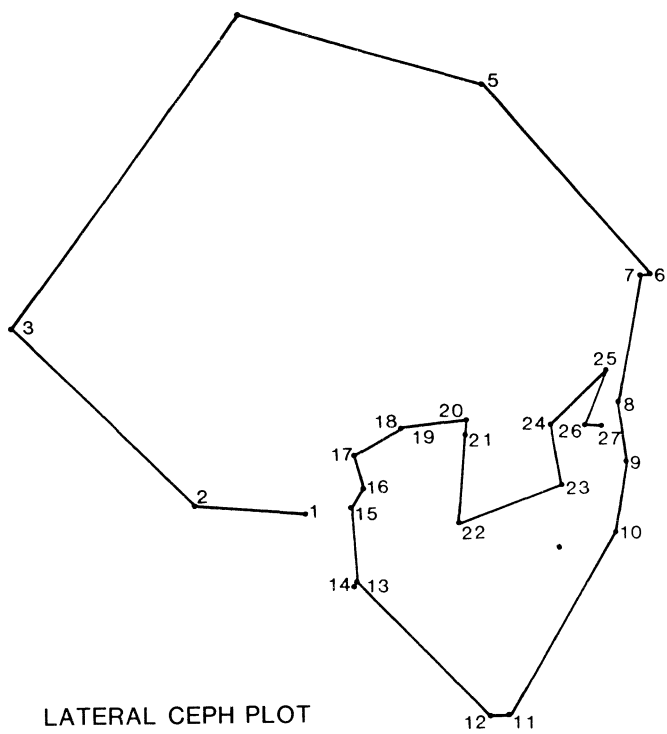


Fig. 4. Lateral and frontal hard copy plots of digitized cephalograms.

to be measured, constructed points and identifying labels are defined through a menu driven routine that provides the following choices:

1. Undefined.
2. Point to point distance.
3. Point to line distance.
4. Angle between 3 points.
5. Constructed point of intersection between two lines.
6. Constructed point perpendicular to a line.
7. Add two variables.
8. Subtract two variables.
9. Multiply two variables.
10. Divide two variables.
11. End process control.

The PCB is created by a menu driven routine. Measurement instructions are created thru the above routine which prompts for the type of measurement to be made, the location in the X and Y data structure for constructed points or the MEASURED data structure (Each data structure is a one dimensional array.), the points to be operated on and a label to identify the constructed point or variable

created. To illustrate its function assume we want to measure an angle, function 4 above is selected which prompts for the 3 digitized points and a label to identify the angle. There will be a separate PCB created for lateral and frontal points.

To measure the distance between point 30 and point 33 on the frontal cephalogram, the menu driven routine would prompt as follows:

|              |         |                      |
|--------------|---------|----------------------|
|              |         | Comments:            |
| function ==> | 2       | ; distance function  |
| variable ==> | 100     | ; measured variable  |
|              |         | ; stored at          |
|              |         | ; location 100       |
|              |         | ; in MEASURED array  |
| point ==>    | 30      | ; digitized point 30 |
| point ==>    | 33      | ; digitized point 33 |
| label ==>    | Ecl-Ecl | ; ectocranium right- |
|              |         | ; ectocranium left   |

To measure the perpendicular distance between point 12 and the line connecting points 18 and 8 on the lateral cephalogram,

```
function ==> 3 ; distance to a
 ; line function
variable ==> 101 ; variable stored at
 ; location 101
 ; in MEASURED array
point ==> 12 ; digitized point 12
endpoint1==> 18 ; digitized point 18
endpoint2==> 8 ; digitized point 8
label ==> Me-NSPL ; menton to
 ; sella-nasion plane
```

To measure the angle between points 18,8,9 where point 8 is the vertex on the lateral cephalogram,

```
function ==> 4 ; angle function
variable ==> 102 ; variable stored at
 ; location 102
 ; in MEASURED array
point ==> 18 ; digitized point 18
vertex ==> 8 ; digitized point 8
point ==> 9 ; digitized point 9
label ==> S-N-Na ; sella-nasion-nasale
```

To construct a point of intersection between two lines where the first line is from points 8 to 12 and the second line from points 10 to 22,

```
function ==> 5 ; intersection
function
variable ==> 45 ; point stored at
 ; X,Y location 45
endpoint 1 of line 1 ==> 8
endpoint 2 of line 1 ==> 12
endpoint 1 of line 2 ==> 10
endpoint 2 of line 2 ==> 22
label ==> intersection 8-12/10-22
```

To construct a point perpendicular to a line, point 12 perpendicular to line 8-18,

```
function ==> 6 ; point construction
 ; function
variable ==> 46 ; point stored at
 ; X,Y location 46
point ==> 12 ; digitized point 12
endpoint1==> 8 ; digitized point 8
endpoint2==> 18 ; digitized point 18
label ==> point 12 on a line
 ; perpendicular to line 8-18
```

To operate on the MEASURED data structure from the above menu we can add, subtract, multiply and/or divide any two stored measured variables. To find the ratio between the two measured distance variables 100 and 101,

```
function ==> 10 ; divide two
 ; variables function
variable ==> 103 ; ratio stored at
 ; location 103
 ; in measured array
numerator ==> 100 ; will divide the
 ; contents of
denominator ==> 101 ; variable 100 by
 ; the contents
 ; of variable 101.
label ==> ratio Ecl-Ecl/Me-NSPL
```

This menu driven routine creates a process control block (see Figure 5) that is stored in a file with a .PCB extension. This control block stores the information specifying each function selected and its arguments. The process control blocks for the examples given above is shown in figure 5.

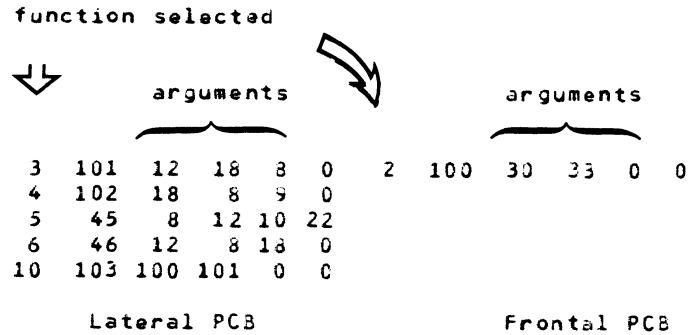


Fig. 5 Process control block (PCB) for examples of variable computation.

This block can be as large as necessary. Constructed points (functions 4 and 5) will fill in after the existing digitized points in the X and Y arrays.

After all variables to be measured have been defined in the process control block the computation of measurements begins. These variables must be measured for N cephalograms. The .PCB file is accessed and the desired operation is performed on the appropriate raw data file (digitized points and constructed points) to create a results file with extension .RES. When a computation is performed on a missing value the variable is marked invalid to aid in future analysis. The results file contains N cephalograms with M measured variables (N records with M\*7 bytes per record).

Cephalograms have different magnification factors associated with them causing inconsistencies. For example, when trying to observe growth on a patient the magnification may vary from one cephalogram to another. Measured variables are initially computed for each cephalogram without any corrections for magnification. However, correction must be made before any measured variables can be properly analysed. Magnification correction is made by scaling to a desired magnification, which is usually 0%. This scaling is done by operating on the results file. All measured variables can be scaled for magnification except for angles.

In summary the following files are created:

Step 1

```
project.FRD ;frontal digitized points
project.LAT ;lateral digitized points
```

## Step 2

```
projectf.PCB ;process control
 block (frontal)
projectl.PCB ;process control
 block (lateral)
projectf.RES ;measured results (frontal)
projectl.RES ;measured results (lateral)
projectf.SCA ;scaled results (frontal)
projectl.SCA ;scaled results (lateral)
```

## Step 3. Perform analysis using VAX.

The scaled results file containing N cephalograms with X measured variables may be accessed by DATATRIEVE or SAS for easy accurate statistical analysis. After scaling the magnifications for frontal and lateral cephalograms, the records can be crossed on patient identification number to perform any ratios containing both frontal and lateral points. Since each patient has frontal and lateral variables associated with him, we use datatrieve to cross files on patient identification number which gives access to all frontal and lateral variables for that patient. Ratios between frontal and lateral variables can then be found for various groups of patients on several keys such as sex, age, race, diagnosis, etc. Means, standard deviations, confidence intervals, ranges, T-tests, and/or non-parametric statistics can be performed on the resulting measured variables and ratios of measured variables with DATATRIEVE or SAS.

## CONCLUSION

The calculation of variables was written in fortran. Some of the larger studies required up to an hour of CPU time to perform all measurements. This time of course is directly proportional to the number of patients being analyzed and the number of variables to be measured for each patient. DATATRIEVE was used to performed the statistical portion and was slow due to creating collections and crossing collections. Most jobs were run overnight to avoid slowing down the system. This cephalometric system has saved many users hundreds of hours of time while improving all studies performed with it.

J. R. Birkelund, J. A. Abate, T. S. Lund  
Kodak Research Laboratories  
Eastman Kodak Company  
Rochester, New York 14650

#### ABSTRACT

Comparison of interrupt response times for PDP11/44 and PDP11/23+ machines, using the FORTH programming language, is presented. The interrupt response of the machines with FORTH implemented as a stand-alone operating system is compared with an implementation of FORTH running under the RSX11M operating system. The comparisons have been made on systems operating CAMAC parallel interface buses, both as single bus controllers (IEEE 583) and as parallel highways (IEEE 596), which require a branch driver interface. The measurements show that time overhead in response to interrupts is reduced by a factor of about 2 when stand-alone FORTH is used compared to FORTH implementations under the RSX operating system.

#### 1. INTRODUCTION

Many factors are involved in the selection of an operating system and programming language for real-time computer applications. Among the most important of these factors are the response of the chosen system to interrupts and the ease of programming external hardware functions, especially interrupts. The FORTH language has been developed with real-time applications in mind and offers several advantages to the programmer when access to hardware external to the computer is called for. FORTH may be implemented as a task running under some operating system, such as RSX, or it may be implemented as a stand-alone system. The operation of FORTH under a standard operating system has attractive features, since it permits both the ease of access to hardware provided by FORTH and access to the powerful multipurpose functions provided to a user by other operating systems. This paper considers the advantages and disadvantages of these two types of implementation from the viewpoint of interrupt response.

This paper addresses the issue of the interrupt response time for several operating system and hardware configurations that use the FORTH language, in an attempt to quantify the advantages or disadvantages here from the use of FORTH. The work described applies to a particular application in which the external hardware conforms to the CAMAC interface standard, and the data described apply to that system. However, the performance measurements should give some guidance to users implementing real-time systems with different configurations.

Section 2 describes salient features of the FORTH language for real-time operations. Section 3 describes some features of the CAMAC interfaces. Sections 4 and 5 describe the hardware and software configurations used for the testing. Section 5 describes the test methods used to evaluate the interrupt response of the systems described in Section 3. Section 6 gives the results, and Section 7 gives the conclusions.

#### 2. THE FORTH LANGUAGE

The FORTH language has been described elsewhere (1, 2). FORTH is a threaded interpretive language that may be run in a computer as a stand-alone operating system or as a task running under some standard operating systems such as RSX-11 or RT-11. The operation of FORTH is in some respects similar to the more familiar BASIC language, in that the programmer may define new operations, called words, which become immediately available for execution, through a resident compiler, without the necessity of a separate compile and link step required by noninterpretive languages. The FORTH system maintains a dictionary of defined words, and new words can be constructed out of combinations of already defined words. The FORTH dictionary contains pointers to the code and necessary parameters required to execute the functions required by the word definition. By looking through the dictionary for the code to execute as each word is encountered, the machine performs the functions required of it. Words exist that allow new definitions to be compiled into the dictionary as required. In addition, FORTH allows the possibility of inline code, called code words, constructed by a FORTH assembler, which, once entered, are executed in specified order as a set of machine instructions, without the necessity to look through the dictionary pointers. Use of these code definitions is equivalent to code built by a conventional assembler, which provides the fastest possible execution, and such code definitions are used in the FORTH interrupt service routines tested for this paper.

The operation of FORTH is controlled by a section of FORTH code comprising an inner interpreter and outer text interpreter. The text interpreter parses the control input stream, which may come either from the terminal or from other storage such as a disk. The text interpreter places input numbers on a first-in, last-out parameter stack and looks through the dictionary for the definition of words found in the input stream. If the words are found, the code defined for them is executed

by using the parameters on the stack. A new dictionary-entry-defining word exists, usually ':', which causes the text interpreter to pass control to the inner interpreter if it is encountered in the input stream. When the inner interpreter has control, it compiles into the dictionary new code defined by the words and parameters in the input stream. When the inner interpreter encounters the compiler exit word, usually ';', control is returned to the text interpreter, and the new dictionary entry is complete. Note that the input stream is not retained, and interpretation of the input stream is done by execution in the order specified by the input stream, of the code pointed to by the dictionary entries. Thus, the only parsing step required to build executable code occurs when code is compiled into the dictionary, and thereafter execution of the code is very fast.

In its stand-alone form, FORTH allows the user access to all addresses accessible to the computer, including the I/O page in PDP11 systems and the memory management registers. With the FORTH equivalent of MOV instructions, the user may write directly in these registers to control external devices through the I/O page or rearrange the memory mapping. No protections, checks, or controls are provided for any of these addresses, and the programmer has complete control of them. This is both a blessing and a curse, but it is the source of the special utility of FORTH in operations involving external hardware. In addition, for interrupt programming, the programmer may define a FORTH code word or a high-level FORTH word to serve as the interrupt service routine and place the entry address of this ISR directly into the vector address of the external device that will produce the interrupt requiring service. When an interrupt occurs, since FORTH does not make use of the various modes available in a PDP11 and remains always in kernel mode, the system is simply interrupted and jumps directly to the address given in the interrupt vector, returning by execution of the machine instruction RTI.

When operated as a task under an operating system, code to handle external devices with FORTH must conform to the rules laid down by the operating system for I/O page and interrupt access. With RSX-11, the FORTH task must be privileged, must be mapped to the I/O page through a previously built operating system device common with the use of active page register 7, and must have pr:0 set at task build time to allow the use of the CINT\$ connect to interrupt system service. When an interrupt occurs, the action of the machine is somewhat more complicated than in the stand-alone case, owing to the necessity for RSX to take care of the multiple tasks and the memory mapping. The interrupt puts the machine into kernel mode, at processor priority 7, maps a designated region of the user's task through PAR 5, and jumps to the address specified in the CINT\$ directive. The interrupt service routine runs in kernel mode with PAR 5 mapping of its logical address space and exits with the FORTH assembler equivalent of the RTS PC instruction. This causes some time overhead, which is discussed further below.

This interpretive scheme is useful in interaction with hardware, since it allows new control functions to be easily defined and tested and is thus attractive for development of hardware control systems.

The advantages claimed by proponents for the FORTH language are that code in FORTH is fast in execution, quickly developed and tested incrementally, and compact and convenient when constant development is required. The major disadvantages of FORTH as a stand-alone system are its inability to run code built in other languages, its lack of a convenient file structure on external storage devices, and its too simple multitasking and multiuser facilities. In addition in the particular application of interest to us, a stand-alone FORTH machine could not easily be incorporated into DECNET. The use of FORTH under a standard operating system is an attempt to gain the best of both worlds.

### 3. THE CAMAC INTERFACE

CAMAC is an acronym for Computer Automated Measurement and Control and is defined by IEEE standards 583, 595, 596, 675, and 683. The system consists of a standard 86-line bus, housed in a mechanical assembly called a crate, which usually contains 25 slots for the functional modules. The CAMAC bus, referred to as the dataway, and the functional modules are interfaced to the control computer via a crate controller module inserted into the crate, usually at station number 25. The CAMAC crate has slots into which modular CAMAC units may be inserted, for various control and measurement functions to be performed in the external world. The mechanical assembly is such that CAMAC modules may easily be removed and inserted in the crate, and the functions of the bus lines are so standardized that control of the modules from the computer may be obtained by software alone. Thus the system provides a potentially flexible control and measurement interface between the computer and the outside world. The arrangement is shown schematically in Figure 1. The arrangement of the dataway is such that the controller station has two exclusive connections to each slot, via lines known as N, the slot address, and L, the attention line used by the modules in each slot to get the attention of the controller. If the L line is asserted by a module, then the system may be configured to interrupt the control computer. These CAMAC module requests are called Look-At-Me or LAM requests.

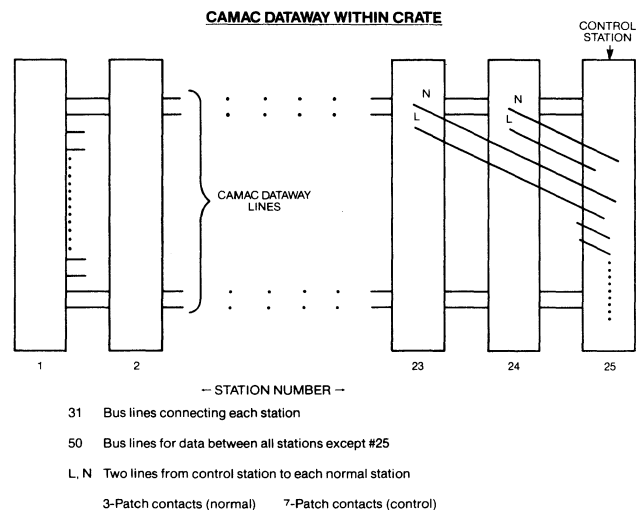


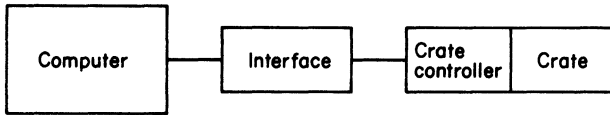
Figure 1.

Schematic of the CAMAC crate dataway lines.

There are two major configurations for CAMAC crate connections to a computer: the dedicated-interface configuration and the branch highway, shown schematically in Figures 2 and 3. The dedicated-interface configuration allows a computer to communicate with one crate controller, for control of the modules in a single crate, whereas the branch-highway configuration provides an interface between the computer and a branch driver, which allows the computer to communicate with several crate controllers via a highway, which may be either a parallel or a serial bus link. For this paper only parallel highway systems will be considered.

### CAMAC CONFIGURATION

#### Dedicated interface



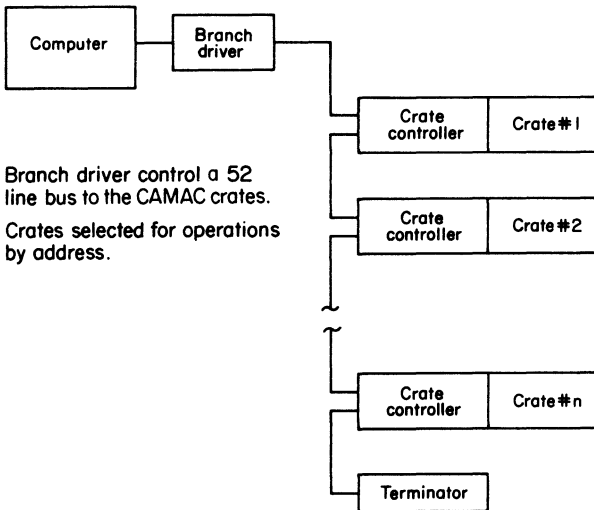
Controller - Interface connection via a 52 line connection.

Figure 2.

Schematic of the connections in a dedicated CAMAC computer interface connection.

### CAMAC CONFIGURATION

#### Branch highway



Branch driver control a 52 line bus to the CAMAC crates. Crates selected for operations by address.

Figure 3.

Schematic of the CAMAC crate-computer connections for a branch-highway connection. The connections between the branch driver interface and the crate form a parallel bus.

Each module in the crate is controlled by the computer according to a standardized protocol in which the control word is divided into three subsections: N for the slot address, A for the sub-address in the module, and F for the function to be

performed by the module. These NAF codes may be contained in a single word, thus making it possible to perform an action of a single module with a single computer instruction, moving the control word to the interface control register. On some systems a second instruction is also necessary to set a 'Go' bit in the interface CSR.

#### 4. HARDWARE CONFIGURATIONS

These are shown in Figures 4 and 5. The interrupt performance of two computers and two CAMAC interfaces has been tested. In addition, the performance of stand-alone FORTH is compared with FORTH implemented as a task under RSX-11-M. The hardware configurations are a PDP11/23 processor (KD11AA) connected to a dedicated CAMAC interface (Kinetic Systems 2920) on the Q-bus and a PDP11/44 with its UNIBUS connected via a Qniverter (Able Computer) to a Q-bus with the same dedicated interface and the 11/44 connected to a branch-highway interface (Jorway 411) directly on the UNIBUS.

#### 11/44 CONFIGURATIONS TESTED

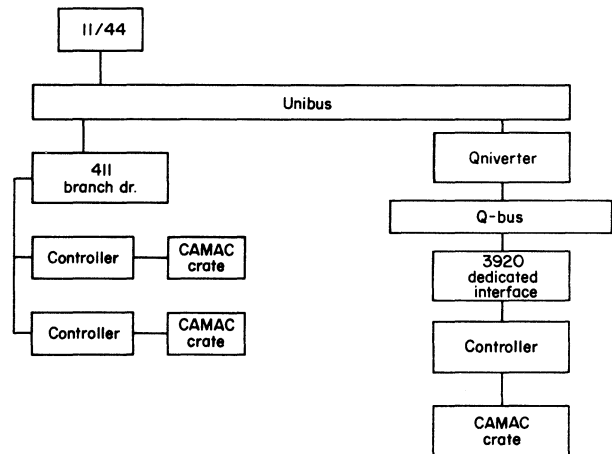


Figure 4.

Diagram of the PDP11/44 system tested for interrupt response. Both the dedicated and branch-driver interfaces were tested on this machine.

The configuration for testing the interrupt response is shown in Figure 6. The tests were performed with a triggerable CAMAC analog-to-digital converter (Standard Engineering 212), which produces a IAM request after a conversion initiated by the trigger. The triggers were produced by a TTL pulser, and these triggers were counted by a fast CAMAC scaler (Kinetic Systems 3640), which could be read from the computer by CAMAC instructions. In addition, the pulser trigger started the sweep on an oscilloscope and was displayed on the scope to provide a time reference. The action of the dataway could be monitored by a dataway display control (Kinetic System 3296), which gave a gate pulse when NAF codes selected by switches were detected on the



# 11/23 CONFIGURATION TESTED

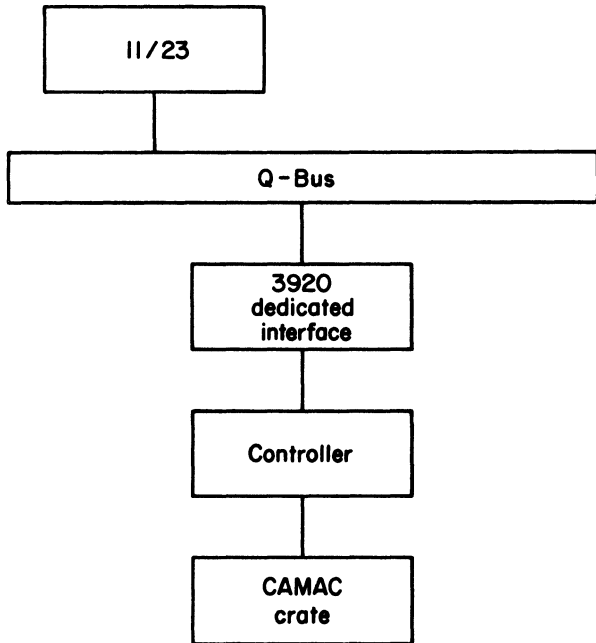


Figure 5.

Diagram of the PDP11/23 system tested with a dedicated CAMAC interface and a single crate.

dataway. The output of the dataway display control was counted in a second channel of the scaler and also displayed on the oscilloscope. Thus time measurements could be made on the oscilloscope, and the scaler values could be used to check that the number of triggers was the same as the number of dataway operations, to ensure that no interrupts were lost. A schematic view of the timing cycle is shown in Figure 7.

## TEST CIRCUIT

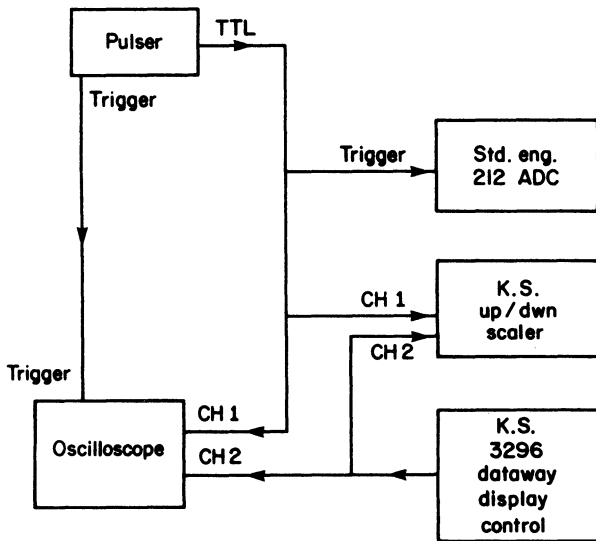
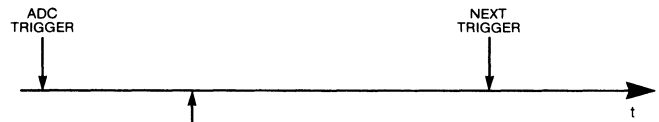


Figure 6.

Testing circuit for the interrupt-response-time measurements.

## TIMING DIAGRAM



1. Fast Loop Time:  
—Time between triggers when first dataway cycle pulse loses strict synchronism with trigger.
2. Fastest Time:  
—Time between triggers with fast loop enabled when lams begin to be lost.

Figure 7.

Schematic timing diagram for the display seen on the oscilloscope during testing.

## 5. INTERRUPT SERVICE ROUTINES

The functions of the interrupt service routines used in the tests are shown in Figure 8 for the dedicated interface and in Figure 9 for the branch highway. Some differences in the actual code were necessary between the stand-alone and RSX implementations, but these were minor and were mostly concerned with which PAR mapped the code when the ISR was running. The interrupt service routines were coded in FORTH assembler and are machine-language routines directly programmed. Thus, apart from inefficiencies that may be introduced by the programmer, they run as fast as the computer hardware will allow. We believe that these are coded as efficiently as possible, and therefore run-time differences in the various configurations tested represent overhead inherent in the different software implementations of the FORTH language.

## FUNCTIONS OF ISR [3920]

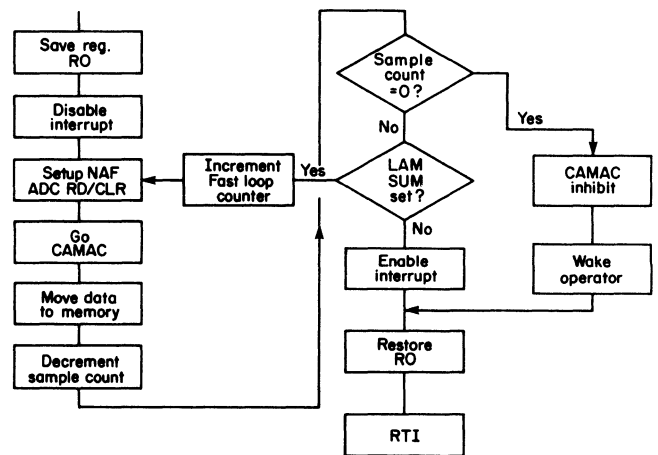


Figure 8.

Logical functions of the interrupt service routine tested for the dedicated CAMAC interface.

FUNCTIONS OF ISR

[411]

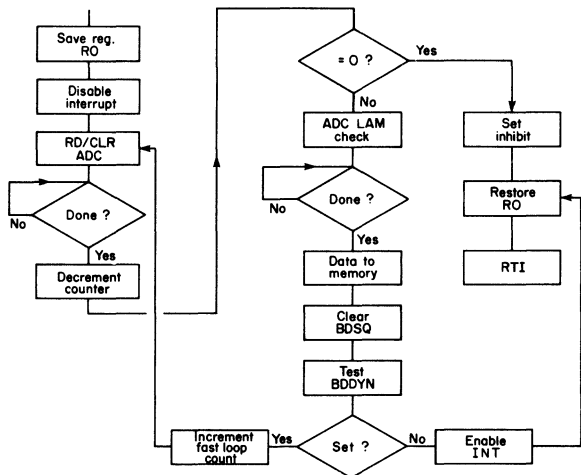


Figure 9.

Logical functions of the interrupt service routine tested for the branch-driver interface on the PDP11/44.

The interrupt service routines described in Figures 8 and 9 represent the minimum interrupt service routine possible with the CAMAC interfaces used here. Once the interrupt service routine is entered, the routine saves a register for use within the routine, disables interrupts from the CAMAC interface, reads and clears the ADC, transfers to memory the data sent to an interface register by the ADC, decrements the ISR cycle counter, and inhibits the CAMAC crate if this counter is zero. If the counter is not zero, the interfaces have a bit in their CSR that indicates if another LAM request was made during the run time of the ISR. This pending LAM will not interrupt the computer, since interrupts are disabled, but by checking the appropriate bit, the pending request can be serviced in a 'fast loop' without incurring the overhead required to get into and out of the ISR. The number of fast loops is counted by the ISR. Finally, the ISR restores the register and returns.

6. RESULTS OF THE MEASUREMENT

The timing measurement results are shown in Table 1. All times are measured with respect to the ADC trigger pulse. The time to the first dataway cycle in the CAMAC crate represents the sum of the ADC conversion time, the interrupt latency of the processor, the software overhead in transferring control to the user's interrupt service routine, and the run time of the instructions required to set up and read the contents of the ADC output register. Measurements show that the ADC conversion time is 10  $\mu$ s, and there is no appreciable delay in transferring a LAM seen by the crate controller to the computer bus interrupt lines.

The 'fast loop' times shown in Table 1 are the times between sequential ADC triggers when the 'fast loop' or overhead free interrupt service cycles begin. This can be seen on the oscilloscope trace shown schematically in Figure 7 by increasing the trigger frequency until the exact time synchronism is lost between the first dataway cycle and the trigger

PDPII INTERRUPT RESPONSE TIME (CAMAC)

TIMES IN  $\mu$ s AFTER ADC TRIGGER

DEDICATED INTERFACE (3920)

|            | 1st Dataway Cycle | Fast Loop | Fastest |
|------------|-------------------|-----------|---------|
| 11/23 S.A. | 50                | 90        | 70      |
| 11/23 RSX  | 100               | 170       | 150     |
| 11/44 S.A. | 35.2              | 48.2      | 39      |
| 11/44 RSX  | 63.6              | 102.0     | 102     |

BRANCH HIGHWAY (411)

|            | 1st Dataway Cycle | Fast Loop | Fastest |
|------------|-------------------|-----------|---------|
| 11/44 S.A. | 25.2              | 45.0      | 33      |
| 11/44 RSX  | 53.8              | 103.2     | 80      |

These times include 10  $\mu$ s for the ADC conversion time.

Table 1.

Interrupt response times for the various hardware and software configurations tested. Times are recorded in microseconds from the trigger of the ADC and measure the period to the first cycle seen on the CAMAC dataway, the shortest time between triggers before overhead-free interrupt service occurs, and the shortest period between triggers before interrupts are lost. Data are recorded for both 11/23 and 11/44 computers, for both dedicated and branch-highway CAMAC systems, and for stand-alone FORTH.

pulse. This fast-loop time represents the run time of the interrupt service process up to the test of the 'LAM sum' bit on the dedicated interface or the 'BDDYN' bit on the branch-highway interface. Except for the register restore, interrupt enable (BIS), and return instructions, that is the whole run time of the ISR.

The third column of Table 1 shows the fastest interrupt frequency that can be obtained from the system and is measured as the time between triggers of the ADC at which interrupts begin to be missed or not serviced at all. This represents a condition when all interrupts are serviced in the 'fast loop' mode.

Table 1 shows that there is a factor of almost 2 increase in speed upon going from the 11/23 to the 11/44 regardless of which operating system is used, and stand-alone FORTH gives an increase of a factor of about 2 in interrupt response speed over the RSX implementation on both the 11/23 and the 11/44. Since the interrupt service routines implemented for this test do few useful operations, they represent almost completely the overhead in processing interrupts from the CAMAC crates.

Table 2 gives times from the processor manual for the 11/23 for the instructions executed in the interrupt service routine for the dedicated controller up to the point of initiation of the first

dataway cycle. These times add up to 26  $\mu$ s. The manual indicates that the interrupt latency should be 9.75  $\mu$ s, and the ADC conversion time is 10  $\mu$ s. This gives a hardware theoretical time to the first dataway operation of 46  $\mu$ s, compared with the measured value of 50  $\mu$ s. This shows that FORTH handles interrupts at the hardware speed of the machine if implemented in the stand-alone configuration.

### 11/23 DEDICATED INTERFACE

Instruction Times at ISR Start.

|            | Instruction | Source | Mode  |       | Time ( $\mu$ s) |       |       |  |
|------------|-------------|--------|-------|-------|-----------------|-------|-------|--|
|            |             |        | Dest. | Basic | Source          | Dest. | Total |  |
| MOV        | 10046       | 0      | 4     | 2.025 | 0               | 2.025 | 4.05  |  |
| BIC        | 42737       | 2      | 3     | 2.025 | 1.425           | 4.275 | 7.73  |  |
| MOV        | 12737       | 2      | 3     | 2.025 | 1.425           | 3.15  | 6.60  |  |
| BIS        | 52737       | 2      | 3     | 2.025 | 1.425           | 4.275 | 7.70  |  |
| TOTAL TIME |             |        |       |       |                 |       | 26.08 |  |

### 11/23 DEDICATED INTERFACE (cont'd.)

Thus,

|                     |                                |
|---------------------|--------------------------------|
| Software overhead   | 26.08 $\mu$ s                  |
| Int. latency manual | 9.75 $\mu$ s                   |
| ADC Conversion      | 10.0 $\mu$ s                   |
|                     | <u>45.83 <math>\mu</math>s</u> |

|                      |             |                                        |
|----------------------|-------------|----------------------------------------|
| Stand alone measured | 50 $\mu$ s  | <u>4 <math>\mu</math>s Deficiency.</u> |
| RSX Measured         | 100 $\mu$ s |                                        |

Thus, additional overhead added by RSX operating system is 50  $\mu$ s on 11/23.

For 11/44 this additional overhead is 28.4  $\mu$ s  
(Both 411 & 3920 give same value)

Table 2

Theoretical times for interrupt response for an 11/23 processor running stand-alone FORTH. The times are taken from the DEC processor manual and compared with the measured time between the ADC trigger and the first dataway cycle.

## 7. CONCLUSIONS

The measurements described here show that FORTH, implemented as a stand-alone system, gives the user access to the hardware speed of the computer for servicing interrupts when run on PDP11 machines. When speed of response is important, this is a significant advantage of the stand-alone system over the RSX implementation. Measurements of interrupt latency made by Clark et al. (3) with the RT11 operating system suggest that the hardware speed of the machine is available to the user for interrupt service with RT11, but implementations of FORTH under this system have not been tested here.

It is necessary to distinguish between interrupt response and maximum possible interrupt frequency. FORTH as a stand-alone system will always provide better interrupt response than the RSX implemen-

tation, but although the stand-alone system has roughly half the interrupt overhead of the RSX implementation, when this overhead is a small fraction of the run time of the interrupt service routine, the stand-alone system will not run at a much higher frequency than the RSX implementation. This is because the fastest interrupt service routines must be written in Assembler, and whether this is done from FORTH or using a conventional assembler, the same machine code will result.

It is possible, with CAMAC, to run a PDP11/23 at an interrupt frequency of 14 kHz and a PDP11/44 at 30 kHz with the stand-alone FORTH and a minimal interrupt service routine. The corresponding figures for the RSX implementation of FORTH are 6.7 kHz for the 11/23 and 12.5 kHz for the 11/44.

### REFERENCES

1. C. H. Moore, Astron. Astrophys. Suppl. 15 (1974) 497.
2. Starting FORTH. L. Brodie, Prentice-Hall (1981).
3. D. L. Clark, T. S. Lund, J. M. Melvin, IEEE Trans. Nuclear Sci. 30 (1983) 3804.

LUBRICANT LABORATORY INFORMATION  
MANAGEMENT SYSTEM

Andrew M. Wims and Ching Po Wang  
GM Research Laboratories  
Warren, Michigan 48090-9055

ABSTRACT

Computer programs have been designed and successfully implemented on our departmental VAX-750 computer for management of sample descriptive information, storage of analytical test data, and preparation of analysis reports. Any of the programs can be selected from a menu displayed on entry into the lubricant directory. A special sample analysis request form is displayed at the terminal which simplifies the task of inputting the sample analysis request information.

INTRODUCTION

The applications for computers in analytical laboratories are increasing at a rapid rate. Because a modern laboratory contains instrumentation which is automated for both data collection and data reduction, the next logical step is the development of a laboratory information management system (LIMS) to further increase productivity. LIMS is a comprehensive data base management system for the laboratory which usually runs on a large computer. Numerous articles have appeared in the literature in the past few years describing the features of LIMS (1-10). Some of the typical benefits of LIMS are shown in the sample and information flow diagram in Fig. 1.

Five year ago a LIMS software package was successfully developed on a Honeywell time share computer for the Petroleum Products Testing group in the Analytical Chemistry Department. That system was designed to maintain files, store administrative and analytical information, and provide computer-prepared reports for approximately 3000 lubricant samples per year with up to 16 analytical requests per sample. This original system demonstrated that the managing, recording, and reporting on samples could be handled effectively in our analytical laboratory using a computer.

Because of our initial success, we decided to obtain a large 32-bit computer for implementation of a complete LIMS package for the department. Two factors played an important role in our selection of the hardware and software: 1. the implementation of an integrated laboratory wide office system capability (11) with several hundred professional workstations (refer to Fig. 2) networked to the mainframes, and 2. centralized support in the Computer Science Department for other departments that need their own computer for laboratory applications.

The first phase of the laboratory-wide system has included 240 workstations (~ 1100 over five years), laser printers, and a DEC VAX. The Computer Science Department also selected a DEC VAX and is providing a range of consulting services, training, system support, and a centralized VAX facility. Currently, a broadband network connects

the workstations to the VAX, IBM, and CRAY computers. Ethernet (12) also is used to connect many of the VAX's together. A network diagram is shown in Fig. 3. With these developments occurring, we selected a VAX for our departmental needs. Once that decision was made, selection of the DEC LIMS software package naturally followed.

Although our VAX-750 and most of the software products have been installed, the complete DEC LIMS software package will not be available until mid-year. Thus, in the interim, we converted the Fortran code for our lubricant LIMS software to VAX Fortran. (A block diagram of lubricant LIMS is shown in Fig. 4.) In the conversion to the VAX, we have made many improvements to the lubricant LIMS based in part on user suggestions and special features available on the VAX. Any of the programs in this new package can be selected from a menu (Fig. 5) displayed on entry to the oil directory. Many features of the Lubricant LIMS will be used in the development of a complete LIMS system later this year.

DESCRIPTION OF LUBRICANT LIMS PROGRAMS

Input of Analytical Requests

Sample and Report Form. A new form was developed with a structured format that reduces any uncertainty in input requirements of the sample descriptive and analysis request information (see Fig. 6). For example, a secretary is no longer required to count characters in a sample description, identification, or any other sample input information to ensure that computer limitations on string length are met. In addition, multiple lubricant samples (up to eight), all requiring the same analyses, can now be reported on one report form. The sample description must be the same (up to 36 characters) for all the samples reported on a single form.

The test method code which was initially entered as a character string is now entered as a two digit number. The test numbers, test symbols, full name of the tests, and units are printed on the reverse side of the form (see Fig. 7). A menu option is available to list this test information at the terminal, sorted by number or in alphabetical order. The two digit number facilitates

grouping test methods and assigning additional numbers for new tests to the proper group. This approach also makes possible the printing of test methods in a specific sequence for data presentation in a final report. On inputting the sample descriptive information, the test numbers can be entered in any order. The program sorts the test numbers in increasing order.

Storage of Sample Information in Computer Files. RESAMP is a computer program that allows the input and storage of sample descriptive and analyses request information into computer data files. The original version of this program required an input response to a query, one at a time. With this approach, an input error can not be easily corrected after a response has been given. In the present program, the DEC forms management system software (FMS) was used to design a FMS screen display (Fig 8) that simulates the actual sample request form. The main advantage of FMS is that a secretary can supply the requested information and return, if necessary, to a screen location to change a previous response. An additional advantage is that restrictions to any input field can be imposed to minimize the chance for an improper response. The sample submitter can also conveniently respond to the questions on the FMS screen.

Two files, MASTER.DAT and ANFILE.DAT, are generated during the execution of RESAMP. MASTER.DAT contains all the sample descriptive information. ANFILE.DAT contains all the test symbols and data. Data values of zero are initially stored in the file.

System Files. A feature of the original version of the code was that the files MASTER.DAT and ANFILE.DAT continually increased in size as new sample analysis request information was added to the system. Because these files are searched by other programs, the search time was increasing to perform a particular operation. In the current version of the code, this is no longer a problem. When the final report is generated (discussed in a later section), the corresponding information in MASTER.DAT and ANFILE.DAT is removed and archived in two new files, MASTOT.DAT and ANATOT.DAT. Thus, the original files contain only the information for the current samples being analyzed.

#### Input of Analytical Results

INDATAN is a program that is used by the analyst to input analytical test data into ANFILE.DAT. This program is designed primarily for the input of data collected for one particular test type on a number of samples. The analyst responds to questions for analytical test symbol, number of analytical results, and whether the samples have consecutive identification (certificate) numbers. If the user supplies an incorrect analytical symbol that is not in the symbol dictionary, the analyst is informed and again prompted for the symbol. For the case of consecutive certificate numbers, only the initial certificate number is required. Otherwise, the certificate number and the data point for each sample is entered. A very convenient verification feature can be used to view the input data and make corrections if necessary before the data is sent to ANFILE.DAT (refer to Fig. 9).

An experimental test result is a string of up to eight characters. The data can include in the string a less than or greater than character (<,>), a plus or minus sign or NR for no result. A menu option is also available for listing, by analytical test symbol, the samples (certificate numbers) that need to be analyzed.

JOHNI is another program for inputting data into ANFILE.DAT. This program is faster to use when data has been collected on one sample for a number of different analytical test, which is the case with spectrographic results. The other features of this program are similar to INDATAN.

#### Output of Analytical Reports

Final Reports. Two programs, OUTPRO1 and REPCOL1, are used to generate final reports. These programs only need to be selected for execution from the main menu; no other user input is required. OUTPRO1 searches the file ANAFIL.DAT to find every analysis request for which all the work is completed. For example, if six samples for 20 analytical tests were submitted, the data for all six samples would have to be in the file before OUTPRO1 would select that request for a final report. After searching the entire file, OUTPRO1 generates a file called OUTFIL.DAT that contains the test data for each analysis request ready for a final report.

The program REPCOL1 selects the administrative data from MASTFIL.DAT and the test data from OUTFIL.DAT and generates a final report for each completed request. This new program, REPCOL1, can generate a one page report in column and row format for up to eight samples as specified on the input sample request form. An example of a final report is shown in Fig. 10.

Status Reports. An important feature of the VAX version is that the sample submitter can conveniently obtain a status report from a remote terminal location. This feature is implemented using a captive account on the VAX which limits access to the OIL directory. On logging into the VAX, the sample submitter's name is asked, and a table of descriptive information, including certificate numbers, is displayed (refer to Fig. 11) for all matches to the name. A certificate identification number can then be selected to obtain a status report. At this point in the program, a status report can not be obtained if the results have already been made available in a final report.

Archival Reports. Once a final report has been obtained, the status report option can not be used to review that information. However, an archival report can be obtained using a procedure similar to the one described above.

#### Summary

A special sample analysis request form simplifies the task for a requester to describe the samples and to select the specific analytical tests needed. In addition, a secretary can easily transfer the information from the request form to the computer files using a program designed to minimize the possibility of an incorrect reply. The other

important new capabilities provided on the VAX computer version are:

1. The removal of information from the active files for storage in archival files when a final report is printed. Thus, active files are kept small, thereby reducing file access time.
2. A single page final report can be obtained for up to eight samples and 24 analytical tests.
3. Sample submitters can obtain reports on current status of analyses or archived data.
4. Two programs are available for input of analytical results grouped by test or by sample.

ACKNOWLEDGMENTS

The authors want to thank Professor Gerald G. Johnson, Jr. of the Pennsylvania State University for his contributions on the VAX version of the programs. We also want to thank Messrs. Alex Peat and Mike Klim for their assistance during the evaluation phase of the new programs.

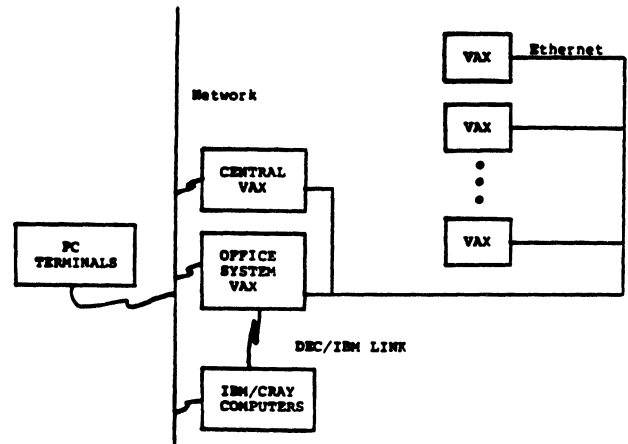


Figure 3. Network Communication Links to Computers.

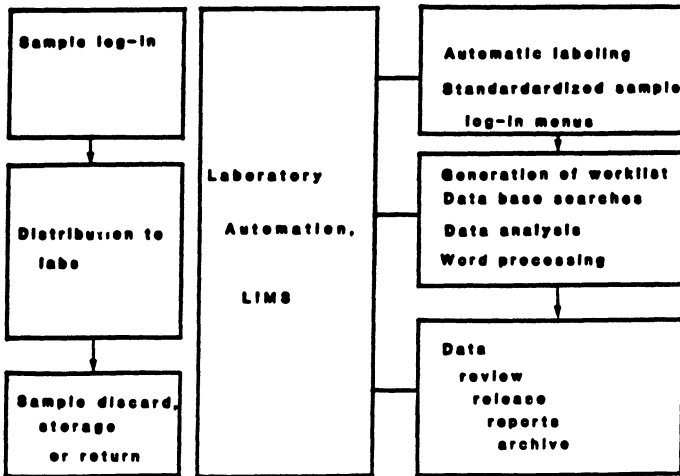


Figure 1. Sample and Information Flow in the Laboratory.

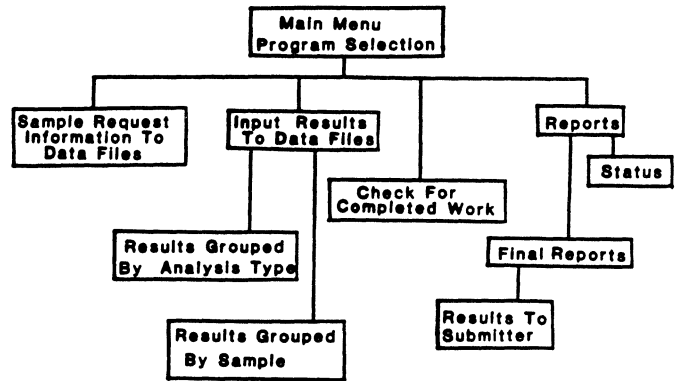


Figure 4. Block Diagram for Lubricant LIMS.

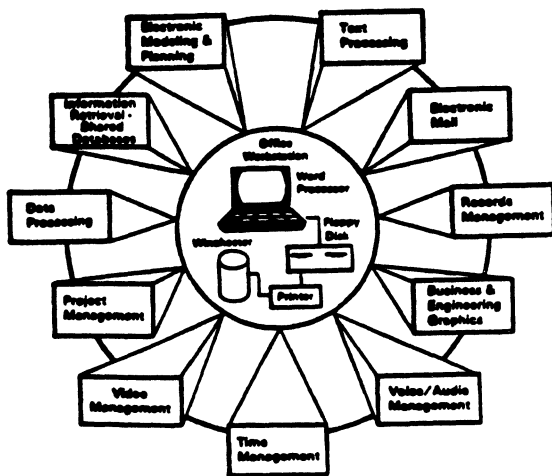


Figure 2. Workstation System Applications.

YOU HAVE THE FOLLOWING CHOICES:

- |                                |            |
|--------------------------------|------------|
| 1. INPUT OF ANALYTICAL REQUEST | (REBAMP)   |
| 2. INPUT RESULTS BY TEST TYPE  | (INDATAN)  |
| 3. INPUT RESULTS BY SAMPLE     | (JOHN1)    |
| 4. OUTPUT OF THE TEST RESULTS  | (OUTPRO1)  |
| 5. FINAL REPORT                | (REPCOL1)  |
| 6. BACKLOG BY ANALYTICAL TEST  | (ANASTATA) |
| 7. STATUS ON SAMPLE            | (STATUS)   |
| 8. LIST OF TESTS               |            |
| 9. TERMINATE                   |            |

Please enter your CHOICE:

Figure 5. Menu for Selection of Program Option

**OIL SAMPLE ANALYSIS REQUEST**

Analytical Chemistry Department  
 General Motors Research Laboratories  
 Warren, MI 48090-9085

Certificate No.(s) - Office use only: \_\_\_\_\_

Charge Number:  -  -

Requested by:

Location (Dept., Div., Staff):

Date Submitted (mo. - day - yr.):  -  -

Description of Sample(s): For up to 8 samples per request  
 all requiring the same analysis

Sample Number(s): Assigned by the submitter and also placed on the sample container.  
 Prefix the sample number with the submitter's initials.

(1)  -  (5)  -

(2)  -  (6)  -

(3)  -  (7)  -

(4)  -  (8)  -

Analysis Request: Identified by the test number for each method. (Use other side.)

(1)  (7)  (13)  (19)

(2)  (8)  (14)  (20)

(3)  (9)  (15)  (21)

(4)  (10)  (16)  (22)

(5)  (11)  (17)  (23)

(6)  (12)  (18)  (24)

Comments: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Figure 6. Analysis Request Form.

| TEST NO.          | TEST METHOD                               | TEST NO.                  | TEST METHOD                       |
|-------------------|-------------------------------------------|---------------------------|-----------------------------------|
| <b>GRAVITY</b>    |                                           | <b>DILUTION</b>           |                                   |
| 01                | SG Specific Gravity at 15.6 C             | 45                        | FD Fuel Dilution (X)              |
| 02                | APG API Gravity at 15.6 C                 | 46                        | WD Water Dilution (X)             |
| 03                | DEN Density at 15 C (g/ml)                | 47                        | GLY Glycol (ppm)                  |
| <b>FLUIDITY</b>   |                                           | <b>INFRARED</b>           |                                   |
| 05                | VL Viscosity at 40 C (cs)                 | 55                        | IR Infrared Spectrum (date comp.) |
| 06                | VM Viscosity at 100 C (cs)                | 56                        | DIR Differential IR (date comp.)  |
| 10                | CC1 Cold Crank Viscosity at -18 C         | 57                        | CAR Carbonyl, IR (abs.)           |
| 15                | BF1 Brookfield Visc at 0 F (cP)           | 58                        | SDO Soot, IR (X)                  |
| 16                | BF2 Brookfield Visc at -10 F (cP)         | <b>THERMAL ANALYSIS</b>   |                                   |
| 17                | BF3 Brookfield Visc at -20 F (cP)         | 65                        | TGA Soot, TGA (X)                 |
| 18                | BF4 Brookfield Visc at -30 F (cP)         | 66                        | DSC Diff. Scan. Cal. (C, min)     |
| 19                | BF5 Brookfield Visc at -40 F (cP)         | <b>ELEMENTAL ANALYSIS</b> |                                   |
| 22                | PF Pour Point (degrees C)                 | 70                        | ASH Ash (X)                       |
| <b>OXIDATION</b>  |                                           | 71                        | SA Sulfated Ash (X)               |
| 24                | TAN Total Acid Number                     | 72                        | CL Chlorine (X)                   |
| 25                | TBN Total Base Number                     | 73                        | N Nitrogen (ppm)                  |
| 26                | PII Pentane Insolubles (X)                | 74                        | S Sulfur (X)                      |
| 27                | TII Toluene Insolubles (X)                | 85                        | P Phosphorus (X)                  |
| 28                | RES Resin (X)                             | 86                        | CA Calcium (X)                    |
| <b>VOLATILITY</b> |                                           | 87                        | ZN Zinc (X)                       |
| 30                | FLP Flash Point (degrees C)               | 88                        | BA Barium (X)                     |
| 31                | FIP Fire Point (degrees C)                | 89                        | MG Magnesium (X)                  |
| 32                | GC GC Simulated distillation (date comp.) | 90                        | PB Lead (X)                       |
| 36                | PE1 Pan Evaporation at 177 C              | 91                        | CU Copper (X)                     |
| 37                | PE2 Pan Evaporation at 204 C              | 92                        | B Boron (X)                       |
| 38                | PE3 Pan Evaporation at 232 C              | 93                        | FE Iron (X)                       |
| 39                | PE4 Pan Evaporation at 288 C              | 94                        | SI Silicon (X)                    |
|                   |                                           | 95                        | AL Aluminum (X)                   |
|                   |                                           | 96                        | CR Chromium (X)                   |
|                   |                                           | 97                        | NI Nickel (X)                     |
|                   |                                           | 98                        | NA Sodium (X)                     |
|                   |                                           | 99                        | MN Manganese (X)                  |

Figure 7. List of Test Methods (Printed on Reverse Side of Analysis Request Form).

**OIL SAMPLE ANALYSIS REQUEST**

Certificate Number: 554677 Charge Number: 22-5000-600  
 Requested by: A B JONES Location: FUELS  
 Date Submitted: 03/21/85

Description of Sample(s): For up to 8 samples per request  
 all requiring the same analysis  
**OIL TEST SAMPLES FOR FLEET**

Sample Number(s): Assigned by the submitter and placed on the sample container.  
 Prefix the sample number with the submitter's initials.

ABJ-1234 ABJ-1235 ABJ-1236 ABJ-1237  
 ABJ-1238 - - -

Analysis Request: Identified by the test number for each method.  
 05 06 23 24 27 28 09

Comments: SAMPLES WILL BE AVAILABLE TODAY

Is there another request to input? [Y/N] Y

Figure 8. Examples of FMS Screen Display.

```

RUN INDTAN
PROGRAM INDTAN VERSION AUG 8, 1984
ENTER ANALYTICAL SYMBOL FE
ENTER NUMBER OF RESULTS ((101) 7
IS DATA TO BE ENTERED FOR CONSECUTIVE CERT NOS. (YE/NO) YE
ENTER INITIAL CERT. NUMBER 556789
ENTER DATA POINT-HIT RETURN
556789 7.0
556790 5.6
556791 2.8
556792 23.8
556793 34.6
556794 10.6
556795 5.9

***VERIFY CERT. NUMBER AND DATA FOR FE
NUM CERT DATA
1 556789 7.0
2 556790 5.6
3 556791 2.8
4 556792 23.8
5 556793 34.6
6 556794 10.6
7 556795 5.9
HOW MANY LINES NEED TO BE CORRECTED 0
DO YOU WANT TO ENTER OTHER ANALYTICAL DATA (YE/NO) NO

```

Figure 9. Example of Input of Analytical Results into LIMS, Showing Verification Feature.

```

OIL SAMPLE REPORT
ANALYTICAL CHEMISTRY DEPARTMENT
RESEARCH LABORATORIES
GENERAL MOTORS CORPORATION
WARREN, MICHIGAN

Charge No 22-4019-700
Requested By S. S.
Date Reported 02/08/85
Date submitted 02/21/85

Sample Description MEDH FLEET
Requester Code SES
Location FUELS

Cert. No.: 540553 540554 540555 540556 540557 540558 540559 540560
Requester No.: M151 M140 M148 M152 M153 M155 M150 M146

5 Viscosity at 40 C (cs) 64.3 64.1 61.3 89.7 73.6 73.4 66.0 80.9
6 Viscosity at 100 C (cs) NA 9.08 8.85 11.17 9.30 9.73 9.58 10.29
24 Total Acid Number 3.60 9.43 9.45 NA 8.10 1.83 6.30 11.0
25 Total Base Number 4.70 2.61 10.9 19.6 23.3 4.97 3.13 8.48
34 Pentane Insolubles (X) .87 1.02 .88 .74 .33 1.85 .12 .40
45 Fuel Dilution (X) 1.15 1.36 <.1 <.1 <.1 <.1 <.1 <.1
46 Water Dilution (X) 1.5 .22 <.1 <.1 <.1 <.1 <.1 <.1
55 Infrared Spectra (date comp.) 3-85 3-85 3-85 3-85 3-85 3-85 3-85 3-85
64 Diff. Scan. Cal. (C/min) 16 3 28 3 69 39 3 13
85 Phosphorus (X) .11 .12 .13 .14 .17 .14 .10 .13
86 Calcium (X) .20 .10 .40 >.5 >.5 >.5 .18 >.5
87 Zinc (X) .12 .13 .14 .18 .15 .15 .13 .15
89 Manganese (X) <.01 .05 <.01 <.01 <.01 <.01 <.01 <.01
90 Lead (X) .04 .02 <.01 .12 .03 .04 .09 .02
91 Copper (X) <.001 .002 <.001 .001 <.001 <.001 <.001 <.001
93 Iron (X) .098 .079 .088 .014 .003 .012 .067 .011
94 Silicon (X) .003 .013 <.001 .003 .002 .008 .003 .001
95 Aluminum (X) .005 .008 .002 .010 .001 .005 .010 .005
96 Chromium (X) <.001 .001 <.001 .001 <.001 .002 <.001 .001
98 Sodium (X) .10 .02 .02 .03 .03 .02 .10 .02

```

Figure 10. Example of a Final Report.

```

O R STATUS
Please give the NAME as submitted in the request C. K.

SES-M134 20-FEB-85 MEDH FLEET 540546
SES-M138 20-FEB-85 MEDH FLEET 540547
SES-M144 20-FEB-85 MEDH FLEET 540548
SES-M157 20-FEB-85 MEDH FLEET 540549
SES-M159 20-FEB-85 MEDH FLEET 540550
SES-M145 20-FEB-85 MEDH FLEET 540551
SES-M136 20-FEB-85 MEDH FLEET 540552
SES-M151 20-FEB-85 MEDH FLEET 540553
SES-M160 20-FEB-85 MEDH FLEET 540554
SES-M148 20-FEB-85 MEDH FLEET 540555
SES-M132 20-FEB-85 MEDH FLEET 540556
SES-M133 20-FEB-85 MEDH FLEET 540557
SES-M135 20-FEB-85 MEDH FLEET 540558
SES-M130 20-FEB-85 MEDH FLEET 540559
SES-M146 20-FEB-85 MEDH FLEET 540560

Do you want to view the status of the analysis ? (Y/N) Y

Please enter the CERT # that you want to examine 540558
UL 0. UN 0. TAN 0. TBN 0. PIN 0. FD 0. HD 0. IR 0.
DSC 0. P 0. CA 0. ZN 0. MB 0. PB 0. CU 0. FE 0.
SI 0. AL 0. CR 0. NA 0.

Do you want to view another sample ? (Y/N)

```

Figure 11. Example of a Request for a Status Report.

REFERENCES

- G. A. Gibbon, "Trends in Laboratory Information Management Systems," Trends in Analytical Chemistry, Vol. 3, No. 2, 1984, p. 36.
- R. E. Dessey, "Laboratory Information Management Systems: Part I," Analy. Chem. 55, No. 1, 1983, p. 70A.
- R. E. Dessey, "Laboratory Information Management Systems: Part I," Analy. Chem. 55, No. 2, 1983, p. 277A.
- J. H. Golden, "Computerizing the Laboratory: The Importance of System Specification," Amer. Lab., 12, No. 11, 1980, p. 111.
- J. G. Liscouski, "Distributed Laboratory Data Collection and Management," Amer. Lab., 15, No. 9, 1983, p. 127.
- M. Podany and J. Vezina, "Real Time Multi-Processor Data Acquisition Network," Proceedings of the Digital Equipment Computer Users Society, Cincinnati, Ohio, June 1984, p. 147.
- L. Malkenson, "Use of Computers in a Laboratory of Cardiovascular Medicine," Proceedings of the Digital Equipment Computer Users Society, Cincinnati, Ohio, June 1984, p. 153.
- K. Lewis, F. Chow and E. Cassaro, "Interfacing Laboratory Data Systems to a VAX," Proceedings of the Digital Equipment Computer Users Society, Las Vegas, Nevada, October 1983, p. 247.
- S. E. Stern and G. G. Johnson, Jr., "A Generalized Laboratory Automation Scheme for a Group of Different Analytical Instruments," Computer Automation of Materials Testing, ASTM STP 710, editor B. C. Wonsiewicz, American Society for Testing & Materials, Philadelphia, PA., 1980, p. 59.
- R. J. Betsch and G. G. Johnson, Jr., "Bridging the Hardware/Software Gap in Instrument Control," Computer Automation of Materials Testing, ASTM STP 710, Editor B. C. Wonsiewicz, American Society for Testing & Materials, 1980, p. 11.
- C. Snyder, "Managing the Electronic Laboratory: Part II," Anal. Chem. 56, No. 7, 1984, p. 855A.
- J. E. McNamara, Technical Aspects of Data Communication, 2nd ed., Digital Press, Bedford Mass., p. 230.





Jean M. Lareau  
 34 Windham Rd.  
 Willimantic, CT.  
 06226

ABSTRACT

The objective is to communicate in an asynchronous mode between a RAINBOW 100 and the DCT11-EM EVALUATION BOARD. The two computers will be interfaced through the RS-232 serial ports. The program segments can be used in general communications such as RAINBOW to RAINBOW or RAINBOW to PDP11 and, of course, RAINBOW to DCT11-EM. I have already written programs that do these communications so it can be done with some effort but these programming segments will give you almost everything you'll need to know. Segments of the program will be illustrated to clarify the points being made. Before entering the programming section, we should discuss the cable that is needed to connect the two computers together.

INTERFACE CABLE

The cable only needs six lines which are the protective ground, transmit data, receive data, data set ready, signal ground and data terminal ready. They should be connected in the following manner:

| Mnemonic  | Pin Number | Pin Number | Mnemonic  |
|-----------|------------|------------|-----------|
| PROT GND  | 1 -----    | 1          | PROT GND  |
| XMIT DATA | 2 -----    | 3          | REC DATA  |
| REC DATA  | 3 -----    | 2          | XMIT DATA |
| DSR       | 6 -----    | 20         | DTR       |
| GND       | 7 -----    | 7          | GND       |
| DTR       | 20 -----   | 6          | DSR       |

CLEAR INTERRUPTS ON RAINBOW 100

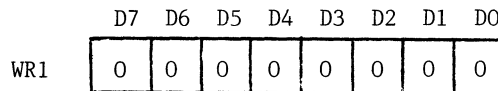
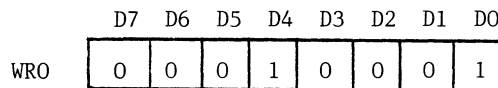
Now we can consider the programming aspect of the communication. The program segments are written in basic and the numerical values are in decimal. The first thing that has to be done is to clear the interrupts. The communications port is handled by the NEC 7201 chip. This is done by writing a 16 to WRO and writing a 0 to WR1 of the communications control/status register which is port 66 in decimal. The way this is completed is by:

```
1000 OUT 66,17
1020 OUT 66,0
```

- OUT = sends a byte of information to an output port
- 66 = communications control/status register
- 17 = sets RESET EXT/STATUS INTERRUPTS on and selects WR1
- 0 = sets (WR1) register to 0

When using the NEC 7201 chip, two OUT statements must be issued. The first OUT statement is to set WRO and to set

the pointer register you want to address next. Bits DO - D2 of WRO are used as a pointer to the next register. The second OUT statement is used to set the pointer register selected by bits DO - D2 of WRO. After the second OUT statement, the control returns back to WRO. After the two OUT statements have been issued, the WRO and WR1 are as follows:



SET INTERRUPTS

After execution of the program, the interrupts should be reset back to the original state. This is done by the following two lines.

```
1200 OUT 66,17
1220 OUT 66,24
```

- 66 = communications control/status register
- 17 = sets RESTE EXT/STATUS INTERRUPTS on and selects WR1
- 24 = sets WR1 to INTERRUPTS ON ALL RECEIVE CHARACTERS

BUSY WAIT

One very important subroutine in asynchronous communications is the BUSY WAIT subroutine. It is vital for detecting the status of the other computer. Using this subroutine, you can determine whether or not you have received a character. This is done by ANDing the

status/control's WRO with 1. If the result is true, then there is a character there.

```
4240 IF ((INP(66) AND 1) = 0) THEN
 GOTO 4240
```

The INP(66) simply means to read port 66 ( which is the communications status/control register ) then AND it with 1 and check to see if the results are equal to 0. If the results are equal to 0 then keep on checking until the results change.

#### TERMINAL MODE SIMULATION

Terminal mode simulation is needed for the ability to read from and write to another computer. This subroutine will allow for a bypass of your own computer's operating system and allow you to run the other computer under it's own operating system. The subroutine is as follows:

```
1200 IF ((INP(66) AND 1) = 0) THEN
 GOTO 1240
1220 PRINT CHR$(127 AND INP(64))
1240 B$ = INKEY$
1260 IF B$ = "%" THEN GOTO 1400
1280 IF B$ = CHR$(24) THEN OUT 64,3
1300 IF B$ <= CHR$(7) THEN GOTO 1200
1320 OUT 64,ASC(B$)
1340 GOTO 1200
1400 END
```

Line 1200 checks to see if a character has been sent by the other computer. If there is a new character then it will be printed. If no new character is found, then print statement will be skipped.

Line 1220 will read the COMMUNICATIONS DATA REGISTER and AND it with 127. This is done to strip the incoming data of such things as parity bits. The character value will then be printed to the screen.

Line 1240 will read anything typed on the keyboard and store it into the variable B\$.

Line 1260 will exit the program when the "%" (percent sign) is typed on the keyboard.

Line 1280 will check if the character type in is a CTRL/Z character. If it is, a CTRL/c will be sent out to the other computer.

Line 1300 will check if the typed in character is a printable character. If the character is non-printable then the program will jump back to the busy wait.

Line 1320 will send to the other computer the character typed in.

#### READ A CHARACTER FROM DCT

Reading a character is accomplished by performing two steps. The first step is the busy wait. The second step is read the COMMUNICATIONS DATA REGISTER and store the contents into the variable.

```
4320 IF ((INP(66) AND 1) = 0) THEN
 GOTO 4320
4340 A = INP(64)
```

#### READ AND STORE CHARACTERS FROM DCT

The following subroutine will read characters from the DCT and also store them into a string. The string will terminate when a carriage return is received. A carriage return being a 13 in ascii value.

```
4300 CC$ = " "
4320 IF ((INP(66) AND 1) = 0) THEN
 GOTO 4320
4340 A = INP(64)
4360 CC$ = CC$ + CHR$(A)
4370 IF CC$ + "HALT" THEN RESET: GOTO 5000
4380 IF A = 13 THEN PRINT #2, CC$:CC$=""
4400 GOTO 4320
5000 END
```

Line 4300 will set string to null.

Line 4320 executes BUSY WAIT.

Line 4340 reads a character from the COMMUNICATIONS DATA REGISTER and stores it into the variable named A.

Line 4360 will incorporate the characters being read in and store them into one large string which will be delimited by a CR(carriage return) in the following steps.

Line 4370 will check for the sentinal.

When the word "HALT" has been read in as the very first four characters, then all the opened files will be closed and the program will end.

Line 4380 will check for a CR. If a CR has been received, the string will be printed to the exterior file and the string will be set to null.

#### SENDING OUT A STRING TO THE DCT

When sending a string to the other computer, the following steps must occur.

First, the length of the string must be determined. This is done with the LEN function. Store the length of the string into a variable. The length of the string will determine the number of iterations that the FOR loop must complete.

The MID\$ function is used to isolate character/s from the string and store them into a new string. In communications such as this, we can only send out one character at a time. The MID\$ function will perform in the following manner. It will look at the original string (B\$) and start at the Ith character (I) and take the next N characters. In our case, only one character is desired.

The next step is to send out that one character in ASCII form. It must be sent in ASCII form or the receiving computer won't be able to read it. The next step is to increment the I variable to get the next character of the string. The last step is to send out a CR to tell the other computer that sentence has been completed.

```
3980 B$ = "PASS2"
4000 L = LEN(B$)
4020 FOR I = 1 TO L
4040 BBB$ = MID$(B$,I,1)
4060 OUT 64,ASC(BBB$)
4080 NEXT I
4100 OUT 64,13
```

## VIEW AN ASCII FILE

I Have included this subroutine as a method by which you can view the program that your computer has received and saved. It will ask you which file do you wish to view.

It will then open that file. It will then print to the screen, line by line, the program until the END-OF-FILE has been reached at which point the file would then close.

```
9270 PRINT "PLEASE ENTER FILE NAME"
9280 INPUT T$
9300 OPEN "I", #2,T$
9320 IF EOF(2) THEN GOTO 9400
9340 LINE INPUT #2,K$
9360 PRINT K$
9380 GOTO 9320
9400 RESET
```

Line 9270 will print message to screen  
Line 9280 will read the file name  
Line 9300 will open the file  
Line 9320 will do until end-of file  
Line 9340 will read line  
Line 9360 will print line to screen  
Line 9380 will goto do loop  
Line 9400 will close all opened files

## DOWNLOAD A FILE FROM DCT ONTO DISK

The read a file and save it on a floppy diskette subroutine will be needed for just that. Almost everything in this subroutine has been mentioned earlier. Some of the lines are specific to the computer that you will communicate with. For our program, this subroutine should appear as follows:

```
3920 PRINT "ENTER NAME OF FILE"
3940 INPUT Q$
3960 INPUT "ENTER BEGINNING ADDRESS",B$
3980 BB$ = "." + B$ + " 177776"
4000 L = LEN(BB$)
4020 FOR I = 1 TO L
4040 BBB$ = MID$(BB$,I,1)
4060 OUT 64,ASC(BBB$)
4080 GOSUB 3820 [READ A CHARACTER
 FORM DCT SUBROUTINE]
4100 NEXT I
4120 OUT 64,13
4180 OPEN "O", #2, Q$
4200 C$=" ": CC$=""
4220 OUT 64,13
4240 GOSUB 3820
4260 GOSUB 3820
4280 GOSUB 3820
4360 CC$ = CC$ + CHR$(A)
4380 C$ = RIGHT$(CC$,4)
4390 IF C$ = "HALT" THEN GOTO 4260
4400 IF A = 13 THEN PRINT #2, CC$:
 C$=" ": CC$="" : GOTO 4280
4420 IF C$ = "END " THEN GOTO 4445
4440 GOTO 4280
4445 RESET
4450 GOTO 1200 [TERMINAL MODE SIMULATION]
```

Line 3980 is specific to the DCT. This string, after being sent, will tell the DCT to send to the RAINBOW the contents of what's in it's memory from the starting address that you specified (BB\$) and end at address 177776. The program will end

upon reading a "END " symbol. For this reason I always put a "END " symbol at the end of each of the DCT programs.

Every time a new character is sent to the RAINBOW, an INP statement must be executed or the DATA REGISTER will not be cleared and that character will stay there until it is read creating a queue of characters. This is the reason for the three GOSUB 3820 statements. After you have sent out the last character of a line, you then send out a CR (carriage return). The DCT will respond by sending back the last character, a CR and a LF(line feed).

## WRITING A FILE FROM A FLOPPY TO THE DCT

Writing a file from either A drive or B drive to the DCT is more difficult than simply sending characters out and sending a CR at the end of every string.

We must concern ourselves with such things as at what address are we loading the program, how does the DCT know that the program has finished or has the DCT accepted the program correctly.

When loading a program in this manner, the DCT will act as an interpreter which will not allow errors to be entered. This program will check and correct these mentioned problems as well as creating a file with an error listing. The program is as follows:

```
4460 PRINT "PLEASE ENTER BEGINNING ADDRESS
OF INSTRUCTIONS ONLY";
4480 PRINT "AN OCTAL ADDRESS BETWEEN 146
AND 177776"
4500 INPUT Z$
4520 Y$="." + Z$
4540 L=LEN(Y$)
4560 FOR I = 1 TO L
4580 X$= MID$(Y$,I,1)
4600 OUT 64,ASC(X$)
4620 GOSUB 3820
4640 NEXT I
4660 OUT 64,13
4680 GOSUB 3820
4700 GOSUB 3820
4720 REM
4740 PRINT "ENTER FILE . EXT ";
4760 INPUT F$
4780 LLL=0
4800 L=LEN(F$)
4820 FOR I=1 TO L
4840 FFF$=MID$(F$,I,1)
4860 IF FFF$="." THEN LLL =(L-I)
4880 NEXT I
4900 PRINT CHR$(27);"2J"
4920 PRINT "PLEASE WAIT FOR THE 'TEM '
BEFORE YOU CONTINUE "
4940 IF LLL=0 THEN FF$=F$+".LST" :GOTO 5020
4960 L = LEN(F$)
4980 LL=(L-LLL)
5000 FF$=LEFT$(F$,LL) + ".LST"
5020 OPEN "O",#2,FF$
5040 OPEN "I",#1,F$
5060 GOSUB 3820
5080 GOSUB 3820
5100 GOSUB 3820
5120 COUNT = 0
5140 IF EOF(1) THEN GOTO 5600
5160 LINE INPUT#1,V$
```

```

5180 L=LEN(V$)
5190 IF MID$(V$,1,1)=";" THEN PRINT #2,V$:
GOTO 5140
5200 FOR I = 1 TO L
5220 U$ = MID$(V$,I,1)
5240 OUT 64,ASC(U$)
5260 GOSUB 3820
5280 NEXT I
5300 OUT 64,13
5320 FOR I = 1 TO 7
5340 GOSUB 3820
5360 UU$=UU$+CHR$(A)
5380 AA$=RIGHT$(UU$,5)
5400 NEXT I
5420 IF AA$="TEM " THEN PRINT #2,V$:UU$=" ":
GOTO 5140
5430 IF V$="HALT" THEN V$=".EVEN" :GOTO 5180
5440 CC$=" "
5460 CC$=AA$
5480 CC$=CC$+CHR$(A)
5500 DD$=RIGHT$(CC$,4)
5520 IF DD$="TEM " THEN PRINT#2,V$,CC$,
"***** ERROR *****":GOSUB 3820:V$="HALT":COUNT
= COUNT + 1:GOTO 5180
5540 IF ((INP(66) AND 1) = 0) THEN GOTO 5540
5560 A=INP(64)
5580 GOTO 5480
5600 REM
5620 REM CLOSE
5640 PRINT #2," "
5660 PRINT #2,,"*****THERE WERE "COUNT"
ERRORS DETECTED *****"
5680 CLOSE #2,#1
5700 PRINT "FILE HAS BEEN LOADED INTO THE
DCT11-EM"
5720 PRINT
5740 PRINT
5760 PRINT "*****THERE WERE "COUNT" ERRORS
DETECTED *****"
5780 OUT 64,3
5800 GOSUB 3820
5820 C$=" ":CC$=" "
5840 C$="PASS2"
5860 L=LEN(C$)
5880 FOR I=1 TO L
5900 CC$=MID$(C$,I,1)
5920 OUT 64,ASC(CC$)
5940 GOSUB 3820
5960 NEXT I
5980 OUT 64,13
6000 GOSUB 3820
6020 GOSUB 3820
6040 GOTO 1180

```

Line 4460 - 4700 sends out a string to the DCT.  
Line 4720 - 4880 checks to see if you entered an EXTention with the filename.  
Line 4940 will put a .LST extention on the filename if no extention was entered.  
Line 4960 - 5000 will delete the EXTention and put a .LST extention in its place.  
Line 5020 - 5040 opens the input file and the new .LST file that will be created.  
Line 5140 - 5300 will send out a string to the DCT.  
Line 5190 will check the string to see if its a comment statement. If it is a comment statement, it will not be sent to the DCT for the purpose of saving time.  
Line 5320 - 5400 will store the reply of the DCT.  
Line 5420 - 5520 will check if the DCT accepted the string without error. If

no error is found, the next line will be sent. If a error has been detected, the DCT will not allow that line to be inputed. The DCT will wait until a valid string is sent.

A dummy line should be sent in place of the string with the error so that after the program has been loaded into the DCT, the dummy string/s can be replaced with the proper syntax of the statement. If the dummy line is not put in, you will not be able to enter any statements at that address because no space was set aside for the error line.

Line 5540 - 5560 performs a BUSY WAIT  
Line 5600 - 5760 prints messages to .LST file and closes it.  
Line 5780 sends the DCT a CTRL/C.  
Line 5720 - 6040 sends a string to the DCT.

**\*\* NOTE \*\***

All numerical values are represented in decimal.

The DCT uses OCTAL values only.

Beverly H. Johnson  
 OAO Corporation  
 1222 N. Main Ave. Suite 307  
 San Antonio, TX 78212

Michael G. Yochmowitz  
 Radiation Sciences Division  
 USAF School of Aerospace Medicine  
 Brooks AFB, TX 78235

G. Carroll Brown  
 Systems Research Laboratories  
 P.O. Box 35313  
 Brooks AFB, TX 78235

#### ABSTRACT

We describe a behavioral control and data acquisition system developed under the RT-11 operating system on a PDP-11/34 and MINC-11/23. This system is used to train and test subjects to perform in a specified manner. The computer turns stimuli on and off, records all responses, and delivers appropriate reinforcements. It can handle up to five subjects simultaneously. At present, 30 discrete behavioral schedules have been completed. The software was designed to be very flexible to accommodate variations of these schedules. It is easy to use and requires no programming background. Interface routines have been written to allow commonly used statistical analysis packages to be run on the data. The software design and typical real-time problems of program size, execution speed, scheduling events, minimizing data file size, and incorporating multiple subjects under RT-11 are discussed.

#### INTRODUCTION

The Behavioral Acquisition and Research System (BARS) was written for the United States Air Force for use in behavioral experiments. It consists of 30 different tasks (called schedules) that are similar in implementation yet appear different to the user.

BARS was developed as a replacement for 1960-vintage digital equipment used to control and collect data from behavioral experiments. The old system was massive in size, collected data on punched paper tape, needed to be hardwired for each experiment, and took a considerable amount of time to troubleshoot. Its replacement was required to

1. control up to 5 test stations,
2. be easy to use by technicians unfamiliar with computers,
3. run a variety of behavioral schedules (now 30) that can be readily modified,
4. be very flexible to accommodate individual experimenters' needs and preferences,
5. allow one schedule to execute immediately after another without operator intervention,
6. provide quick response processing,
7. minimize dependence on one particular computer,
8. run on portable machines such as the MINC (PDP-11/23),
9. provide real-time feedback to experimenters, and
10. allow for data interface to major statistical software packages.

A PDP-11/34 running the RT-11 Single Job (SJ) monitor (version 5.1) was used. The computer is configured with 128K words of memory, two RL01 disks, a KW11-K real-time clock, a DR11-K digital

I/O board, and two DL11 serial lines. With only a few minor modifications, BARS can also run on a MINC. The behavioral testing apparatus is interfaced through the digital I/O and/or serial I/O boards. A variety of testing apparatus can be used, depending on the schedule to be run. One benefit of using the BARS software is that the testing apparatus hardware is completely independent of the software.

All software was written in FORTRAN IV (version 2.5), with the exception of several small MACRO subroutines. The software consists of three phases: input programs, data collection programs, and analysis programs. Each of these phases will be described, followed by a discussion of the solutions found for the problems encountered in the data collection phase.

#### INPUT PROGRAMS

The input programs were designed to be run by a naive user. Experimental setup parameters are input by means of a question and answer session. Most questions include a default answer in brackets: 'HOW MANY STIMULI [1] ?' The user types a carriage return to use the default value, thereby speeding up the input process.

On-line help is available for each question. If the user types '?', a short paragraph is printed describing the input being solicited, along with the valid range for the input value. Then the question is repeated.

All input data can be modified at the end of the input session. This is an important feature, because some of the schedules ask over 50 questions.

After input, all values are displayed with line numbers. For example:

- 1) TYPE: SIDMAN AVOIDANCE
- 2) STUDY NUMBER: 21
- 3) DRUG NUMBER: 4
- 4) NO HOUSE LIGHTS
- STIM. LEVEL CHAN. DESCRIPTION
- 5) 1 1 3 WHITE LIGHT
- 6) RESPONSE CHANNEL: 1
- 7) RS INTERVAL: 5.00 SECONDS
- 8) SS INTERVAL: 2.00 SECONDS
- 9) #LEVELS OF NEG. REINF. FOR NO RESP: 1
- STIM. TYPE LEVEL CHAN DESCRIPTION PROB
- 10) 1 NEG-NO RESP 1 5 BUZZER 100
- 11) DURATION: 1.00 SECONDS
- 12) DATA DISPLAY INTERVAL: 3.0 MINS
- 13) HALT AFTER 0 HOURS, 30 MINS, 0.0 SECS

The lines are numbered consecutively. The program keeps track of which line number corresponds to which entry. If changing a value requires subsequent lines to be added to or deleted from the display, the program automatically renumbers and prints the updated display.

To change an input value, the user types the number of the line containing that value. Then the question is repeated using the user's previous response as a default value. This procedure makes it quick and easy to change any input value.

All 30 schedules in this software system run the SAME input programs. Defaults are set for each variable according to the schedule type. An ASCII file contains 125 indicators that identify the questions to be asked for each schedule. Generally, a '0' means that the question is to be omitted, and a '1' means that the question is to be asked, although '2' through '9' are occasionally used for other purposes. This indicator file can be edited without recompiling any programs, making it very easy to change which questions are asked for each schedule. This is a particularly valuable feature because, in practice, behavioral schedules are not fixed but instead have many possible variations for each schedule. While most of the common variations have been incorporated into this software, there will always be an experimenter who wants a schedule set up in a nonstandard manner. The ASCII indicator file permits him to do this easily without revising any of the program code.

#### DATA COLLECTION PROGRAMS

There are three data collection programs: one to do the preliminary calculations, one to actually run the experiment, and one to create separate data files for each subject. These could run as one program but are used separately because of memory constraints.

The first data collection program uses the information from the user to generate temporary setup files. All of the decision making that can be done before the experiment starts is done at this time. The program generates lists of random numbers and timing values to be used, as well as deciding which stimuli to turn on. If multiple subjects are to be run, the data for all of the subjects are incorporated into the setup files.

The second data collection program reads the setup files generated by the previous program and runs the experiment. Basically, this program turns stimuli on, receives and analyzes responses, delivers appropriate reinforcements, and turns stimuli off. A considerable number of computed decisions are involved in this process. Since all 30 schedules use the same program, features for all of them

were included in this one program. Up to five subjects can be run simultaneously. This program is also capable of chaining schedules; that is, running one schedule after another without user intervention. Responses made by the subject interrupt the system and are received by a MACRO interrupt service routine. Timing is accomplished by the use of a programmable clock running at 1 kHz and an interrupt service routine that responds to 1-minute clock-counter overflows. This program also prints the data in real time using a format appropriate to the schedule being run, writes all the data to one composite data file, and performs other miscellaneous "housekeeping" functions.

The third data collection program separates the data in the composite data file into individual data files for each subject. This facilitates data management and statistical analysis.

#### ANALYSIS PROGRAMS

A variety of analyses can be performed on the data.

Data files contain every event (such as "stimulus on," "reinforcement off," "response received," etc.) that occurred during the experiment and the times at which they occurred. This allows the entire run to be recreated, millisecond by millisecond. Therefore, the experimenter can have no doubt as to exactly what happened during the experiment.

Another advantage of storing every event is that data have not been lost through compression. Additional analysis routines can be written long after all the data have been collected. This is especially helpful for users who want additional data after the initial analysis of the experiment is finished.

For statistical analyses, programs have been written to interface the data to commercial statistical packages. Also, a program is available to plot the data in a wide variety of formats.

#### DATA COLLECTION PROGRAM PROBLEMS

The data collection program presented a variety of problems, some of which are typical of real-time data acquisition programs. These include: 1) program size, 2) execution speed, 3) incorporating multiple subjects, 4) scheduling events, and 5) minimizing data file size.

##### 1. Program Size

Memory available never seems to be enough when programming for the RT-11 operating system. In the case of BARS, 122K words of code and data had to be squeezed into 24K words of memory (RT-11 limit of 32K words less: RMON = 3555 words, I/O page = 4096 words, and DL handler = 486 words). All of the following solutions to this problem were implemented.

1. Using the Single Job (SJ) version of the RT-11 monitor. Version 5.1 of the SJ monitor is 3.3K words smaller than the Foreground/Background (FB) version, and 4.6K words smaller than the Extended Memory (XM) version.

2. Sysgening only the features necessary for the application.

3. Unloading unnecessary device handlers.

4. Compiling routines without line numbers.

5. Linking with \$SHORT. Linking the program with the global \$SHORT saved 838 words by eliminating long system error messages.

6. Developing overlays. The root was kept as

does not improve the problem.

4. RSX-11M tasks in wait state. Each subject has its own task sitting idle waiting for a response. When a response is received, the appropriate task is awakened to service it. This is not feasible for the schedules that receive many responses at a fast rate.

5. RSX-11M one task. All response processing routines can be placed into one task. Then the response data can be sent to the individual tasks after all the time-critical code has been executed. This eliminates the need for tasks to be shuffled in and out to service a response. This possibility looked feasible until a close inspection revealed that so much of the data collection program centers around response processing that very little would be left for the individual programs. So it might as well be all one program.

6. RT-11. Once the decision was made to include all processing in one program, RT-11 was chosen over RSX-11M because of its faster execution speed. The final decision was to continue to use the RT-11 operating system, requiring one program to handle all five subjects.

#### 4. Scheduling Events

Events that occur in this software (such as turning stimuli on and off) are not fixed at certain times. They are scheduled according to what happened during the execution of the schedule, especially in regard to how the subject responded. Therefore, a list of events and times cannot be set up prior to running, and even the list itself is not fixed once it has been set up - both the timing of an event and even its presence on the list may need to be changed after it has been scheduled. The following methods of scheduling events with a way to change the timing were considered.

1. RT-11 ISCHED. The RT-11 SYSLIB routine ISCHED schedules a FORTRAN subroutine to be run at a specified time. This method would not work because a much greater timing precision is required than the one clock-tick (16.7 ms) precision allowed by ISCHED.

2. Foreground/Background programs. A foreground program can be used to do nothing but watch the events and execute them at the proper times; the original data collection program runs in the background. This alternative was discarded because it uses lots of memory (already in extremely short supply), and the only gain is in making the scheduling simpler.

3. Ordered list. A list of events and the times at which they are to occur are listed in order by time. The list is rearranged to accommodate changes in the event timing. This was a feasible solution. A test executed in 49 ms.

4. Linked list. A list of events and the times at which they are to occur is made up in any order; pointers indicate the order in which events are to occur. The linked list approach was selected because it executed the same test as the ordered list in 42 ms - about 7 ms faster than the ordered list approach.

#### 5. Minimizing Data File Size

Every event that occurs and the time at which it occurs is stored in the data file; thus a huge data file could be created in a short period of time. The data file had to be created in a very compact manner. The solution was to store only two

words per event in a binary file.

1. Event word. All information about the event that occurred is stored in only one 16-bit word, requiring very detailed coding of the event, bit by bit.

2. Time word. The time at which the event occurred is stored in one word. This procedure is complicated because timing must be stored to 1-ms precision and experiments can run for 24 hours. Storing the time at 1-ms precision in a 16-bit word means that only 65.535 seconds can be accommodated, which is obviously well under the 24 hours required. The solution was to implement a "time mark" event. Every minute, when the clock overflows, a "time mark" event is written to the data file, indicating that one more minute has elapsed. The time word stored for each event is the number of seconds in the current minute (multiplied by 1000 to make it an integer).

3. Binary file. ASCII files are generally preferred because they are easier to read. But because every bit of every word is used for this data, a '2I6' format would be required. So an ASCII data file would consume three times as much disk space as a binary file. Also, writing data to a binary file is quicker. Binary files were therefore chosen.

#### SUMMARY

BARS has been a challenging software system to develop. The data input programs can be run by a naive user, and the data entered can be easily modified.

All 30 schedules were incorporated into the same programs, which saved a tremendous amount of time in writing and testing the software. This also allowed much greater flexibility for each schedule, because a feature generally found in one schedule can be very easily added to another schedule.

The data collection program included some typical real-time problems. The problem of excessive program size was solved primarily by using overlays, using virtual arrays, and partitioning the one data collection program into three programs. The most significant improvements in execution time were achieved by using FORTRAN inline code, performing calculations early, and streamlining the code. Multiple subjects were incorporated into a single program under RT-11. Event scheduling was achieved with linked lists. And data file size was minimized through detailed coding of events and times in binary files.

Interfaces to standard analysis programs are available, simplifying the subsequent statistical treatment of the data. BARS has developed into an easy-to-use, straightforward program. Its portability to a MINC system makes it an effective tool for the nonprogramming investigator, both in the laboratory and at remote field sites. Its ability to handle many variations of 30 different schedules is one of its greatest assets.



small as possible and three main overlay groups were developed: one to read data from the setup files and initialize, one to collect the data, and one to close files and disable interrupts. This saved 12K words. All routines that actually collect data have to remain in memory simultaneously because overlays from an RL01 disk are too slow for the real-time work - 44 milliseconds (ms) average for this software. Virtual memory overlays that use memory from 32K to 128K words as if it were a disk (through the VM handler) were tried. Overlays from virtual memory averaged 10 ms for this software. But even this was too slow, since several overlays would be required. Therefore, all real-time routines have to reside in memory simultaneously.

7. Virtual arrays. Virtual arrays are arrays placed in memory above 32K words. Large virtual arrays resulted in very substantial memory savings (65K words) while adding very little to execution time.

8. Writing custom MACRO subroutines. Standard DEC subroutines are available for real-time work, such as digital I/O. These routines work well, but are so comprehensive that they are too large. For example, a simple custom MACRO subroutine to place one word in the digital output register only uses 52 words. DEC's 'IDOR' routine and the necessary related routines use 3500 words. Thus, all real-time routines were written from scratch.

9. Dividing program. The three data collection programs were originally designed as a whole but later divided to save 22K words. Everything that could possibly be calculated before collecting the data was put into a preliminary program, and everything that could wait until the data collection was complete was put into a followup program. These programs are chained to one another automatically in order to be transparent to the user.

10. Adding variables. This apparent contradiction actually works rather well. When a number is calculated more than once, it is generally cheaper - both in terms of number of words and execution speed - to assign it a variable name than to recalculate it even once. Even neatly coded programs became smaller by adding variables.

11. Using only one data file. This software can run up to five subjects simultaneously. The most direct and logical approach would be to write one data file for each subject. However, this requires a fair amount of memory. Using only one data file saved 1000 words of FORTRAN buffer space.

## 2. Execution Speed

As with many real-time applications, fast timing is critical. The timing between a subject's response and the delivery of reinforcement is the most important - it must appear instantaneous to the subject. A 200-ms delay can be seen easily, and many subjects can detect a 100-ms delay; therefore, a 50-ms delay was considered the maximum allowed, and a 30-ms delay was preferred.

The computer must perform a lot of calculations and make many decisions during that time. A few milliseconds here and there can make a substantial difference, so even minor changes that would save only a few milliseconds were made. Using all of the following techniques resulted in a response processing time of 28 ms.

1. Inline code. FORTRAN inline code executed about 15% faster than threaded code.

2. Common blocks. Passing data to subroutines through FORTRAN common blocks is much more efficient

than using an argument list.

3. Delay in writing data to file. Finding time to write the data to the data file was a real problem. Writing data to RL01 disks is very slow (writing two words of unformatted data took 11 ms). Writing the data to a data file in virtual memory using the VM handler improved the speed somewhat (7 ms), but still was not fast enough. The solution used was to place data first in a virtual array, which is done very quickly. Then when timing was less critical, write it to the disk data file.

4. Double-buffer data file. This allows data to be written to one portion of the data file while another portion is being physically written to disk. Timing is not affected until writing the last word of the data buffer; then a difference is detectable.

5. Real-time printing. Data is printed during less time-critical sections of the program. Printing on a VT52 terminal takes approximately 1.2 ms per character, which adds up to 84 ms for a 70-character line. This is obviously too slow for time-critical portions of code. The VT100 terminal, incidentally, was considerably slower.

6. Calculations done early. A preliminary program performed all calculations that could be made before the actual data collection started.

7. Floating point processor. A floating point processor was purchased to improve execution speed.

8. Cache memory. Purchasing cache memory was considered as a means of improving execution speed. This software executed about 15% faster with cache memory. While this is a significant amount, the improvement in execution time did not warrant its purchase.

9. Streamline code. There is often a more efficient way of coding a program even when it was done carefully in the first place. This becomes more significant if many changes have been made to the software since its origination. Taking the time to rewrite sections of code were well worth the effort. Many "trivial" changes resulted in greater improvements in speed than anticipated.

## 3. Incorporating Multiple Subjects

A prototype of this software was written for one subject, but expanded software to handle up to five subjects running the same schedule simultaneously was required. The following solutions were considered.

1. One computer per subject. This is very desirable from the software standpoint, but too expensive.

2. DEC's Laboratory Peripheral Accelerator (LPA). This device performs real-time I/O very quickly when running RSX-11M. Unfortunately, it does not hand the information back to the program until its buffer is full. This was unacceptable, because the program needs the response information immediately to act upon it.

3. RSX-11M independent tasks. This provides one program for each subject. The data collection program is so large that there would be room for only three or four subjects rather than the more desirable five. The main problem here is timing. The RSX shuffler, at its fastest, swaps tasks out at every clock tick (16.7 ms). Therefore, with three tasks running, it could take 33.4 ms before the required task was swapped in, 16.7 ms to execute, 33.4 ms while the other two tasks execute, then 10 ms or so to finish processing. This total of 93.5 ms is much longer than the 50 ms allowed to process a response. Swapping at every 2 or 3 clock ticks

**DATA MANAGEMENT SYSTEMS SIG**



# ARTIFICIAL INTELLIGENCE

## What It Is, Where It Has Been, And Where It Is Going

By Terry C. Shannon

THE DEC\* PROFESSIONAL Magazine

Springhouse, PA 19477

### ABSTRACT

*It seems as if artificial intelligence, expert systems and the Fifth Generation have replaced "user-friendly" as the computer buzzwords of choice. Suddenly, everyone knows that AI is becoming a reality, expert systems are commercially viable and the Fifth Generation is just around the corner. Articles touting various aspects and applications of AI technology are appearing with increasing frequency in mainstream and trade publications, but many of these articles are of little value to individuals who lack a background in AI. This paper serves as a novice level introduction to man's past, present and future endeavors to make machines exhibit intelligent behavior.*

Artificial intelligence, the science of making machines mimic intelligent human behavior, has been the subject of dreams, speculation, and prophecy since ancient times. From the talking statues of Greek mythology and the intricate clockwork automata of the 18th century through today's chess-playing programs, expert systems and supercomputers, man has utilized the resources at his disposal in a continuing effort to create machines in his own image.

It's unlikely that we will ever produce a machine that can faithfully emulate every aspect of our humanity, much less process information as efficiently as our minds. However, developments in the past several years bear testimony to the fact that we are on the verge of developing computers that are capable of mimicking some of our higher thought processes. Artificial intelligence has become a reality, and machines designed to implement it efficiently are just over the horizon.

### A BRIEF HISTORY OF AI

Increased research and development efforts have placed the state of the art in AI technology in a constant state of flux. Perhaps the best understanding of where AI is today and where it will be in the future can be gained by first approaching the subject from a historical perspective. Computer-based artificial intelligence has been around for roughly 30 years. In this relatively short timespan the subdiscipline of computer science known as AI has had a career fraught with claims, promises, and failures. However, the concept of AI predates its implementation on the digital computer. As long ago as the first century BC, Hero of Alexandria reputedly designed birds that could fly and sing. The mythology of the Greeks and their talking statues is legion, as are the stories of the talking bronze heads owned by medieval theologians and philosophers. Some of these legends are obscure, others have been well documented.

A well known example of man's ageless effort to create artifacts in his own image is the golem. The word "golem" is derived from Talmudic writings and refers to an unformed or incomplete entity which can be brought to life through a solemn rite. The most famous of the golems is attributable to Rabbi Loew, a fifteenth-century theologian in Prague. Loew's golem was an eight-foot-tall automaton fashioned from clay taken from a nearby riverbank by the rabbi and his assistants. After the rabbi affixed the Hebrew Name of God to the golem's forehead and subjected it to the ap-

propriate incantations, the artificial man came to life and went about such tasks as performing custodial duties in the temple, patrolling the streets of Prague, and spying on potential evildoers during this period of anti-Semitism. It's uncertain why, but Joseph Golem's career was subsequently cut short by some creative de-programming on the part of Rabbi Loew.

Somewhat later, elaborate clockwork automata became the rage in Europe. In the early 1700's, a craftsman named Vaucanson fashioned an artificial duck which imitated many of the functions of a real waterfowl. These efforts were not limited to reproductions of lower animals, as evidenced by the watchmakers and artisans who constructed human figures that could play musical instruments, move in a fashion, and perform even more sophisticated functions. In 1805, a Frenchman named Henri Maillardet built an automaton that could write and draw pictures. A series of rods and cams served as the read-only memory that permitted the machine's articulated arm to produce incredibly complex drawings. "The Draftsman," an automaton representative of this era, is permanently displayed at Philadelphia's Franklin Institute. Beside the figure is an example of its drawing prowess—a three-masted sailing ship. Although this automaton can't mimic thought processes, it does exhibit an almost human degree of artistic skill.

Having embodied their creations with simulated motor skills, craftsmen next turned to fraudulent emulations of human thought. The best known of these creations was Baron von Kempelen's chess-playing Turk, a robot decked out in robes and a turban which sat behind a wooden cabinet and played world-class chess. Ostensibly, the cabinet contained the clockwork mechanism which enabled the mannequin to plan its strategy and move the appropriate chess pieces. The fact of the matter is that a human accomplice hid beneath the Turk's chess table and called the shots. The Turk was celebrated throughout Europe, reputedly besting Napoleon in an 1809 chess match—proving that a well-planned, elaborate hoax can dupe just about anyone. The Turk's career was brought to an abrupt end when, in the middle of a world tour, it was checkmated by a Philadelphia hotel fire. History does not record whether the Turk's accomplice shared this untimely demise.

Finally, the early nineteenth century produced a classic novel, written by a fourteen year old girl, that has a thread of AI woven throughout its structure. The name Mary Shelley may not mean anything to you, but chances are you've read her book, or at least have seen the movie of the same name: Frankenstein. In this novel,

based on a story originally told around a campfire, Shelley recounts scientist Victor Frankenstein's efforts to create a living "man," and the irony of his success: Creating a being is one thing, controlling it, quite another. Shelley obviously had read the golem legends, for Frankenstein's creation was eight feet tall and subject to the same intractability as the Talmudic automatons. Subsequent books, plays and motion pictures bear proof that the creation of a human-like artifact is a recurring theme.

### Modern Times: The Dartmouth Conference

From a twentieth century viewpoint, AI as we know it today had its beginnings at the Dartmouth Conference in Hanover, New Hampshire. Officially called the Dartmouth Summer Research Project on Artificial Intelligence, this 1956 conference brought together a handful of research scientists who were exploring ways to make computers behave intelligently. Included in this group of scientists were such AI notables as John McCarthy, Marvin Minsky, and IBM's Claude Shannon.

Among the research projects spawned at Dartmouth during the summer of 1956 were efforts to construct a system of artificial neurons that would function like the human mind, build a robot that could learn about its own environment, and build a working model of the brain's visual cortex. These proposals certainly sounded bizarre — after all, in 1956, machine intelligence was the stuff of science fiction.

In the heady days subsequent to the conference, researchers began to make even more impressive and exorbitant claims for their new science. It was just a year later that conference participants Allen Newell and Herbert Simon issued the prediction that, within ten years, a computer would be the world's chess champion, would be capable of composing aesthetically pleasing music, and would derive the theorems of the Principia Mathematica.

Needless to say, history proved these prognostications to be somewhat premature. AI became cloaked in a pall of skepticism and its practitioners were referred to as hucksters, charlatans, and worse. Interest in the field waned and was confined mainly to graduate students majoring in cognitive psychology, and to researchers at institutions like Stanford, MIT, and Carnegie-Mellon University.

This "Dark Ages" period of artificial intelligence elicited a barrage of anti-AI journalism and oratory. Perhaps the most distinguished opponent of AI is Hubert Dreyfus, who gained initial prominence during this era of broken promises and unfulfilled predictions. Author of such essays as "What Computers Can't Do", Dreyfus appears to take great delight in exposing alleged AI fraud and poking fun at the science in general. While Dreyfus evidently enjoys subjecting AI to ridicule, and undoubtedly has gloated over the less than meteoric rise in the science, it will be interesting to see who laughs last.

Even though there was more style than substance to early efforts in AI, research in this "black art" led to some technological spinoffs that are used in the computers available today, dumb or otherwise. AI programmers were responsible for the first timesharing systems, which they developed in 1960 or so on a DEC PDP-1 for the admittedly self-serving purpose of obtaining computer time, a precious commodity in those days before VAX. Computer graphics also had its origin in AI research: AI programmers wanted the ability to create pictorial representations of their abstract concepts, so they invented the first crude graphics packages. Later, screens with pull-down menus and other features used today on several popular microcomputers were developed by AI researchers.

Other AI developments took place during this timeframe as well. John McCarthy's 1957-vintage LISP programming language was improved and cloned in a variety of dialects. The efforts of Simon and Newell to produce a general problem solver program were not entirely unsuccessful. And ongoing efforts by numerous

researchers led to programs that were capable of playing checkers and chess by mimicking the strategies that people use to play these games. While these pioneers failed in their efforts to produce a truly intelligent computer, they left an indelible imprint on computer technology.

### AI Defined

As of yet, no concrete definition of machine or artificial intelligence is generally accepted. There are probably as many definitions of artificial intelligence as there are AI researchers, including the cynical exclusionary definition, "If it works, then it isn't AI." And it's certain that AI gets redefined each time a new book on the topic graces the shelves of your local bookstore. Considering that we are still unable to precisely define human intelligence, being unable to get a handle on AI should come as no surprise.

One of the better AI definitions is attributable to Nils J. Nilsson, a prominent researcher at SRI International in Menlo Park, California. According to Nilsson:

*The field of Artificial Intelligence has as its main tenet that there are indeed common processes that underlie thinking and perceiving, and furthermore that these processes can be understood and studied scientifically. . . . In addition, it is completely unimportant to the theory of Artificial Intelligence WHO is doing the thinking or perceiving—man or computer. This is an implementation detail.*

This is an interesting, thought-provoking definition that explains some of the philosophy behind AI. However, I find my own definition to be equally useful and a bit more digestible:

*Artificial Intelligence is the science of creating machines that emulate the human mind.*

### Can A Machine Think?

Machine intelligence is hard to quantify. Just when does a machine become "intelligent"? Back in the 1950's, a mathematician named Alan Turing attempted just such a quantification by developing the Turing Test. Turing postulated that a computer could be deemed intelligent if it was capable of engaging in a logical, coherent two-way conversation (albeit on a terminal) with a human, providing the human was unaware that he or she was conversing with a machine. Although the Turing Test is an interesting concept that has prompted numerous philosophical debates, it is not a valid benchmark for determining machine intelligence.

Many programs that pass the Turing Test are available today. The most publicized example is ELIZA, a program that simulates a Rogerian client-centered psychotherapist. This "silicon psychiatrist" was developed in the late 1960's by Joseph Weizenbaum. ELIZA was appropriately named after Pygmalion's Eliza Doolittle. Like the character in the play, ELIZA was taught to speak what seemed to be fluent English. Though this program passes the Turing Test, it is by no means intelligent. ELIZA works by elementary pattern matching—it picks up keywords and parrots stock answers to them. In fact, one of Weizenbaum's motives in writing ELIZA was to prove that true natural language understanding by a machine was out of the question. No doubt Weizenbaum was disgusted when several psychologists proposed using ELIZA as a screening mechanism for their own clients.

The ELIZA program reinforces the fact that a computer does not and can not "think" in the same fashion that you and I do. Computers are incapable of original thinking or emotion, and they certainly don't possess human nature - there is no soul or sentience in silicon. For the time being at any rate, we don't have to worry about the possibility of a silicon-based intelligence competing with our carbon-based minds. The concept of computers, intelligent or otherwise, gaining control of mankind went out of fashion long ago. None of the recent developments in AI lend

credence to the robotic revolt in Karel Capek's play, "R.U.R" or to science fiction novels of the "computer takes over world" genre.

However, computer programs that draw on the collective wisdom of numerous talented individuals are available and in use now. Because such programs concentrate the expertise of many talented people on the solution of a single problem, they can be considered to be "smarter than a person." Appropriately enough, these programs are called "expert systems". At the present time, expert systems represent the most practical and profitable real world application of AI technology, and therefore warrant a closer look.

## THE BLOSSOMING OF EXPERT SYSTEMS

While journalists herald Japan's endeavors to give birth to the so-called Fifth Generation of intelligent computers, we are putting artificial intelligence to work right now in real-world applications. Today, there are a growing number of "expert systems"—intelligent assistants—working in concert with human experts on such diverse tasks as VAX system tuning and configuration, geological exploration, medical diagnosis and mass spectrographic analysis. These expert systems bear names like XCON, DENDRAL, PROSPECTOR and CADUCEUS. Despite their unusual names, these programs should not be taken lightly: They perform at the level of human experts, drawing upon pools of knowledge painstakingly extracted from numerous VAX system specialists, chemists, geologists and doctors.

The development of expert systems is a result of some of the lessons learned by researchers during the period that AI was regarded as a hoax by mainstream computer scientists. Among the discoveries made in the AI and cognitive psychology labs were the facts that not only does intelligence require knowledge, but that the acquisition and application of knowledge form the basis of intelligent behavior.

This led to the realization that a general purpose problem solving program would have to contain vast amounts of knowledge about an incredible variety of topics. Because no existing computer could store, much less process, this volume of information, the approach to artificial intelligence shifted from a quest for a generic problem solver to efforts to develop specialized programs capable of performing intelligently in specific domains. These programs were the precursors of today's expert systems.

## The Anatomy Of An Expert System

Also referred to as a knowledge based system, an expert system is a computer program, usually written in an esoteric language like LISP, OPS or PROLOG, that can perform at the level of a human expert by mimicking the activities that a human expert would undertake in the resolution of a problem. Expert systems archive human knowledge and permit a collective pool of wisdom and expertise to be applied to problems for which there are no clear-cut yes-or-no answers.

Expert systems have two main components—a knowledge base and an inference engine. The knowledge base is the pool of information that the expert system can draw from. The inference engine contains the heuristics, or rules of thumb, that help determine which pieces of information the expert system will draw from the knowledge base to resolve a problem. These components are accessed through an I/O subsystem called a "front end" or "user interface." The I/O subsystem generally has a natural language interpreter, allowing you to communicate with the expert system on a conversational basis. This imparts an aura of intelligence or friendliness to the expert system, but it isn't where the intellect resides.

What makes an expert system possible, and makes it emulate human thought processes, is the fact that knowledge can be ex-

pressed as rules. Although you may have never stopped to think about it, we store knowledge as rules. Consider the weather—IF it is raining, THEN put on a raincoat before you go out. An expert system works the same way, using its inference engine to evaluate the information that is stored as IF-THEN constructs in its knowledge base.

## The Knowledge Base: Codified Expertise

The knowledge base component of an expert system may be likened to a data base that reflects the thoughts and procedures of the best people in a given field. Because of the prodigious storage capacity of modern computers, a given expert system can embrace the decision making methods of hundreds of human experts. This IF-THEN codification of expertise is analogous to the experience base that we humans accumulate and store as memories. Like memories, the elements or rules in a knowledge base must be defined or "experienced," then recorded and cataloged. It is the job of computer scientists known as knowledge engineers to coax this kind of information from human experts, code the information into sophisticated expert systems, and put the programs to work performing tasks that normally require human expertise.

## Building The Knowledge Base

Extracting human knowledge is not a simple undertaking—in many cases, we perform tasks intuitively or heuristically and can't easily explain to someone else how we arrive at decisions. You can prove this to yourself by attempting to describe your job and the exact manner in which you carry it out in such detail that your description could be converted into a computer program.

In addition to being adept at extracting knowledge from people, a knowledge engineer must also know what specific kernels of information are germane to the expert system that he or she is building. This isn't as easy as it might seem, because almost everything we humans do involves a wealth of background information and presupposed knowledge. For instance, if you wanted to build an expert system that could cook an omlette, you would first have to teach it how to break eggs. Not only is breaking an egg something that we take for granted, it isn't the sort of activity that's conducive to being expressed as a series of rules.

This example underscores a statement by Arnold Kraft, an artificial intelligence specialist with Digital Equipment Corporation. According to Kraft, "one of the main obstacles to the implementation of expert systems is putting knowledge into a box." The egg-breaking problem is but a simplistic example of the challenge faced by knowledge engineers.

## Educated Guesswork

Expert systems mimic humans in another regard: when confronted with a problem that contains ambiguities, an expert system program uses heuristics, or rules of thumb, to attempt to resolve the problem. We use this same form of judgment based on experience to resolve problems for which we don't have stock, predefined answers—only we refer to the process as making an "educated guess." The outcome of heuristics is the same for both man and machine: Such guesswork usually achieves the desired results, but does not guarantee them.

Heuristics are applied to a knowledge base through the other main component of an expert system, the inference engine. It is the inference engine that's responsible for the actual problem solving that takes place in an expert system. Using heuristics, it determines which rules or IF-THEN constructs will be evaluated during problem resolution. Inference engines utilize one or both of two general strategies to resolve a problem. These strategies, or lines of reasoning, are called forward and backward chaining.

Forward chaining works on the principle of “given these facts, what will happen?” Determining the outcome of a football game based on the score late in the fourth quarter is a good example of forward chaining, or working from facts to conclusions. A forward chaining inference engine rummages through the knowledge base of the expert system, testing all of the available rules again and again and adding new facts until no rule applies. In the case of the football game, an inference engine would likely conclude—with a high degree of probability—that the team with the highest score in the fourth quarter would ultimately win the game.

Backward chaining relies on an opposite principle—“given this fact, what events led up to it?” In this strategy, the inference engine is confronted with an unsubstantiated theory which it must attempt to prove. This strategy is based on finding the rules within the knowledge base which support the theory, and then verifying the facts which enable the rules to work.

CADUCEUS, a medical expert system, uses a backward chaining inference engine to diagnose illnesses. When CADUCEUS is supplied with a set of symptoms, it will backchain and associate these symptoms with what it knows about diseases. Ultimately, the program will deliver its diagnosis of the disease in question based on the symptoms it was supplied with. Thus, a backward chaining inference strategy is diagnostic, while a forward chaining strategy is predictive.

### Heuristic Search Limitation

The need for an inference engine or a specific problem solving strategy is based on a phenomena known as the “combinatorial explosion.” This combinatorial explosion must be contained by a search strategy that limits the expert system’s evaluation of rules to those rules which are appropriate for a given problem or situation.

For example, there are expert systems that play championship chess. In order to do so, these expert systems must limit their search for moves—there are more possible moves in a chess game than there are atoms in the universe. An example of a heuristic used in chess by human chess masters and computer programs alike is “try to control the center of the board.” Through the use of a forward chaining inference engine and heuristic search, an expert chess system is able to limit its search to a small portion of its knowledge base—those moves that would be considered feasible at any given point of a chess game.

You can better understand the significance of a heuristic search strategy if you consider the way people use heuristics to make decisions every day. Life involves dealing with problems on a constant basis. Few of these problems are insoluble, and most are dealt with on a subconscious level. However, all of these problems would be insurmountable without heuristic search. A brute-force approach to problem resolution would require you to examine all of the information in your mind each time you had to make even the most trivial decision. Considering the vast amount of data stored in the human brain, one such brute-force solution would take an eternity. Fortunately, through rules of thumb or heuristics, we can limit our searches and arrive at optimal answers almost instantaneously.

### Supply And Demand

One of most significant problems with AI in general and expert systems in particular is based on the law of supply and demand. The “Dark Ages” of AI caused a rapid decline of interest in the field, and a consequently diminished AI research and development effort. Expert systems have proven that AI works and have brought the discipline back out of the closet and the cognitive psychology labs. Now that AI has regained a mantle of respectability, just about everyone wants it. However, knowledge

engineers and people with doctorates in AI are scarce—and expensive—commodities.

The expert system toolkit has been touted as a short term solution to the paucity of experienced knowledge engineers. Toolkits are software packages which allow average programmers to develop limited expert systems without the assistance of AI specialists. These application development tools are available for a broad spectrum of computer hardware, from microcomputer to mainframe. They are sometimes referred to as expert system “shells”, for they are essentially inference engines supplied with a knowledge base framework that must be filled with data or rules by the end user. Shells or toolkits cannot replace knowledge engineers, but they do increase the availability of expert systems and allow programmers without LISP, OPS or PROLOG experience to use AI techniques.

The ES/P Advisor is a representative expert system shell. Developed by Expert Systems International, a firm that specializes in business applications of AI, this tool runs on a number of popular microcomputers and is used primarily for “text animation”. In this scheme, a consultation shell is used to access and interact with a user-definable knowledge base, which is typically a complex document. The Advisor is presently used for such applications as interpreting tax regulations and guiding clerical personnel through activities like issuing mortgages. Both of these applications normally require a deep familiarity with complex procedures which have their basis in legal and financial documents.

General Research markets TIMM, The Intelligent Machine Model, an expert system generator that has been used to build an intelligent VAX tuner. While TIMM/Tuner it can’t equal the performance of a VMS wizard equipped with VAX SPM and plenty of time, it does a remarkably good job adjusting SYSGEN parameters and offering advice—including the familiar DEC sales litany, “buy more memory.”

A number of other companies offer expert system toolkits, and more are presently under development for micro, mini and mainframe implementations. Representatives from DEC’s AI technology center in Hudson won’t say much, but it’s likely that DEC will offer an toolkit of its own in the near future.

### To Err Isn’t Just Human

Even though AI owes most of the credit for its resurgence to the increasing viability of expert systems, these programs are far from perfect. Today’s expert systems function only in very narrowly defined problem contexts, or domains. Despite the inroads being made in the microcomputer arena, expert systems are typically large, memory-intensive programs that work best on a mainframe or supercomputer.

Creating a program that has the ability to solve problems and combining this program with a knowledge base of accumulated experience and data is an important achievement, but expert systems are far from perfect. An expert system lacks the human quality of spontaneous insight, and it can not acquire or create new knowledge. You can’t get more information out of an expert system than you put in in the first place. Finally, expert systems can and do make mistakes—just as human experts do.

### NATURAL LANGUAGE UNDERSTANDING

Natural language understanding remains a thorny problem in the domain of expert systems and AI. While great strides have been made in voice recognition technology, true natural language understanding remains an elusive goal. Many attempts have been made towards natural language understanding with less than satisfactory results. In fact, some of the earliest efforts in AI involved automatic language translation programs. These efforts were doomed to failure because computers couldn’t apply the correct

meaning of an ambiguous word within a given context. That is to say, when confronted with a common nursery rhyme, the computer couldn't tell whether Mary gave birth to, owned, had sex with, or ate the little lamb.

The stumbling blocks to natural language understanding are related to the way we humans use and process language as opposed to the way that computers communicate. All computer languages ultimately boil down to binary machine code, while we communicate with each other symbolically. This creates a significant communication gap between people and computers.

Computers can be programmed to understand or "parse" sentences by analyzing nouns and verbs, but they have a difficult time with adjectives, metaphors and commonly used expressions. The development of parsers capable of dealing with slang and context will not be an easy undertaking. Consider the phrase "Bank Failure Rocks Wall Street." A computer would likely interpret this phrase as being indicative of an earthquake in Manhattan. Or the often recounted English to Russian translation of "the spirit is willing but the flesh is weak." An early automatic language translation program interpreted this phrase as "the vodka is good but the meat is rotten."

We also have a unique ability to filter noise. How many times have you been at a party with several dozen other people, all of whom are conducting conversations simultaneously, when you suddenly hear someone mention your name? In this situation, you can focus your attention on that specific conversation by filtering out the "clutter" of the other, concurrent, conversations.

Another thing that we can do better than computers is fill in the blanks. Many of our day-to-day conversations, particularly with close friends, involve phrases and sentence fragments that would appear cryptic and incomplete on paper, yet are perfectly comprehensible when spoken. And we can often understand what someone means to say even if we are listening to the other party over a poor telephone connection. Finally, we are capable of recognizing and processing continuous speech in real time—we can instantaneously understand sentences we have never heard before. As of now, no computer shares any of these capabilities with us.

Progress is being made in the area of natural language understanding, as evidenced by the availability of natural language query programs which interface with database management systems. INTELLECT, EASYTALK and THEMIS are well known VAX implementations, and there's even a natural language program called Clout that runs on microcomputers. However, these programs can perform only elementary parsing on a limited vocabulary. Much work still needs to be done before true natural language understanding, the gateway to more efficient man-machine interfaces, becomes an affordable commercial technique.

## **A MACHINE OF VISION**

Significant advances in the the area of computer vision are critical to the success of a number of AI application areas, including robotics, CAD/CAM, autonomous vehicles, sensors, and knowledge acquisition systems. At present, it can be stated that that computer vision technology has a long way to go before machines can see with any degree of clarity. A viable machine vision system awaits developments in pattern recognition, edge detection, and scene integration technology. These developments will be contingent on the availability of advanced supercomputers, for an incredible amount of processing power and speed is needed to implement a vision system. Our own vision system requires a significant percentage of our brain power, and when you consider that the human mind has been described as being more powerful than a network made up of all of the computers in the world, it's clear that a reasonable emulation of human sight remains a distant goal.

Efforts at Stanford University have produced a model vision system called ACRONYM. This system has proven adept at aircraft identification through the analysis of aerial photographs of airport runways. The ACRONYM system contains a knowledge base of representative objects which it compares with photographic images. An edge mapper extracts lines and curves from the photograph while a stereo mapper obtains information on surfaces. This information is integrated and interpreted by searching and matching subsystems. Although intended to be a general purpose vision system, ACRONYM does a less than satisfactory job of feature extraction, and is incapable of analyzing scenes which contain numerous discrete objects.

The magnitude of the machine vision task is underscored by another Stanford project, the autonomous "golf cart" designed by researcher Hans Moravec. This vehicle is capable of moving at three to five meters per hour in one-meter increments through the use of an on-board vision processing system. Each movement is the result of a single image analysis conducted by a one MIP computer. To speed up Moravec's cart to even a slow walking pace of one meter per second would require a processing speed of one to 10 billion instructions per second. These speeds are unattainable today, but the next generation of computer hardware will make projects far more ambitious than Moravec's vehicle technically feasible.

## **THE FIFTH GENERATION**

The first four generations of computer technology have lasted roughly ten years apiece. The passing of each generation has been marked by technological refinements to an existing concept. Computers have gotten smaller, cheaper and faster at a blinding pace—the word processor or personal computer we take for granted today would have been inconceivable little more than ten years ago. Although the representatives of the fourth generation are no longer icons of power and prestige, they remain a variation on a familiar theme.

The advent of the next generation of computing will involve more than technological change. It will redefine the concept of computing and of computers themselves, for the machines of the Fifth Generation will be knowledge information processors—computers that will emulate human thought and cognition. Three major criteria will distinguish the computers of the Fifth Generation from the AI hardware and software in use today—Computer architecture, processing speed and the ability to manipulate symbolic rather than numeric information.

### **Computer Architecture**

Computers of the fourth generation, like the three preceding hardware evolutions, are characterized by Von Neumann architecture. This architectural concept has remained essentially unchanged since the introduction of the first digital computer. Machines of this design are restricted by their inability to do more than one thing at a time—data and instructions must be passed through a single link or bus to the central memory. This so-called Von Neumann bottleneck is the limiting factor in computer performance today.

Attempts to bypass the Von Neumann bottleneck through the use of networking have been unsuccessful. While numerous processors can be linked together in a network, a single program cannot be effectively broken down into subroutines which are assigned to individual nodes. Each subroutine call, return or addressing operation would of necessity require data transfer between nodes. This approach to processing would be inefficient at best—more time would be spent on internode communication than actual information processing.



## Pipelining

One technique employed by hardware designers to speed data through the Von Neumann bottleneck is the pipeline. This method is analogous to a factory assembly line in that a steady stream of data is fed into a string of processor segments, each of which performs a specialized function. Each segment executes one portion of an instruction and then passes the instruction along to the next segment in the processor. While one segment is executing an instruction, another can fetch a second instruction, a third segment can obtain an operand address, and yet another segment can be decoding a fourth instruction. Through the use of pipelined processing, a steady stream of data can be input to the computer as if by conveyor belt, keeping each processor segment in almost constant use. Not only is this technique used in supercomputers like the processors available from Cray and Hitachi, it's responsible for a good deal of the increased speed of DEC's VAX 8600.

While pipelining yields impressive gains in processor performance, a computer employing this architecture is still a serial Von Neumann machine. As such, it's still subject to the constraints of the processor-memory link. At this point in time, we are nearing the theoretical limits of Von Neumann processor performance. One short term solution is the gallium arsenide VLSI chip, which is significantly faster than today's silicon chips. However, the potential speed advantage of a gallium arsenide chip might be in the vicinity of ten to one, certainly not one or more orders of magnitude. The Josephson junction and superconductivity are also hailed as multipliers of processing power, but the most we can reasonably expect to gain from this technology is a geometric increase in processing speed. To obtain the performance necessary for Fifth Generation computers, we will have to develop and implement radically new machine architectures that yield exponential increases in computing power and speed.

## Towards A New Architecture

The most promising answer to the Von Neumann bottleneck is embodied in dataflow and parallel processing technology. A dataflow processor has multiple CPUs that are tightly coupled and connected by circular pipelines through which data is transferred at a high rate of speed. Where a network might allocate individual programs or tasks to different nodes, a dataflow processor assigns packets of machine language instructions that make up a given program to individual CPUs.

Parallel processors take this concept a step further. The CPUs of a dataflow processor are connected serially by a simple ring topology. The CPUs of a parallel processor will not be constrained by this simplistic networking scheme. Each CPU will be able to communicate with a group of CPUs instead of just the processors adjacent to it. Once a program is broken down into its smallest discrete elements, this information will flow through the parallel processor much in the fashion that water flows when poured over the roots of a plant.

Each data element will be free to seek an available processor by following the path of least resistance. If a specific processor is already "busy," the data packet will seek out the nearest available alternative CPU rather than wait for the attention of the processor that's already occupied. This arrangement will permit each CPU chip in a parallel processor to devote itself to a specific aspect of a problem and work in concert, rather than in contention, with all of the other chips. Conceptually, this is very similar to the way work is done by colonies of social insects, the denizens of a beehive or anthill, for instance. It is also similar to the way that we humans think.

One of the obstacles to achieving processing simultaneity in a parallel processing environment is our present inability to efficiently decompose a program into the discrete segments which will be assigned to the individual CPUs in a parallel processor. Con-

tinuing research and development efforts in the area of relational databases should resolve this problem. Once this database software is perfected, it should be possible to engineer and write a parallel processing operating system that will regard a program as a relational database. The operating system would decompose the program into segments which would in turn be addressed and manipulated as discrete members of the database.

True "Non-Von" dataflow and parallel processors are largely experimental, but the concept of leveraging multiple CPUs against a single problem is viable today. The incredible speed of the \$17.6 million dollar Cray-2 supercomputer is attributable to its four-processor architecture, and Cray Research is already developing a still faster and more powerful supercomputer, the 16-processor Cray-3. And next-generation prototypes abound: The Lawrence Livermore Laboratory has developed a 16-processor non-Von Neumann computer and engineers in Manchester, England are designing a dataflow computer with 256 processors. Columbia University and MIT are developing parallel processors with as many as one million individual CPU chips, and the Japanese have voiced similar goals for their Fifth Generation project.

## Processing Speed

Processing speed will make a quantum leap in Fifth Generation computers. These machines will manipulate data at such blistering speeds that the supercomputers of today, like the Cray X-MP and the CDC CYBER, will be snail-like in comparison. Fourth Generation supercomputers typically have processing speeds on the order of 100 MIPS, or millions of instructions per second. It is expected that the processing speed of Fifth Generation machines will be measured in BIPS—billions of instructions per second. In fact, Hitachi's latest supercomputer reportedly has attained speeds in excess of one gigaflop, or one billion floating-point instructions per second.

## Symbolic Data Manipulation

Machine instructions per seconds will be replaced with a new unit measure of speed—LIPS, or logical inferences per second. Crucial to the LIPS concept is the manipulation of symbols instead of numbers. Languages like LISP and OPS are designed to manipulate symbols, often represented within IF-THEN constructs, instead of raw arithmetic data. Each execution of an IF-THEN construct or production represents one logical inference. Because both the IF (left hand side) and the THEN (right hand side) of a production can contain many elements, a logical inference requires more machine power to execute than a machine language statement like ADD A TO B. Each logical inference requires anywhere from 100 to 1000 traditional machine instructions. Therefore, processing speed will have to make a quantum leap before we see a computer benchmarked as a one million LIPS machine. This is but one of the objectives of the architects of the Fifth Generation both here and abroad.

## FIFTH GENERATION PROJECTS

The race to create superspeed computers and processors which exhibit the humanlike qualities of inference and decision-making has come to be called the Fifth Generation. Efforts are being made throughout the world to design and implement the hardware, software and technology of a new class of machinery that will represent the Knowledge Age. There is no doubt that a de facto Fifth Generation standard will emerge at some point in the near future. The question that still remains unanswered is who will develop this standard and wield its inherent powers. Although we invented and developed the digital computer and artificial intelligence technology, a very determined Japan is poised

to become Number One in the knowledge business. The actions we take or fail to take in the next few years are likely to be crucial to the outcome of the race for computer supremacy.

### The Japanese Effort

In April of 1982, the Japanese Ministry of International Trade and Industry launched a 10-year crash program called Fifth Generation Computer Systems. Implementation of the FGCS plan began with the establishment of Institute for New Computer Generation Technology (ICOT). ICOT is a collaborative effort involving eight of Japan's major electronics manufacturers which encompasses some 24 individual projects. These projects have as their common goal the development of an intelligent Fifth Generation computer with a natural language interface.

The seeds of the FGCS program were planted in the late seventies when Japan's Ministry of International Trade and Industry (MITI) organized a research team headed by computer scientist Kazuhiro Fuchi to draw up a master plan for the development of an entirely new class of data processing machinery. The result of this effort was a preliminary plan to design, engineer and build an ultrapowerful data processing machine that could reason like a human being. It was only fitting that a computer of such revolutionary design be given a name to distinguish it from its lesser cousins, so the Japanese decided to call their new machines "knowledge information processors," or KIPs.

The design specifications for this new class of computer are impressive. By the early 1990s, Fuchi's team expects to have an operational KIP with the ability to process a 1,000-gigabyte knowledge base at the speed of one billion LIPS. The user interface will consist of a natural language front end with a 10,000 word vocabulary, and a continuous speech recognition system that can handle 50,000 words with 95 percent accuracy. Given the complexity of the Japanese alphabet in comparison to our English alphabet, developing the user interface alone will be an awesome challenge. In addition to meeting the design criteria of speed and "friendliness," the KIP will have to be adept at image analysis and processing, logical inference and independent knowledge acquisition.

The Japanese intend to develop their KIPs by concentrating on three research areas: enhancing relational databases to better support knowledge based systems; personal sequential inference (PSI) machines; and parallel inference systems. The PSI is crucial to the two other research areas, for it is to be used as a development tool for advanced knowledge based systems and massively parallel hardware and software. In its first iteration, the PSI will be a PROLOG AI workstation similar to our LISP machines. Designed to draw inferences from knowledge bases at 20,000 to 30,000 LIPS, the PSI will be enhanced to reach far greater speeds as it plays the role of stepping stone in the quest for more powerful AI engines, including parallel relational database machines. The first PSI implementation should be forthcoming shortly, for it is based on technology available right now (the soon to be announced VAX PC and an experimental 60,000 LIPS PROLOG board would provide a satisfactory architectural environment for such a machine.)

Japan is taking a dual-track approach to the Fifth Generation. While FGCS labors to develop a knowledge information processor, a lesser known research and development program to enhance "traditional" computing is being conducted. Called the National Superspeed Computer Project, the program is aimed at producing an ultrafast processor—a computer 1000 times faster than a Cray—by 1988. Like the FGCS program, the NSCP involves multiple vendors and government seed money. Japan's six largest computer manufacturers are participating in the project, and their initial efforts have been subsidized with 100 million dollars in startup money. Spurring these efforts are the supercomputers recently

unveiled by two NSCP participants. While the Cray-like performance of the new processors from Fujitsu and Hitachi is far short of the NSCP goal, these machines represent the first serious Japanese foray into the commercial supercomputing marketplace.

Japan has established lofty goals for its FGCS program, and it's unlikely that they will be fully realized by the early 1990s. However, a careful analysis of the ICOT program and its objectives does point out some significant, if not disquieting, facts. For the first time, Japan is intent on creating a new technology, not merely improving on existing concepts and methods. The researchers at ICOT and elsewhere plan to leapfrog the current state of the art in hardware and software by adopting a strategy of invention instead of imitation. As Feigenbaum and McCorduck point out in *The Fifth Generation*, Japan no longer intends to be a copycat nation. Whether or not the vaunted Fifth Generation program turns out to be a complete success, it doesn't pay to underestimate the Japanese: Twenty years ago, the word "Honda" was synonymous with "50-CC motorbike."

### The American Response

The United States has not been oblivious to Japan's attempts to forge the Fifth Generation. Having seen the Japanese surpass us in many aspects of computer hardware production, we are not about to sit idly by while they achieve dominance in the software industry as well. Our response includes an array of intensive Fifth Generation research and development projects under the aegis of the Federal government and the private sector.

### The MCC

One element of our response is the Microelectronics and Computer Technology Corporation or MCC. Based in Austin, Texas, the MCC is a nonprofit cooperative joint venture that was forged between a dozen major U.S. computer and electronics firms. At least count, the number of participating firms has almost doubled. The MCC is headed by retired admiral Bobby Inman, former director of the supercomputer-intensive National Security Agency and deputy director of the CIA. The consortium is staffed by computer scientists, engineers and researchers "loaned" to the corporation by their parent firms. The list of MCC participants reads like a "Who's Who" of American computer and electronics manufacturers. CDC, NCR, Motorola, 3M, RCA and Sperry are involved, as is Digital Equipment Corporation.

Two noticeably absent companies are Cray Research and IBM. Cray is apparently an independent-spirited company, and IBM is evidently concerned over the potential of antitrust litigation. However, no antitrust action has been instigated or threatened by the Federal government. In December 1983, a Justice Department ruling removed antitrust barriers to the MCC, indicating that it has adopted a "wait and see" attitude about the consortium's unique collective approach to research and development.

The MCC participants are united by the common desire to produce a state of the art Fifth Generation knowledge information processor, an undertaking that would prohibitively expensive if attempted on a solo basis. To achieve its goals, the MCC is concentrating on research in four general areas—Microelectronics packaging, advanced software technology, CAD/CAM and computer architecture. Seed money for the MCC was generated by its participants—each member joined the project by purchasing a \$500,000 share of stock. Ongoing research in each of the four major MCC programs is subsidized by each firm involved in that application area.

## The SRC

Another acronym in the growing list of American Fifth Generation ventures is the SRC, or Semiconductor Research Corporation. The SRC is a cooperative effort launched by some 30 domestic semiconductor manufacturers and consumers as a hedge against foreign competition. Among the participants in this venture are Burroughs, Control Data, DEC, Intel and IBM. Like the MCC, this cooperative effort is funded by its participants. Contributions are based on each firm's integrated circuit revenues, and range from a minimum of \$60,000 to a maximum of 14 percent of the SRC budget. Instead of conducting its own R&D programs like the MCC, the SRC funds research efforts at colleges and universities, awarding grants much in the same fashion that the federal government does.

## Independent Efforts

While deeply involved in the collective efforts of the MCC, many of America's major computer manufacturers have ongoing proprietary AI research and development programs. Texas Instruments has an extensive internal program, as evidenced by its Explorer AI workstation. AT&T has implemented projects in several AI application areas, and numerous other firms are conducting vigorous programs as well. Of particular note are the efforts of our two largest computer manufacturers, IBM and Digital Equipment Corporation.

## IBM And AI

The number one mainframe manufacturer has maintained a relatively low profile with respect to AI. As an industry leader, what IBM fails to do is almost as important as what it does. In fact, some experts say that government involvement in AI and new machine architectures was spurred by IBM's apparent reluctance to make such a commitment on a corporate level. The firm's only announced commercial AI product to date is an expert system called Epistle which reads your electronic mail and extracts and summarizes the salient points of each message. An expert system shell called PRISM is being readied for commercial introduction at IBM's Silicon Valley research center. Other products, including a continuous speech recognition system, are being developed at IBM's Yorktown Heights research facility, but Big Blue is not publicizing them.

Historically, IBM had an early start in AI. In the early Fifties, certain of its employees began writing programs that played chess and checkers. Ever concerned with its corporate image, Big Blue was aghast at the idea of machine thought and its Forbin Project connotations, preferring to promote the computer as nothing more than a dumb brute that would dutifully carry out repetitive instructions. So much for gameplaying in Armonk.

Although IBM portrayed its hardware as dumb but fast numbercrunchers, its second-generation 36-bit mainframes, the 709 and 7090, were AI mainstays in the early 1960's. However, 1964 ushered in the era of the System/360 family. These third generation processors featured 32-bit addressing, which didn't sit well with AI researchers. Twenty years ago, AI people were used to 36-bit architecture, which allowed them to store two 18-bit addresses in a single word. This characteristic wasn't of any particular value to the business community, so IBM replaced it with the "less is more" System/360 architecture.

## Digital Equipment Corporation

At the request of MIT's Artificial Intelligence laboratory, DEC came to the rescue with the PDP-6, a 36-bit "LISP engine". One of the design goals inherent in the PDP-6 was fast, efficient execution of LISP programs, and DEC succeeded admirably in this regard. Shortly thereafter, the firm expanded its niche in AI research

and development with the introduction of the DECsystem-10, which is still a popular AI system two decades later. And today, the VAX is the general-purpose AI machine of choice.

DEC's internal involvement with AI was accelerated some six years ago when Dennis O'Connor, a DEC group manager, happened to meet Professor John McDermott, an AI researcher at Carnegie-Mellon University. O'Connor was looking for a method to increase the speed and accuracy of VAX system configuration, a task that was rapidly getting out of hand. McDermott, coauthor of the LISP-based OPS production system language, was searching for a problem that an expert system written in OPS could solve to prove the viability of commercial expert systems. This chance meeting resulted in a collaborative effort that culminated in the development of RI, the VAX computer system configurator now known as XCON. XCON proved to be so cost-effective that DEC decided to become more deeply involved with AI, intending to use the new technology to resolve other business problems previously considered intractable by computers.

The firm was in an ideal position to sponsor a large scale internal AI program because it already had the necessary hardware and human resources at its disposal. Because it was obvious that AI applications could be designed and implemented most efficiently if the effort was conducted from a centralized location, the AI Technology Center was conceived and built. Since its opening in January 1983, the Hudson facility has been the focal point for DEC's AI research, development, and marketing.

The Hudson facility is home to three teams of employees, each devoted to a different aspect of AI. The AI Marketing Group is responsible for the promotion and distribution of DEC's commercial AI products. The AI Engineering Group develops and implements new AI applications for internal use as well as commercial distribution. Finally, the Intelligent Systems Technology Group devotes its efforts to manufacturing-oriented knowledge based applications such as internal production management and material requirement planning.

DEC also makes substantial contributions to academic AI efforts through its twenty-year-old External Research Program. In this cooperative program, DEC selects R&D projects that are of particular interest and then farms them out to academic institutions rather than pursuing them in-house. To make the program attractive to universities, DEC supplies at least half of the seed money and expertise required to get each project off the ground. Again, DEC remains somewhat coy about the depth and breadth of this program, but it's estimated that the company is currently funding at least 100 external research ventures.

## GOVERNMENT EFFORTS—AN INTELLIGENT DEFENSE

Not surprisingly, the Department of Defense is involved in AI, expert systems and supercomputers in a very big way. Our social structure and government philosophy does not permit us to match the Soviet Union weapon for weapon or megaton for megaton. Accordingly, we are striving to maintain qualitative rather than quantitative superiority over the Warsaw Pact. Implicit in our defense strategy is the maintenance of a technological edge over the Soviet Union through "smart" weapons and technical intelligence resources complemented by sophisticated computer hardware, software and architecture.

Back in the 1950's, it was the Pentagon that lobbied for intelligent computers that could translate Russian into English. Later, the DOD became interested in natural language recognition and voice and signal processing techniques. These interests dovetailed with an equal concern for electronic intelligence interception and exploitation, pursuits requiring significant computational horsepower. This concern has been reflected by the creation of several government agencies, most notably the National Security Agency and the Defense Advanced Research Projects Agency.

## The NSA

In terms of sheer numbercrunching, our most powerful Federal organization is the National Security Agency. One of the nation's most obscure intelligence agencies, the NSA has an awesome array of supercomputers at its disposal, and has been a center of important breakthroughs in computer technology since its secrecy-shrouded creation in 1952. The NSA's main computer room, located beneath its Maryland headquarters, occupies over twelve acres of floor space. It houses processors from DEC, IBM, and Cray Research in addition to numerous custom-built special purpose machines. NSA, which is often referred to as an acronym for "Never Say Anything", is mum about its hardware, software, and related paraphernalia, but it's a sure bet that these resources aren't devoted to checkers tournaments.

NSA has fractionally lifted its veil of self-imposed anonymity by announcing its role as custodian of a new venture, the Supercomputing Research Center. This facility, located at the Maryland Science and Technology Center, is the brainchild of a federal think tank called the Institute For Defense Analysis. It is expected to draw at least 100 eminent computer scientists, AI researchers and engineers. What is particularly noteworthy about this endeavor is the fact that it represents NSA's first significant involvement in a semi-public-domain research and development project.

## DARPA

More commonly known is the Defense Advanced Research Project Agency (DARPA), the agency that brought you ARPANET and was instrumental in the development of the ADA programming language and the world's first supercomputer, the ILLIAC IV. DARPA's precursor, the Advance Research Projects Agency, was formed in 1958 as part of our response to Soviet technological efforts which culminated in the successful launch of Sputnik. Four years later, the Information Processing Techniques Office was formed to advance the development of interactive computing, timesharing and networking for command and control functions. During its tenure, DARPA has infused over half a billion dollars (its current annual budget) into computer research. It is arguable that DARPA-funded research has been critical to almost every aspect of AI, including supercomputers, parallel processing and natural language understanding. Many experts contend that it was DARPA money which kept AI research afloat during the years of disillusionment and disrespect, and it's certain that the agency will continue to play a pivotal role in AI-related endeavors.

DARPA is the parent agency for a program called Strategic Computing and Survivability, also known as the Strategic Computing Initiative. The SCI has as one of its primary goals the development of a new class of superintelligent machines which can be industrially produced. In its efforts to bring forth an AI "Model T," the SCI is funding research in expert systems development, microelectronic design, machine architecture, natural language processing and speech recognition, and machine vision. Three application prototypes have already been selected and funded as SCI projects. These are an autonomous land vehicle, an intelligent pilot's associate and a battle management system.

The autonomous land vehicle, presently under development by Martin Marietta, is to be a vehicle capable of steering, navigating, avoiding obstacles and otherwise driving itself from Point A to Point B at speeds up to 60 kilometers per hour. All this is to be accomplished—without human intervention—by expert systems. This is easier said than done. The navigational portion of the expert system will require at least 6000 rules which must fire at a rate of 7000 rules per second. Today's expert systems are considerably smaller and much slower—rarely are firing rates in excess of 100 rules per second.

The vision system presents obstacles much more difficult to surmount than the terrain features themselves. For example, the

system will have to be capable of distinguishing between a large rock and a shadow and between a clump of trees it must avoid and a patch of tall weeds it can drive right through. In order to function in real time, this system will require a processor capable of handling as many as 100 billion instructions per second. DARPA expects the processor, as well as the vehicle, to fully operational by 1994.

The second prototype is an intelligent pilot's associate. Unlike the computers without which the new X-29 aircraft would remain airworthy for less than a second, the processors in the pilot's associate would be responsible for such "low level" chores as navigation, dealing with enemy defenses and electronic countermeasures, and identifying, analyzing and otherwise reacting to other aircraft. By codifying the tactics and strategies of veteran fighter pilots with actual combat experience, this expert system would assist less experienced pilots during the critical first few days of combat. Because the pilot's associate would be capable of speech recognition, this system is essentially only one step removed from the thought-controlled MIG in the movie "Firefox".

The final application prototype is a battle management system designed to help the commander of an aircraft carrier during hostilities. This is to be an intelligent, real-time expert system capable of recognizing and analyzing a threat, synthesizing and evaluating responses to the threat, resolving conflicting goals to select the most viable response and then implementing that response. To achieve its goals, the battle manager will have to react with lightning speed to fluid conditions and circumstances, and interact with a knowledge base of some 20,000 rules. The processor selected to run the multiple expert systems comprising the battle manager will have to be at least as powerful as the hardware needed for the autonomous land vehicle. Although a processor of this capability is not yet available, a forerunner of the computerized intelligence analyst is now being tested aboard the aircraft carrier *USS Carl Vinson*.

## Autonomous Systems

The concept of autonomy is essential to one DARPA's ultimate objectives—the development of what it refers to as collaborative and autonomous systems. A collaborative system would work closely with human operators, providing advice and assistance much in the same manner as present day expert systems. By contrast, an autonomous system functions without human intervention. Indeed, one of the linchpins of the Strategic Defense Initiative, or "Star Wars" program, is the development of hardware and software capable of analyzing torrents of data intercepted by reconnaissance satellites and remote sensors, then formulating and implementing an appropriate response. The expert systems envisioned to provide the brains behind the Strategic Defensive Initiative will consist of millions of lines of code. Even if other expert systems are used to help write this code, the task of ensuring that the SDI programs are viable and bug-free is impossible with today's technology.

## The Robotic Battlefield

Robotics, another facet of AI, figures very prominently among some of the other objectives, both announced and implied, of DARPA and other agencies within the Department of Defense. Although the technology necessary to write and implement a Star Wars expert system may be unattainable, the vision of warrior robots advancing across a chemically contaminated battlefield of the future may not be entirely far fetched: The Navy's Underwater Research lab already has plans on the drawing board for robots guided by expert systems that will roam the ocean floor, doing classified things with and to submarines, mines and torpedoes.

Of course, not all of our government involvement in AI is of a purely militaristic nature. The National Bureau of Standards has under its stewardship a totally automated machine shop. The NBS Center For Manufacturing Engineering features a 5000 square foot factory floor which is the epitome of automation. The only time a human worker ever enters this Maryland demonstration facility is when a machine must be repaired. Robots guided by expert systems are responsible for every phase of production that takes place in this machine shop of the near future. Even so-called "white collar" tasks like scheduling, resource allocation and production management are handled by expert systems. To paraphrase author and poet Richard Brautigan, the NBS machine shop is truly a facility "all watched over by machines of loving grace."

## THE FUTURE

It's certain that we'll witness a proliferation of AI applications software in the near future. Most of these packages will be of the expert system genre, designed to maximize productivity, and create friendlier, more efficient user interfaces with conventional systems and software. The following examples of hybrid software packages are not mere speculation—each is already in the prototype or development stage.

Perhaps a future release of VMS will have a smart DCL interpreter that responds to user mistakes with questions and suggestions rather than SYNTAX ERROR. Maybe a group of enterprising knowledge engineers will take it upon themselves to reduce the VAX document set to a knowledge base which new users and system managers alike can query for specific information. And possibly your 1990s word processor will be able to interactively correct spelling and grammar, help you organize your thoughts, and let you know when you inadvertently contradict yourself.

The huge binders filled with circuit schematics that clutter up your computer room could become a thing of the past. When Digital Field Service dispatches an engineer to fix your ailing computer of the 1990s, he or she may uncoil a cable from a large briefcase and plug the leads into a socket on the back of the CPU. The briefcase would contain its own supermicrocomputer which, in conjunction with a expert system and a video display system

somewhat like today's IVIS, could diagnose the computer problem and display the solution one step at a time in accurate, full-color graphics.

Computer programming as we know it today will be supplanted by mechanized expertise. Instead of coding an application program to meet the specifications of an end user, a programmer may be able to give these specifications to an expert programmer's assistant and let it attend to the task of developing an optimal program. With a large enough knowledge base, an expert system could likely respond to a request for a payroll program by building the program itself. It would do so by taking into consideration tax laws, company policies, company-specific financial and accounting systems and dozens of other factors that eclipse the traditional "HOURS TIMES RATE = GROSS" algorithm.

A multitude of things you do with computers will receive an injection of artificial intelligence, even though it won't be called AI. Instead, difficult, repetitious or monotonous tasks will be simplified by concealed software which, if extracted from an integrated application, would meet the definition (whatever it happens to be at the time) of AI.

From this vantage point, it's obvious that artificial intelligence has become a commercial reality. Expert systems are proving themselves every day, and the technology needed to produce Fifth Generation computers is being refined almost as rapidly. While we don't have to worry about Kubrick's HAL, the truly intelligent KIP is no longer a fantasy—it has evolved into a concept which will be realized, at least in part, within a decade.

Japan, through its ICOT consortium and ambitious Fifth Generation crash program, has established some lofty goals to be attained by 1990. How successful their efforts will be is still a matter of conjecture, but it is reasonable to assume that they will not meet with complete failure. Our equally significant research and development campaign should reap at least as many benefits as the Japanese program. With computer technology in a state of flux, the future is difficult to predict with any degree of accuracy. However, one concrete statement can be made today: you'll be hearing a lot more about AI, expert systems and Fifth Generation computer systems in the months and years ahead.

Data Management  
for  
High Energy Laser Systems

Ramon A. Tenorio, David Dayton  
Applied Technology Associates  
Albuquerque, New Mexico

ABSTRACT

The High Energy Laser Systems Test Facility (HELSTF) located at White Sands Missile Range, New Mexico includes an automated data acquisition and processing facility structured around two VAX 11/780'S and two PDP 11/34's. The acquisition and processing facility is charged with recording, transcribing, decommutating, analyzing and reporting data from a wide variety of time history and imaging sensors. To manage post test data processing and analysis, a program, High Energy Laser Processing and Control Environment (HELSPACE) has been developed. Designed around the VAX DATATRIEVE Data Base Management System (DBMS), HELSPACE assists the Data Base Manager, Analysts and Operators in the following ways:

- It helps the Data Base Manager Create the data base
- It helps Operators populate the data base
- It helps Operators extract pre-defined reports
- It helps Analysts run and modify their own Algorithms
- It helps the Data Base Manager back-up and restore the data base
- It helps the Data Base Manager control data base use

INTRODUCTION

The High Energy Laser Data Acquisition and Processing System (HELDAPS) located in the HELSTF at White Sands Missile Range was designed to collect and analyze data from multiple tests of diverse laser systems. Via the High Energy Laser Executive Controller (HELEX), data is collected and stored on magnetic tape. The problem that was addressed and led to the development of HELSPACE was how to manage the data recorded by HELEX to produce reliable and verifiable data in a timely manner. To perform the above tasks, detailed coordination must take place between the analyst who is ultimately responsible for the data, the Data Base Manager who is responsible for the data base, the Operators who run the system and the software packages that do the computations. HELSPACE is the tool designed to coordinate these activities.

EXISTING SOFTWARE

HELEX was operational on the VAX 11/780 at HELSTF when our effort began. The parameters that drive HELEX are stored on VAX based RMS files. These parameters define the method of recording, storing and displaying data during real-time testing. Prior to a test, the parameters are down loaded to the PDP 11/34's. The PDP's drive the hardware that collects the measured data. Pertinent parameters, needed to unpack and scale the data, are written to tape along with the measured data. After a test, the original VAX based RMS files are saved with the DEC BACKUP utility. In addition to HELEX, several other soft-

ware systems existed at HELSTF. These software systems were used to transcribe, decommutate, plot and generate statistics on one dimensional data.

DESIGN GOALS

Since the existing software at HELSTF did not contain a Data Base Management System (DBMS), it was difficult to keep pace with the dynamic nature of High Energy Laser Testing. Needed was a software system that would integrate existing software, as well as, provide a method to integrate new software all within the framework of a DBMS. The goals established for HELSPACE were as follows:

- 1) Develop a data base schema that would be flexible
- 2) Develop a method to interface existing software
- 3) Develop software to process new sources of data
- 4) Provide the capability for tracking measured data and parameters used in generating output products
- 5) Provide an environment in which simultaneous support could be provided for a system being tested as well as allow for modifications to the software and data base to support planned testing.

OVERVIEW OF HELSPACE

The HELSPACE system consists of four classes of software:

- 1) Executive---used to interface with the user
- 2) Functions---populate the data base with pointers to measured data
- 3) Algorithms---manipulate data from the data base
- 4) Utilities---create and manage the data base

HELSPACE is a system of modular FORTRAN and DCL programs that was designed to keep pace with a dynamic testing environment. New Functions can be easily integrated into the HELSPACE system. New Algorithms supporting new or modified analysis requirements are easily added. These new features can be added without affecting existing capabilities. Most programs and command procedures receive control information from the HELSPACE executive either through a command file or through DCL defined symbols. HELSPACE activates external Algorithms and processes based on user input commands.

The executive software is part of the HELSPACE executable image. It makes extensive use of VAX VMS utilities and system calls to translate user entered commands into requests that control the sequence of data processing. The executive supports password protection to insure that only authorized users execute functions. User commands may be entered interactively or through command files similar to VMS command files. In addition, a menu driven capability is provided in the executive to allow easy execution and modification of Algorithms.

The HELSPACE executive controls the activation of a number of Functions. Functions are executed as detached processes or as subroutines of HELSPACE. The HELSPACE Functions execute a variety of tasks. These include transcription and manipulation of measured data, creation of references to data sets in the data base, modification of processing parameters in the data base, creation of the data base for a new series of tests, archival of the data base at the completion of a series of tests, or archival/restoration of processed data sets. If a Function uses parameters, the HELSPACE executive accesses the appropriate DATATRIEVE domains to obtain them. A control file containing the parameters is then created for the Function.

The HELSPACE data base is implemented using DATATRIEVE. The software accesses the data base via the FORTRAN call interface utilities. Users may also access a DATATRIEVE domain interactively to modify parameters.

Algorithms are DCL procedures used as tools to extract information from the data base and manipulate the data base and data sets. They may contain run statements to activate FORTRAN programs. The development of new Algorithms is completely independent of HELSPACE. Analysts may develop tools to analyze data in any fashion. Algorithms are integrated by the Data Base Manager into HELSPACE as Processes. The advantage of running Algorithms through HELSPACE is to provide traceability, via the data base, of output products and processing parameters. The HELSPACE menu system allows the Analyst to interactively examine the input values used by Algorithms to produce output. If the Analyst desires, one or more values can be changed and the Algorithm rerun.

The numbering scheme for Algorithms and Processes was designed to correspond to four information groups in the data base. By examining an Algorithm number, the type of processing and the information group containing the data can be identified. An Algorithm number is a six character number defined as follows:

XXYYYY, where

- XX - Identifies the type of processing done by the Algorithm. Possible values of XX are:  
 OA - Extract data from the data base  
 IM - Generate inferred signals  
 DE - Generate performance assessment data  
 UT - Perform utility Functions
- YYYY - Identifies the data base information group on which the Algorithm operates. Possible values for YYYY are:  
 0000-9999 - Test Planning and configuration  
 1000-1999 - One dimensional data  
 2000-2999 - Two dimensional data  
 3000-3999 - Performance Assessment data  
 4000-4999 - Algorithm Process data  
 9900-9999 - Utility (used only for XX=UT)

The Data Base Manager assigns Algorithm numbers based on the above categories.

Every user may store in the data base his own set of parameters for an Algorithm. To uniquely identify his data, the users UIC is appended to the Algorithm number. Thus, each user can run an Algorithm with a particular set of parameters and track them in the data base.

#### EXECUTIVE-DATATRIEVE INTERACTION

The HELSPACE executive accesses DATATRIEVE via the FORTRAN call interface utility. Most of the accesses are to obtain data that has been stored in the data base. DATATRIEVE ports are used to pass the information. Record definitions are stored as character strings for each of the domains to be accessed. When a new domain is to be opened, the HELSPACE executive runs a processor that reads the record definition and issues the DATATRIEVE commands required to create the port.

To pass data through a port, a collection must first be defined. The DATATRIEVE commands that must be executed to form this collection are stored in character strings. Part of the HELSPACE executive contains a processor that reads these commands and issues the calls to the call interface to form the collection. The collection is then passed to the predefined port. When data is brought through the port, it is done as an entire record. The record is passed as one long byte string. HELSPACE extracts the various pieces of information contained in the string and stores that information in separate variables.

To modify a DATATRIEVE record, the executive uses the same processor used to form a collection. The DATATRIEVE commands necessary to modify the record are stored in character strings and passed to the processor which executes them via the call interface utility.

## HELDAPS DATA BASE

The HELDAPS Data Base was designed to simultaneously track the data from various tests of one or more systems. The following four categories of information are tracked by the data base for each test:

- 1) Test Planning and Configuration
- 2) Measured Data
- 3) Performance Assessment
- 4) Data for Report Generation

The Test Planning and Configuration category contains information required by HELDAPS in order to conduct a test. Currently, most of the information in this category is supplied by the HELEX generated configuration data base. After a test, the information in this category is used to unpack, scale and store the measured data.

The Measured Data category contains information for sensor data recorded via HELDAPS or that was brought over to HELDAPS for processing. Pointers to the data files are stored in the data base.

The Performance Assessment category stores information generated by a data processing algorithm. For example, very useful information can be gained by looking at the mean or standard deviation of one or several signals across a number of tests.

The domains in the Report Generation category are used by Algorithms that extract data from the data base. This allows for the semi-automatic generation of output products. More importantly, it allows the tracing of the parameters used from test configuration through the delivered final products.

The HELDAPS Data Base is maintained by the DATA-TRIEVE Data Base Management System. Two other VAX utilities, VAX-11 FMS and the VAX COMMON DATA DICTIONARY (CDD) are used to provide a user friendly environment in which to manage the data base.

The CDD contains the structure of the data base. Objects maintained by the CDD include record definitions, domains, procedures, tables, dictionaries, sub-dictionaries, access control lists, key and alternate key values. Figure 1 depicts the structure that was developed to support the testing conducted at HELSTF. In order to extract information from the data base, the user must point to the correct dictionary. Objects shared by multiple systems and multiple tests are stored at the HELDAPS level. Record definitions, procedures and list procedures (for most domains) are stored at this level. Objects that can be shared by multiple tests are stored at the SYSTEM level. Objects specific to a test are stored at the CONFIGURATION level. The higher in the hierarchy that objects are stored, the easier it is to maintain data base integrity since there is more sharing of schema information.

The data base files are kept separate from the dictionary structure information. This approach allows the data files to be spread across several disks. Figures 2 and 3 depict the current data directory structure. Note that the measured data directory is maintained on one disk while the remainder of the data files reside on another.

The requirement to simultaneously track several tests from one or more systems involves tracking a

tremendous amount of detail. The detail ensures that the correct record definition is tied to the correct domain, which in turn is tied to the correct data file and display form.

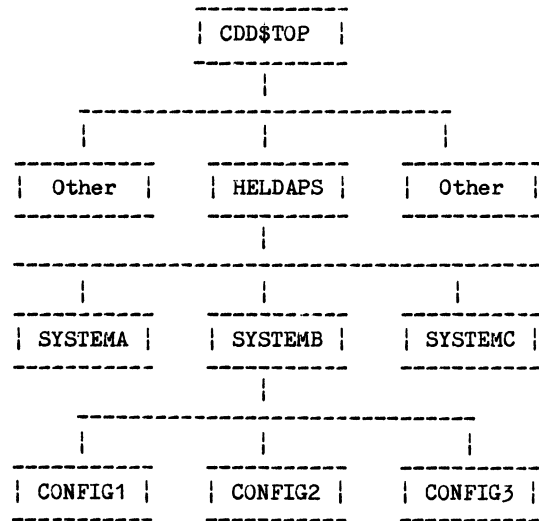


Figure 1  
Dictionary Structure

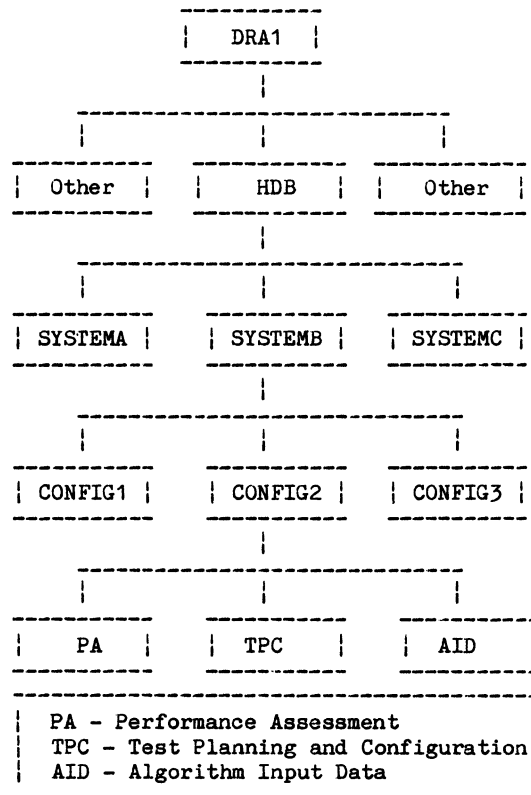


Figure 2  
Data Directory Structure  
(Disk DRA1)



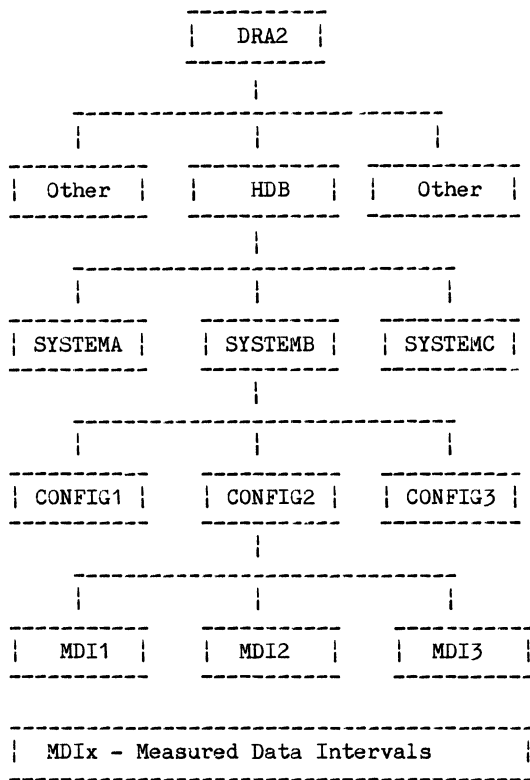


Figure 3  
Data Directory Structure  
(Disk DRA2)

When a new series of tests is initiated, a data base is generated in several steps. Most of the configuration information is supplied by the HELEX generated configuration data base. This information is copied into the new data base when it is created. Algorithm and Function pointers and parameters are copied from the previous test data base into the new data base. The remainder of the information is populated manually as the various HELPACE Functions and Processes get executed. At the completion of a series of tests, the data base is copied to tape and archived. If reprocessing is required for a test, the data base for that test can be selectively restored. Algorithm parameters can be inspected and modified for reprocessing.

#### PROGRAM FILE MANAGEMENT

The files that collectively form HELPACE and the HELDAPS Data Base fall into one of the following categories:

- Source Code
- Algorithms
- DCL Command Procedures
- Include files
- Data Base files
- Object libraries
- Executable files
- Forms files
- Documentation files
- Historian files
- Scratch files

In a dynamic environment, such as the one that exists at HELSTF, changes to one or more of the above categories are constantly being made. The ideal structure for these files is one that allows for the processing of data by baselined software, as well as for modifications to support new requirements. Figures 4 and 5 reflect the directory structure in place at HELSTF to support processing of data using baselined software. The following is a brief description of what each directory on DRA1 and DRA2 contains.

- HDB Data files at the HELDAPS level
- ALGORITHM HELPACE Algorithms to support prior and current analysis requirements.
- DOCUMENTS All on-line documentation
- BIN All executable programs
- FORMS FMS forms to support DATATRIEVE domains
- SYSTEM Data files at the SYSTEM level
- TEST Data files to TEST level
- AID Data files to support Processes
- TPC Data files to support Test Planning and Configuration domains
- MDI Data files that track measured data in data base
- PA Data files to support performance assessment domains

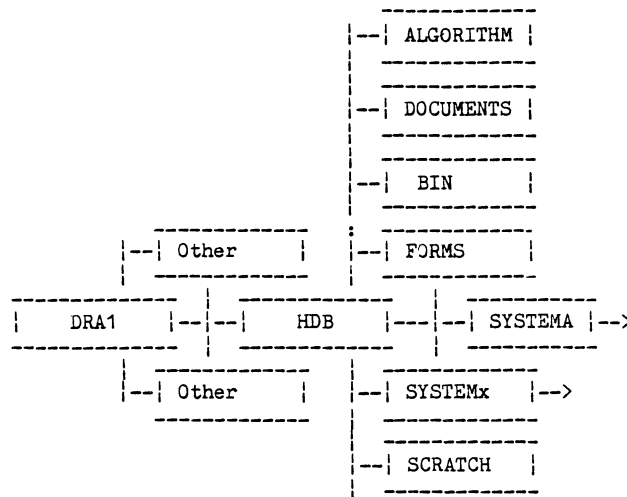


Figure 4  
Data Structure to Support  
Production Processing  
(Disk DRA1)

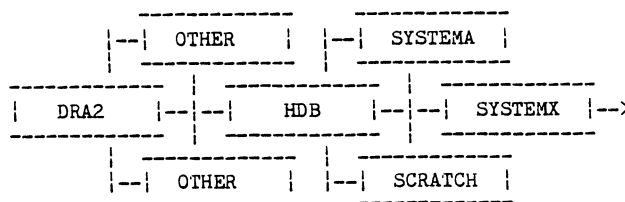


Figure 5  
Data Structure to Support  
Production Processing  
(Disk DRA2)

Storage of the measured data files requires large amounts of disk space. In order to maintain the Data Base on-line for ad hoc retrievals, these files are stored on DRA2. The HDB, SYSTEM, CONFIG and MDI (Figure 3) directories point to the location of these data files. If the data files are taken off line, their location is tracked in the Data Base.

### SUMMARY AND CONCLUSIONS

HELSPACE is a dynamic data management facility that provides the following features.

1. It is based on the DATATRIEVE DBMS utility using the call FORTRAN interface utility.
2. It provides a user friendly executive to control data processing and management functions.
3. The executive uses processor modules to execute DATATRIEVE commands via calls to the call interface. Data is passed to the executive through ports.
4. It is designed to track collected and processed data sets, processing parameters, and provides a degree of automated processing.
5. It tracks user written processing Algorithms.
6. It is modular and is easily modified.

In the course of our work, we encountered the following shortcomings in using the DATATRIEVE call interface utility:

1. When ports are used to access DATATRIEVE domains from FORTRAN programs, the domain record definitions must be coded into the program. If the record definition is later modified, program modifications and recompilation are also required.
2. Accessing records through the call interface is slow. This seems to be due to computational as well as page fault overhead when making a DATATRIEVE call.
3. The lock wait call interface option works only one way. If a user locks a DATATRIEVE record, and second user attempts to access it, the second user will wait. On the other hand, if the first user accesses a record but does not lock it and a second user attempts to lock the record, the second user will receive an error.



# Encryption for Beginners

B. Z. Lederman

2572 E. 22nd St.  
Brooklyn, N.Y. 11235

## Abstract

The purpose of this paper is to make people aware of what data encryption is, how it is used, who needs it, and why it is needed. It is intended as an introduction to the subject, so it will not go deeply into the mathematical internals of ciphers

As is true for many subjects, what something is and how it is used is often interlinked, so that one needs to understand one before the other can be explained; so to begin with, some very simple definitions will be given, and later they will be expanded.

Cryptography covers the general field of transmission of information which is protected from unauthorized access, and includes secret writing (concealing a message by various means), codes, ciphers, and their use and defeat. Lately, encryption and decryption have come to be used in place of encipher and decipher to refer specifically to the use of ciphers to protect data, and will generally be used as such here.

Stated more simply, data encryption is a method of protecting data so that it can be accessed only by the people who are supposed to be able to get to it. This definition, while correct, is rather vague (it could apply equally well to the physical protection of data such as locking it up in a safe, or translating it into an obscure language): it does, however, explain the purpose of encryption, which is to limit the accessibility of selected items of information. This will be explained first, as it is desirable to understand why access should be limited to understand how it is to be done.

## Do You Need It?

If you are working on a computer system which can be accessed by one or a very limited number of users, and which has no outside lines (no modems or dial-in lines), and which stores all information on easily removable media (floppy disks or tape cartridges), and you always remove this media and lock it in a safe when you are

not using it, then you may not need encryption. If you can eliminate all access to your data other than by having the key or combination to the safe, and if no-one can look over your shoulder or otherwise tap into your computer or terminal lines while you are examining your data, then access to your information has been made about as secure as possible through physical means, and encryption is probably not necessary. Unfortunately, this ideal state of affairs does not often exist. Sometimes your storage media cannot be kept in a safe, or you must store your information on a fixed disk which cannot be removed, or you must share the system with many other users at the same time, or you must have dial-in lines so that people outside your physical location can access the same machine, or you must send information to other locations: in any of these cases, you may need to limit access to your information, and encryption is one method of doing this.

The immediate reaction many people have to this is:

"Our computer is used only by people within our company. We don't have dial-in lines, [or our dial-in lines are secured by other methods, such as passwords or dialback], and all of our terminals are within our company area. Why do I have to protect my data?"

## Some Reasons.

Even in this situation, there may still be good reasons for using encryption. First, you may have information which you are obliged to keep confidential. If you use your system to administer company medical benefits, for example, you may be obliged to keep personnel medical records confidential. Without some sort of encryption or other protection scheme, it may be possible for many people in your

company to peruse the medical records of other employees at will. Even if you are certain no-one will do this, increasing demand for rights to privacy of personnel records may set a legal requirement that you protect information from indiscriminate access. (Note that encryption will not protect against the persons who must still have access to the data: other checks are needed to insure that persons who must have the data will not misuse it.)

Next, there may be information you want to keep confidential. If you use your system to keep track of employee performance records, or calculate salaries as part of your budget planning, you might not want the employees involved to read or modify that data. It is all well and good to say you trust your employees, and probably most people can be trusted: but locks were invented to keep out the small percentage of society which cannot be trusted. I rather imagine that most people reading this lock their houses and cars before leaving them, even if they trust most of their neighbors: if you would do that, then you probably have information which should also be "locked up". Similarly, you might be preparing information for contracts, order placements, payroll records, competitive bids, and similar information which could represent a significant portion of your company's assets, and might be several times the annual salary of many of the people who have access to it (and they are not always only the people whom you think have access to it). The more important an item of information is, the more likely it is that someone could benefit by getting it, and therefore the need to protect it increases directly with it's importance.

The case where a "hacker" or other unauthorized person calls into a computer system and proceeds to cause various type of mischief and/or damage is one that probably most people fear. You may have a system where it is necessary to have dial-in access for your own personnel, and it then becomes necessary to guard the system as much as possible. There are various methods of limiting access to a system through passwords, or through hardware, which are outside the scope of this paper. Data encryption can act as a second line of defense, however, and should also be considered. In many cases, "hackers" are simply looking for files they can read, or programs they can run: encryption can make data unreadable and programs unrunnable, and thus defeat two of the hackers main goals. Encryption will not prevent the random modification of data (where the modifier doesn't care what the change actually does) or deletion of files: other methods of protection are required to guard against that type of damage.

The situation may also be reversed, as many computer users do not own their own systems and have to use time-sharing or other outside computer processing to store data and provide other computer services. In this case, you may have little control over who in the world has access to your data. An encryption scheme that can be implemented on your own data on the outside machine would be one way of protecting your information. Similarly, many companies store copies of their records in outside warehouses or other storage facilities to protect against fire or earthquake damage at their main location, and while such facilities usually offer guarantees against unauthorized access, some extra protection might be desirable.

One last situation which probably occurs to most people is when data has to be transmitted from one location to another, usually over some public facility (telephone, teletype/telex, leased communication line, air freight, or mail). It is actually more likely that the data will be accessed from within your company than from without (intercepting telephone channels from microwave links is possible, but rather difficult), but the more important the information is, the more likely it is that someone will try, and it wouldn't hurt to take some reasonable precautions. If you are engaged in any type of electronic funds transfer, such as depositing your employees payroll directly to their bank accounts, or transfer of company assets to your bank or to other companies, the sums of money involved may be so great that not encrypting the data in some way is courting disaster. Consider what would happen if someone were to change the records just once: if that would seriously hamper your business, or cost you a significant amount of money (either by direct loss or the effort to replace the missing information, or loss of goodwill of the person at the other end), then you should consider encrypting your data. Remember that the true cost of data might not be just what it cost you to obtain it, but also what it will cost if you lose it.

#### Other Types of Protection.

It can be seen, therefore, that many users will have some use for a data protection scheme of some kind, as nearly everyone has some type of information which is not to be accessed by everyone else. This leads to the methods which can be used to protect information. Various computer operating systems are in use today, some of which include access protection through requiring users to log into accounts, or various methods of verifying that persons accessing dial-in lines are properly authorized, or

through protection codes within the storage system (such as the file protection codes used in RSX-11, RSTS, and VMS). These are outside the range of this paper, but it will be mentioned that they don't always provide the limit of protection needed, either because there has to be at least one privileged user of the system who can bypass the checks, or because backup copies of the data must be stored off of the machine, or from other limitations of the system. Even when such schemes work well, they may not be enough, and they don't work at all if the information has to be sent outside (by wire or mail, etc.). This leads us back to data encryption, which will allow the information to be protected by a method which is independent of any protection which may be provided by the operating system. This does not mean that other protection schemes should not be used, or that encryption is the answer to everything, either: different protection schemes cover different areas, and usually complement rather than substitute for each other.

Once the need for some type of data protection is recognized, a protection scheme must be selected. As previously mentioned, cryptography covers, in general, secret writings, codes, and ciphers.

#### Secret Writing.

Secret writing covers such things as invisible inks, and concealing messages within other messages. This is a highly specialized field, and one which is not likely to have much general application: it is usually too cumbersome for easy use, and is not applicable to storage of large amounts of information on computer media. Just to show what it is like, consider the message:

"Inspect details for Trigleth,  
acknowledge the bonds from Fewell."

which doesn't seem to mean anything. If you take the third letter of each word, however, you get the message "Strike Now". This is an example of secret writing, (a method which follows a fixed formula like this may also be called a concealment cipher), and it can be easily seen that it would not be easy to use: if it had no other faults, the concealed message has become over 6 times it's original length, and if you have to pay for disk storage space or transmission costs, you can see a big disadvantage to this type of protection. Invisible inks can be used on paper messages, but obviously won't work at all on data stored on disk or magnetic tape. (There was one fictional story where

a message was written on a reel of tape with a grease pencil, but this tends to gum up the drive, and isn't very practical.) They can be useful to authenticate documents, as they cannot be duplicated by photocopying machines, but again, this is a field where expert assistance from a printing company or ink manufacturer is required. We will not give any more attention to this subject.

#### Codes.

A code is the arbitrary mapping of symbols to other symbols. It is usually one to one, but can be one to many or many to one. One example of a code which is in very common use every day is ASCII, the American Standard Code for Information Interchange, used by most computer terminals to map binary signals to numbers, letters, and other characters, a portion of which is shown here.

|         |       |       |       |
|---------|-------|-------|-------|
| 040 SPA | 060 0 | 100 @ | 120 P |
| 041 !   | 061 1 | 101 A | 121 Q |
| 042 "   | 062 2 | 102 B | 122 R |
| 043 #   | 063 3 | 103 C | 123 S |
| 044 \$  | 064 4 | 104 D | 124 T |
| 045 %   | 065 5 | 105 E | 125 U |
| 046 &   | 066 6 | 106 F | 126 V |
| 047 '   | 067 7 | 107 G | 127 W |
| 050 (   | 070 8 | 110 H | 130 X |
| 051 )   | 071 9 | 111 I | 131 Y |
| 052 *   | 072 : | 112 J | 132 Z |
| 053 +   | 073 ; | 113 K | 133 [ |
| 054 ,   | 074 < | 114 L | 134 ® |
| 055 -   | 075 = | 115 M | 135 ] |
| 056 .   | 076 > | 116 N | 136 © |
| 057 /   | 077 ? | 117 O | 137 - |

It isn't usually thought of as a code, and it certainly isn't a secret, but it is a code: it transforms one type of data into another through an arbitrary mapping. Note that the mapping is indeed arbitrary, even though the letters do follow the alphabet for convenience: there is no reason why they would have to do so for the code to work.

Another code which better fits the general public's perception of a code is the type of code which has been used for telegrams, a portion of which is reproduced here:

MUWUB Improving rapidly  
 MUXAW Improving slowly  
 MUXEX Is not improving as I/we  
       could wish  
 MUXIZ Is there any change  
 MUXNO Is there any improvement  
 MUXPU Progressing satisfactorily  
 MUXRY Sorry to year you are  
       (..... is) ill  
 ---  
 MYGEL How would  
 MYGIM HURRY (See Haste)  
 MYGON HYPOTHECATE-D  
 MYHAL IF  
 MYHCI And if  
 NYHDO And if not

Ciphers.

A cipher is a method of transforming data from one form to another through a logical process, usually with a geometric or mathematical basis. Since a cipher is a method or system rather than a group of pre-defined mappings, it should be possible to transform any "plain" or "clear" text, regardless of length or content, into a single enciphered message. This is more easily understood with an example, such as a simple geometrical cipher. I will take the familiar phrase,

"THE QUICK BROWN FOX JUMPS OVER THE  
 LAZY DOGS BACK"

and write it out in a square in the usual fashion, left to right, top to bottom.

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| T | H | E |   | Q | U | I |
| C | K |   | B | R | O | W |
| N |   | F | O | X |   | J |
| U | M | P | S |   | O | V |
| E | R |   | T | H | E |   |
| L | A | Z | Y |   | D | O |
| G | S |   | B | A | C | K |

To encipher this message, I can take the letters out by some sequence other than the way they went in: for example, top to bottom, right to left. (This is an example of a transposition cipher, as it works by transposing or changing the order of the letters in the message, but not the letters themselves.) This will give me:

"IWJV OKUO OEDCQRX H A BOSTYBE FP Z  
 HK MRASTCNUELG"

which doesn't look anything like the original. The underlying principle here is that there is a definite method of transformation between the original text and the enciphered text without considering the actual content (even if it is not obvious on a cursory inspection), whereas in a code the transformation was completely arbitrary and very sensitive to content. Because ciphers work on a method of translating data from one form to another, they are generally much easier to implement on a computer, and they are generally much less data sensitive than codes would be. In this example, each character could easily be a byte or word of binary data, and the scheme would work just as well.

and so on. It can be seen that the mapping between the original phrase (the "clear" text) on the right and the code word on the left is completely arbitrary, and that the book is the only way to go from one to the other. This particular code had the advantage that in most cases the coded text was much shorter than the original message: two groups of five letters could be pushed together to make one 10 letter group, which was counted as only one word in the cost of sending the telegram. Since the mapping is arbitrary, codes can be very secure. Generally, you have to have the arbitrary mapping in order to defeat (or "break") the code, though if the code is re-used often enough, the mapping can sometimes be deduced. They are also vulnerable if one can obtain a copy of the clear text and the coded text which goes with it, and of course are defeated if the wrong person obtains a copy of the code book. Some authorities consider book codes like this that are used once only to be completely unbreakable, and it would be easy to use a computer to generate lists of arbitrary code words to use.

Codes do have many disadvantages in the computer environment, however. A computer program to automatically code a message with a scheme like the example would be very complex, as the context of the angles message is needed to search through the list of phrases on the right and find the appropriate code word: decoding the message by looking up the letter group would be a easier. Encoding large strings of numbers is tedious and likely to increase the size of the message, and there is always the problem of what to do if you need a phrase which is not pre-defined in the code book. Binary data cannot be coded at all which this particular scheme, and would be difficult to encode with most coding schemes. Since we would like a method which would work on a computer, and accommodate a wide variety of data with a minimum of human intervention, we will not consider codes further.

There are a great many types of ciphers, some more secure than others, and some easier to use than others. One which is very common, and even occurs in some daily newspapers, is a simple letter substitution, where one letter is replaced by another. For example,

ABCDEFGHIJKLMNOPQRSTUVWXYZ

can be replaced with

EFGHIJKLMNOPQRSTUVWXYZABCD

This is a substitution cipher, which changes the letters in the message, but not their order in the message. This would make the sample phrase "THE QUICK BROWN ..." come out to be:

"ZLI UYMG0 FVSAR JSB NYQTW SZIV ZLI  
PEDC HSKW FEGO"

Since this is a one to one mapping, I am going to leave it to the purists to determine if it is a code or a cipher, though it is content insensitive (there is obviously some overlap between some codes and ciphers). The drawback to a simple cipher like this is that it is too easy to break with just a pencil and paper, and with even the least expensive home computer it is literally child's play. (You can read "The Gold Bug" by Edgar Allan Poe or "The Adventure of the Dancing Men" by Sir Arthur Conan Doyle to find out how.) There have been many other, more sophisticated, transposition and substitution ciphers than the ones demonstrated here in use in the past few centuries, but since they were all implemented by hand, they are all too easy to break by modern methods. You can simply go out and buy a number of books that will tell you exactly how to do it with just pencil and paper, and the proliferation of home computers makes most of them very simple to break indeed. They may still be adequate for some purposes however, but considering how good a cipher needs to be will be discussed later.

If existing ciphers are too easy to defeat with computers, then what is left? The answer is that most modern encryption schemes are based on the same principles as older ciphers, but use the power of the computer to expand the magnitude of the scheme. For example, in the transposition cipher shown, the box was 7 letters on a side: it could be made larger, but when encryption is done by hand, a box much larger than 15 or so on a side becomes too cumbersome to use. With a computer,

however, there is no limit to the size of the box: simply increasing the box to 100 per side makes it too large to "break" the cipher by hand. This scheme of using the computer to expand on a good encryption method can be used to create ciphers that are difficult to defeat, even with another computer (the box cipher would still be too easy to break by computer and is given only to illustrate the idea). One which I have used is a variation on the periodic number substitution (also known as an addition or Vigenere) cipher. In this scheme, a number sequence is added to the text: a simple example would be to add the sequence

1357135713571357135713571357135713 etc.

to the numeric value of the ASCII characters in the message

THE QUICK BROWN FOX JUMPS OVER THE  
LAZY DOGS BACK"

to get this:

UKJ'RXNJL#GYPZS'GR]'KXRWT#T]FU% [  
IH%SB]®'ERLZ!EFJL

With a number sequence this short, the cipher would not be too secure (you can see even in this short message that a SPACE becomes a ' four times, and the sequence "SPACE-something-U" has twice been changed to "'-something-X", for example) though it is more secure than the simple substitution cipher shown before. Various methods of obtaining a less repetitive sequence have been tried in the past, but usually produce no real increase in security. Using the computer, however, a number sequence can be generated that appears to be random, and is thousands of digits long. Most computer languages have a random number generator (or more accurately, a pseudo-random number generator, as the sequence can be repeated exactly when desired), such as:

LET A = RND(B) in BASIC, and  
A = RAN(B) in Fortran,

and similarly for other languages. There are theoretically an infinite number of such pseudo-random sequences, and even for a specific generator there are a very large number of specific sequences: in DEC's Fortran-77, the number that starts the sequence (the variable B) can have at least two billion possible values. This



particular cipher is sometimes called the Fast "Infinite-Key" method, and has been widely used with good results. We could then repeat the above procedure by generating a pseudo-random number sequence such as:

1986833925153857265815341697347183 etc.

and adding it to

THE QUICK BROWN FOX JUMPS OVER THE  
LAZY DOGS BACK

to obtain

UQMYXLLM&CWR\_S'HU](KZPTT&X]HV'U  
PH'UJ\_a'JULZ)JHGM

At first glance, this doesn't appear significantly different from the first example, but if someone were to attempt to defeat the cipher by the usual method of looking for repetitive patterns and common adjacent letters, they wouldn't find any, and would not be able to defeat the cipher. This cipher has the additional advantage over the "box" cipher in that the characters can be processed in the order they are read: in the box cipher, a large portion of the message has to be read in and stored before any of it can be processed. In most computer ciphers, it is an advantage to be able to process the message serially, and to not have the length of the message have any effect on the encryption scheme itself, especially then the messages being processed are being transmitted from one place to another (over a communications line, or to a disk or tape drive are two examples).

It can be seen, therefore, that even though the computer has made it easier to defeat some encryption schemes, the power of the computer can also be used to raise the complexity of a cipher to the point where it is very difficult to defeat, even with another computer. This is the basic principle behind most good modern computer ciphers: the use of the computer to raise the complexity of the cipher until it is (hopefully) beyond the ability to defeat by any practical means.

It was mentioned that the telegraph code example shown earlier also compressed the information into a more compact form. There are a number of data compression schemes in use on computer systems to minimize the amount of space data occupies when stored, or to reduce the amount of time needed to transmit information from one location to another (and hence reduce the cost of transmission). Some of these compression schemes could also be thought of as ciphers, as they transform data from one form to another. While they have the obvious advantage of compressing the data, generally the compression algorithms are too well known for this to act as a really secure cipher. You can, of course, compress your data before or after encrypting it as long as the cipher is not data sensitive.

With some understanding of what encryption is, we can perhaps present a better definition. One such definition could be:

"Encryption is a method of transforming data into a state where it is not easily available to persons other than those for whom it is intended (using ciphers)."

This is a very general definition, and it does appear to be somewhat cumbersome, but it is worded in this way deliberately. Note especially the emphasis of the phrase, "not easily available". Generally, no encryption scheme is absolutely secure from ever being defeated, and a decision has to be made as to how good a scheme is needed. From a practical standpoint, the real purpose of encryption can be defined as this:

"To make obtaining the data more expensive than the data itself is worth."

(Where expense is counted in time, effort expended, cost of labor, cost of computer services, etc.)

While this definition may not precisely define a cipher, it does clearly define the goal encryption should achieve.

To evaluate a potential encryption scheme, one must consider from whom the data is being protected. Some possibilities are:

1. Curious employees
2. "Hackers"
3. Outside visitors
4. Service personnel and/or vendors
5. Competitors
6. The Criminal Element (internal or external)
7. The IRS
8. The "spooks" (CIA, NSA, KGB, MI5, etc.)

among others. The first four can probably be discouraged with even a very simple cipher: as mentioned before, most "hackers" and other idle curious are simply looking for files that can be read or run. If they were to see a file such as this:

```
RTP $%& &.2H8I]).4HHQPPJ8IKNUIOQPP
RUP $%& &.3H8I],%.H342DH).4H8III
RVP 2%-
SQP 02).4 B#/-054!4)/. /& -/2'!'% 0!9-%.4
SRP 02).4
SUP 02).4 B0,%!3%).054 4(&% 02).#)0!, H7
SVP).054 0
SWP 02).4 B).054 4(% !..5!,).4%2%34 2!4% H
SXP).054)
SYP 02).4 B).054 4(% 4%2- H). 9%!23IB[
TPP).054 4
TQP 02).4
TSP 4]4JQR
TUP 1])
TVP))OQRPP
UPP -]%.2H0J)OHQM0OHQK)I>4II
UTP 02).4 B02).#)0!,B[4!"HSUI[BDB[0
UUP 02).4 B).4%2%34 2!4%B[4!"HSTI[l[BEB
UVP 02).4 B4%2-B[4!"HSTI[4[4!"HTPI[B-/.4(3B
UWP 02).4 B-/.4(,9 0!9-%.4B[4!"HSUI[BDB[4!"
```

they might well pass it by, or maybe make a few simple attempts to read the file as if it was binary data. But if anyone should happen to figure out or guess that it is really a BASIC program, then it would not take long to decipher it, as it happens to be encrypted with a simple letter substitution cipher. Since a computer is going to do the work, it would be just as easy to use a more secure cipher, and one which will transform the data into something which will not look like obviously encrypted data when examined. For example, the "Infinite-Key" method takes no more computer time or disk space than simple substitution, is very much more secure, and the resulting data doesn't look at all like text, so there is no reason to use the simple substitution when such superior methods are easily available.

If interception of data by a competitor, or by a dishonest employee (which is really the greatest threat) is a serious consideration, then you will probably want the most secure cipher that can be reasonably implemented (one which protects the data well, but will not use up so great an amount of computer resource that it becomes more expensive than the data it is protecting).

If you intend to protect your data from categories 2, 3 and 4, then other protection schemes should be your first choice, such as not allowing outside visitors to wander un-escorted about your plant, removing your data from the system before allowing it to be serviced by outside personnel, and using various protection schemes to prevent un-authorized dial-in access. Encryption of data can act as a second line of defense in these cases, however, and should still be considered: it must be stated again, however, that encryption is not necessarily the best solution to every situation, and that all methods of protecting data need to be evaluated to determine what best suits a given need.

Against the last two categories: you have to be realistic, and understand that any government agency that can put the gross national product of a world power into it's efforts is going to be able to break any cipher you could use. That doesn't mean you have to make things easy for them, and there are ciphers available now which are very difficult for anyone to defeat, but you must remember that no cipher is absolutely unbreakable.

#### How Good is "Good Enough"?

It was stated that a good encryption scheme costs more to defeat than the information is worth. This means that the cost of the labor expended, and computer resource dedicated to the task are more than the ultimate value received from the information which may be obtained. For example, the only ways known to break the Infinite-Key and DES ciphers is by brute force: trying every possible key, and looking at the result to see if it makes sense. Even if someone is willing to dedicate a computer to the task, it could take months or even years of effort to break one message, by which time the information may be useless. In addition, the time a computer spends on breaking the code cannot be used for anything else, like doing payroll, or inventory, or other normal business functions. If you are preparing bids on a contract which will yield, say, \$10,000 and a competitor tries to steal your information and under-bid you, then your encryption scheme is successful if it either takes so long to break cipher that the competitor can't meet the deadline for submitting bids, or if it costs the competitor more in computer resources than the \$10,000 or so that the contract would yield: even though the cipher is broken, the person who broke it comes out with a net loss. Few "hackers" are going to have the patience to let their

home computer run for several months or years to decrypt one message and not use the computer for anything else, and not much information is so valuable that it would be worth while renting a Cyber or Cray super-computer for several months to break the message relatively quickly (unless you are a government agency, and can do whatever you like).

It is possible that someone within a company might use the company computer to try to break a cipher by brute force, reasoning that the computer time doesn't cost them anything. Since defeating a good encryption scheme would use up relatively large amounts of computer time over an extended period, it should be possible to detect if anyone within a company is using the computer system in this manner, and deal with the problem directly.

#### What Else Must I Do?

A consideration which is equally important as the selection of an encryption scheme is keeping the keys themselves secure. Just as it would do no good to buy the most expensive lock and lock your house and if you then put the key under the door mat, it does little good to encrypt your data if anyone can get the key. In terms of internal security, this often means correct selection of a key to use: since most modern ciphers use a number as the key, there is a great temptation to use an easily remembered number such as your telephone number, birth date, social security number, wedding anniversary, or some such number as a key. Unfortunately, any number that you can remember easily will also be easy for anyone who knows you to guess. If you are trying to protect data internally in your company, using such a number would defeat the best cipher: rather than having to try several billion possible keys, the number of attempts are reduced to a few dozen or so. This leads to the paradox that you must choose a number you can remember (or you may never get your data back if you forget the key), but one which no one else is likely to guess; or else you have to write the number down, but in a place where no one is likely to get it. The latter scheme is probably better than trusting to memory, but you should not keep important numbers laying about: keep them in your wallet (and keep your wallet with you), or in some other secure place. Similarly, don't put them in the telephone directory or card file that sits on top of your desk, or in other easily accessed places. It is also a good idea to change the keys periodically, especially if it is being used for data transmitted externally. (Internally, the threat is greater that

someone will figure out your key, or may see you type in the key, or be able to compare the encrypted data with the "clear" data, and deduce the key that way). Basically, you must use at least as much caution in dealing with cipher keys as you would use in handing out door keys to your plant, or electronic lock keys to your personnel: they all protect your assets, and have to be treated with the same respect. While you can hire guards for physical security in a plant, you cannot do the same for information in a file or transmitted over a wire, and information is easier to move than equipment; so if anything, the cipher keys must be kept even more secure than other kinds of keys.

#### Public Key.

When data has to be transferred from one location to another, then the risk is doubled, as the key has to be kept in two places. One absolute rule is that you never, ever, transmit the key with the data it protects (you might just as well not bother encrypting at all). It is usually a good idea to use an encrypted transmission to send the next key to be used at one time, and the data at some other time, and that both parties must exercise the same caution in protecting the keys. Otherwise, you must use some secure method of transmitting the keys to the locations where they will be used (such as sending someone you can trust to carry them by hand), and storing them in a safe or other secure location. One partial solution to the problem is the Public Key method of selecting keys. This is not an encryption scheme, but is a method where two people can create a large numeric key by each selecting a number which forms half of the key, and were each party knows only half of each key. The advantage of this method is that one half of the key can be made public, and anyone can use it to encipher a message intended for you, but only you can decipher the message using the other half of the key which was kept secret. This can also be used for source verification if both halves are kept secret: for you to be able to decipher the message, it will have to have been enciphered using the matching half of the key. The method is based on the fact that it is difficult to factor a very large number which is the product of two very large prime numbers (each party picks one of the large primes): lately, there have been some announcements that it might not be as difficult to break as was formally thought, but it may still be useful to many people. If you are transferring data within an organization, and can keep the key secret at both ends, then Public Key isn't necessary: it's

primary use is where the security of the key at one end isn't known, or must be made public.

### Encryption with Hardware

and the DES.

So far, we have considered encrypting data while it is in the computer system, and before it is stored or transmitted. This is not the only way it can be done: it is also possible to attach a device to a communications line so that information passing through it is encrypted in one direction and decrypted in the other direction. For example, the device could be attached between your computer and a modem, so that "clear" information being transmitted from your computer will be encrypted before it goes into the modem and out into the world. Most of the special hardware currently offered for sale for this purpose use the Data Encryption Standard (DES), also called the Data Encryption Algorithm (DEA). This method of encryption was developed by the National Bureau of Standards to provide a standard, secure encryption method, and it involves many stages of transposition and substitution. Furthermore, there are several modes for data to pass through the encryption scheme: the method any individual will use depends upon the application. According to the developers, the DEA is intended for use only with hardware encryption schemes for several reasons, two of which are security of operation and verification of correctness.

The first reason includes protecting the key and the encryption method: if it is in special hardware, you have to enter the key into that piece of hardware, and it won't be "floating around" your computer system as it might be if a software program was used. Similarly, only the manager in charge of the special hardware knows what the key is: you don't have individual users losing their keys (or giving them away). In addition, there are often ways for one user to monitor another user's program on the same computer (for example, to watch someone type in their key), and it was felt that it would be more difficult to tap into a separate piece of hardware. With the protection in hardware there is the additional advantage that no-one can forget to encrypt data before sending it out: anything which is transmitted on that line is automatically encrypted. It was stated

before that encryption might not prevent "hackers" or other un-authorized persons from accessing a system, but the one exception is if there is a hardware encryption device placed between the system and the modem which always encrypts the data on that line. Encryption would then prevent unauthorized access, as anyone who wishes to dial in on that line must have an encryption device which uses the same cipher and key. In a similar manner, a hardware device can be placed between a computer and a peripheral device: for example, a disk. If this is done, then all data on the disk is automatically encrypted, and you don't have to worry about users forgetting to encrypt sensitive data, or service personnel reading it during maintenance.

The second reason, that it would be easier to test if the hardware is working correctly than to test if a program is working correctly, is a reason I do not entirely agree with. It also means that the use of DES would be limited to those applications that can send the data through a line to the special hardware, and that you would have to buy the special hardware for every location which wanted to encrypt data: this meant that locations with personal or small business computers had to buy an encryption device that was as large and as expensive as the computer itself. This is changing rapidly as more large scale integrated circuits which implement the DES are being placed on the market, so that the cost of a peripheral device that does encryption in hardware is decreasing, but it still has many drawbacks for some users. As a result, software houses are offering data encryption programs that use the DES method to encrypt data on the system itself with no special hardware.

Use of the DES is likely to increase in the next several years, especially where information has to be exchanged between different companies, because it is a standard and it is possible to obtain different pieces of hardware or software which implement it and will still be compatible, as they have to meet the standard to be able to say they use DES.

Like most modern ciphers, DES uses a numeric key, and there has been some arguments lately about how secure DES really is, based on the length of the key, which is 56 bits (the scheme adds bits to make it 64 bits long). Some of the developers suggested that the key should be 128 bits long, but the National Security Agency required the NBS shorten the key: some critics have suggested that a key of this length is such as to be virtually unbreakable by anyone except the NSA

itself, which tries to stay about 10 years ahead of everyone else in technology. Even if this is true, DES will probably be secure enough for most commercial users for the foreseeable future (remember what was said earlier about determining from whom you wish to protect your data).

#### Extra Precautions.

If you expect a real effort will be made to defeat your encryption scheme, there are a few extra precautions that can be taken to reduce the risk. The easiest way to break a code is if you have a copy of the enciphered message and the clear text together, and can compare the two to work back to the cipher. This indicates that access to important information should be carefully restricted: for example, if encryption is used to protect data during transmission, then when the data is deciphered and safe, the enciphered copy should be erased or destroyed. If it is carelessly discarded, it might give someone a chance to work on it at leisure, especially if the threat is within the company, where the clear text might also be available. Some newspaper codes were broken because the text of an article was transmitted in cipher (by radio, where it could be heard) and then printed word for word the next day in the paper: sending the contents of the article but re-wording it before releasing it to the public solved that problem. Similar precautions could be taken if such things as financial reports are to be transmitted: if possible, don't transmit the data in exactly the same form in which it will be published. In the case of business letters and memos, most start with a date and the person to whom it is addressed, and someone could know (or guess) how the message starts, and use that to cut down the number of attempts needed to find the key to the cipher: one way to stop that is to arbitrarily cut the memo in the middle somewhere, and put the last part before the first. The recipient, after deciphering, can easily see where the real beginning is, and move it back where it belongs. In short:

Don't be predictable.

There are also a few other precautions one can take if you feel that someone is really trying to defeat your encryption scheme. If you think someone is trying to get your key by brute force, you can put random garbage

at the beginning and end of your data: anyone who is trying a key and checking only the beginning of the file to see if the data makes sense will not realize it if they do find the right key, as the decrypted data still won't make sense. Of course, anyone can simply check the entire contents of the message for every key tried, but this is much slower, and anything that slows the process of defeating an encryption scheme means the scheme is that much more secure. If there is some reason to believe that whole messages are being intercepted and stored (with some ciphers, the more data you have, the easier it is to find the key, though it might not help much with Infinite-Key, DES and some other modern ciphers), then you should change the key more often than you might otherwise do. In any event, you should not use a given key for too great a period of time, just in case someone is collecting your messages. You can also occasionally send out messages which are the same length and otherwise look like your real messages, but which contain enciphered garbage. The contents (before enciphering) should look as much like real data as possible, without actually meaning anything. This will add to the difficulty of defeating the encryption scheme, but is only worth while if there is a real possibility that someone is making a concerted effort to break the cipher.

#### Bibliography

There are a number of good descriptions of cryptography in popular literature. In addition to the two examples of the simple substitution cipher given before ("The Gold Bug" by Edgar Allan Poe and "The Adventure of the Dancing Men" by Sir Arthur Conan Doyle), two books by Dorothy L. Sayers (in addition to being entertaining in themselves) are of interest. "Have His Carcass" contains a good description of the Playfair cipher (a good combination transposition and substitution cipher which is easily worked with only a pencil and paper), and a good description on one way to attempt to break it which also clearly shows the hazard of sending messages in a form which allows the content to be deduced. "The Nine Tailors" contains an extremely ingenious example of secret writing. Both are currently published in paperback.

On a more formal basis, the following will be useful:

"Cryptanalysis, a Study of Ciphers and their Solutions", by Helen Fouche Gaines (Dover Publications, Inc.)

though written before computers, contains thorough descriptions of many ciphers, and specifically the methods used to defeat them, with worked examples and reference tables. Dover has a mail order department.

"Security and Privacy in Computer Systems" by Lance J. Hoffman (Melville Publishing Co.)

treats a wide variety of computer security subjects, one of which is the use of data encryption. It includes a good description of the "Infinite Key" cipher, with a mathematical test of it's effectiveness. It also covers operating security, physical plant security, and other subjects.

"Cryptanalysis for Microcomputers" by Caxton C. Foster (Hayden Book Co. Inc, Rochelle Park, New Jersey)

Contains explanations of many ciphers, with programs in BASIC to implement them or act as aids in defeating them. The programs may require some work to implement (you have to search through the book to find the subroutines, and sometimes the names of variables change), but some good material is included. The programs are in a simple version of BASIC which most computers should handle as is or with only minor changes.

"Securing Data Inexpensively via Public Keys" by Brian Schanning (Computer Design, April 5 1983, Vol 22 #4)

is an article which describes the mathematics used to generate the two halves of a Public Key.

"The Data Encryption Standard, Recent Controversies" by John E. Hersey, (Telecommunications, Sept. 1983, Vol 17 #9)

gives an encapsulated history of the development of the DES, with some of the arguments for and against it's method of implementation and use.

I have not been able to review the following sources myself, but they may be useful.

"The Codebreakers" by David Kahn (Macmillan)

gives a good history of ciphers and their use, and a description of how some good modern ciphers were broken. The paperback

version may be abridged. Considered one of the classic works on the subject.

"RSA: A Public Key Cryptograph System" by C. E. Burton, (Dr. Dobb's Journal, Mar 1984, 16-21)

"Mathematical Games" by M. Gardner, (Scientific American, 237(2), August 1977, 120-124

The following government publications may also be useful:

"Data Encryption Standard"  
Federal Information Processing Standards  
Publication 46

"DES Modes of Operation"  
Federal Information Processing Standards  
Publication 81

Standards Information Office  
Institute for Computer Sciences  
and Technology  
National Bureau of Standards  
Washington, D.C. 20234

The Smithsonian Institution has a section devoted to cipher machines, and give the following address for inquiries for more information on the subject:

Division of Mathematics  
The National Museum of American History  
Smithsonian Institution  
Washington, D.C. 20560



Theodore J. Smith

Department of Radiation Therapy  
Hospital of the University of Pennsylvania  
Philadelphia, Pennsylvania 19104

Jill M. Baren and Robert F. Curley  
Department of Radiation Therapy  
University of Pennsylvania  
Philadelphia, Pennsylvania 19104

### Abstract

The Department of Radiation Therapy at the Hospital of the University of Pennsylvania sees and treats a large number of patients each year. We have implemented a system to manage the data relating to these patients. We used DBMS-11 running IAS on a PDP-11/70 with FORTRAN as the principal programming language. This paper presents our design philosophy, programming conventions and the measured results of the operation of the data entry and scheduling modules.

### DBMS-11 Overview

#### Introduction

The Radiation Therapy Department at the Hospital of the University of Pennsylvania sees approximately 120 patients per day. An enormous amount of information concerning the treatment and handling of these patients is necessary not only for daily use by medical and office personnel, but also to establish a permanent record for long term research and statistical functions performed by the department. Our primary goal for a computer based patient information system is to provide quick, easy and nondestructive access to accurate patient demographic and treatment information. Our system was planned to eliminate the aggravation of locating a patient's chart {1}, by displaying pertinent information at terminals located throughout the department when requested. Although we have reached only an intermediate stage of this goal, we have expanded the functions of the database to include the automation of several clerical tasks and are currently incorporating a patient tracking system into the scheme. The foundation of a patient information management system consisting of data on 15,700 patients has been established in our department using DBMS-11 version 2.1, a database management system for the Digital Equipment Corporation PDP-11/70 computer running IAS {2}. DBMS-11 is a network structured database which enables the design of a data model that closely reflects the way information is collected and used by the Department. From this foundation, many department functions have become automated while others are currently being developed.

DBMS-11 is a implementation of a CODASYL {3} database stressing the logical relationships among records and the ability to access only related information using sets. Our CODASYL database is a direct access file divided into pages of 1024 bytes each. Information is organized into records each having a unique address (DBKEY). The DBKEY is the combination of the page and line number of the page where the record is stored. For example, the fifth record on page 4321 has the DBKEY 4321:5. Sets are used to represent the logical relationships between records. A data dictionary is created to contain all record and set definitions describing the database, using a COBOL like data definition language.

Example of a Data Dictionary record discription:

Record Name is DEMO.  
Record Id is 203.  
Location mode is VIA IDSET SET.  
Within HUPDRT Area.

|    |         |            |
|----|---------|------------|
| 05 | SEX     | PIC X.     |
| 05 | RACE    | PIC X.     |
| 05 | BORN    | PIC X(4).  |
| 05 | ADDRES. |            |
| 10 | STREET  | PIC X(20). |
| 10 | CITY    | PIC X(15). |
| 10 | STATE   | PIC XX.    |
| 10 | ZIP     | PIC X(10). |
| 05 | PHONE   | PIC X(10). |

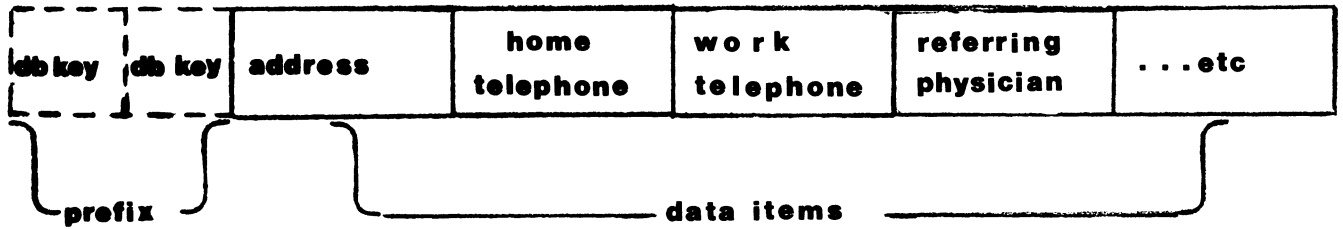
#### Records

A record is a named group of data items



fig.1

**RECORD**



stored together, such as DEMO (Figure 1) which consists of sex, address and phone number, telephone number, etc. Reference to a record implies a reference to all the data items in the record. Each DBMS-11 record also contains a prefix maintained by the Database Manager (DBM) task. The prefix contains one to three DBKEYs for each set in which the record participates. Records are stored in the database in either CALC or VIA location modes.

CALC location mode calculates (hashes) the DBKEY of the record based on the contents of a data item in the record. If two records have the same calculated address, an overflow algorithm is used by the DBM to resolve the conflict. Storing records with the CALC location mode will evenly distribute the records throughout the database in an ideal world. To locate the CALC record, the user supplies the value of the CALCed data item and the address is calculated by the DBM. For example, Social Security numbers are the CALCed data item for the Social Security record SSNRCD. To

locate the patients with missing numbers, the user enters "000-00-0000". The DBKEY of the SSNRCD record is computed and the record located. The patients in the Social Security set are then listed.

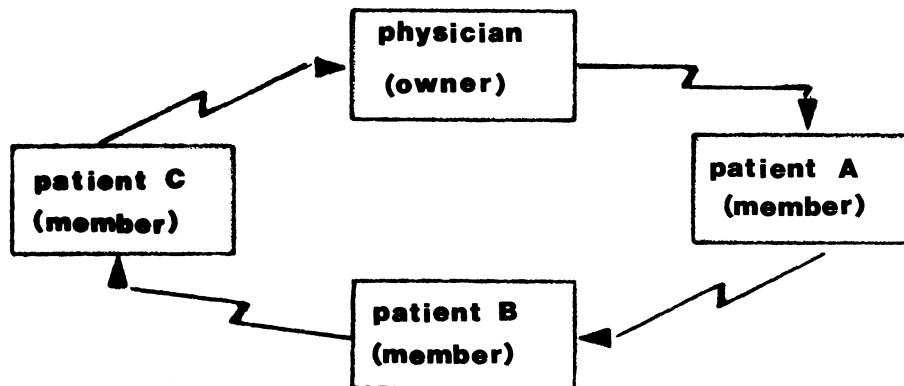
VIA location mode stores the members of a set physically close to the owner. This has the effect of grouping the members of the set onto a few pages. This reduces the number of pages read and the time to access members of the set. VIA is best used for small records in sets in which many members are accessed frequently.

Sets

A set is a named group of records with one designated owner, having zero or more members. Sets are stored as linked lists of DBKEYs connecting the owner and member records. A record may participate as either owner or member of many sets, however, a record cannot be both member and

fig.2

**SET**



owner of one set. Sets provide fast access to information related to the owner of the set by using DBKEYs to point to the next member of the set. Each record prefix contains the DBKEY of the next record in the set and may also have the DBKEYs for the prior and owner records of the set. The owner prefix contains the DBKEY of the first member in the set while the last member prefix contains the DBKEY of the owner. For example, the patients (members) treated by a certain physician (owner), illustrate the retrieval speed provided by the set structure (Figure 2).

#### Example of Set definition in Data Dictionary:

```
Set name is STFSUM.
Set ID is 323.
Order is Next.
Mode is Chain linked to Prior.
Owner is STFDOC
 Next DBKEY position is 3
 Prior DBKEY position is 4.
Member is DEMO
 Optional Automatic
 Next DBKEY position is 9
 Prior DBKEY position is 10
 Linked to Owner
 Owner DBKEY position is 11.
```

#### Currency

Most database operations are relative to the user's "current" location in a set. Operations such as "find next" or "find prior" member require knowing which member of the set the user had last accessed in the set. The DBM maintains a group of currency indicators for each user accessing the database. A currency indicator is the DBKEY of the most recently accessed record in the database. There is also a currency indicator for each type of record and set in the database. Currency indicators point to the user's position in sets and the database, and are updated only by the DBM.

#### Data Dictionary

The Data Dictionary contains all record and set definitions for a CODASYL database. It is a separate entity from the database. As the English dictionary provides definitions for the words of English, the data dictionary provides definitions for the records and sets of the database. This eliminates the overhead and possible errors caused by maintaining data definitions in each program accessing the database.

#### Database Manager

No user has direct access to the database. Instead, the Database Manager task is a program which controls what information is

transferred between a user and the database. The Database Manager handles simultaneous requests based on two principles, access modes and locking. "Access modes" inform the DBM of the user's intention to update information. Locking limits concurrent access by other users. Each database program declares itself to require either Update or Retrieval modes. Update mode access allows a user to change information and Retrieval mode allows read only access to the database. Locking can specify either Concurrent, Protected or Exclusive access by other users. Concurrent locking allows other users to read or update the database while the "concurrent" task is still connected. Protected locking allows only one user to update while all other users can read from the database. Exclusive access locks out all other users until the current user is finished. For any task on the computer to have access to the database through the DBM, a separate "sub-schema" task is required. This feature limits the data that be read by a given user. Access to sensitive information can be restricted in this manner.

#### Communicating with DBMS-11

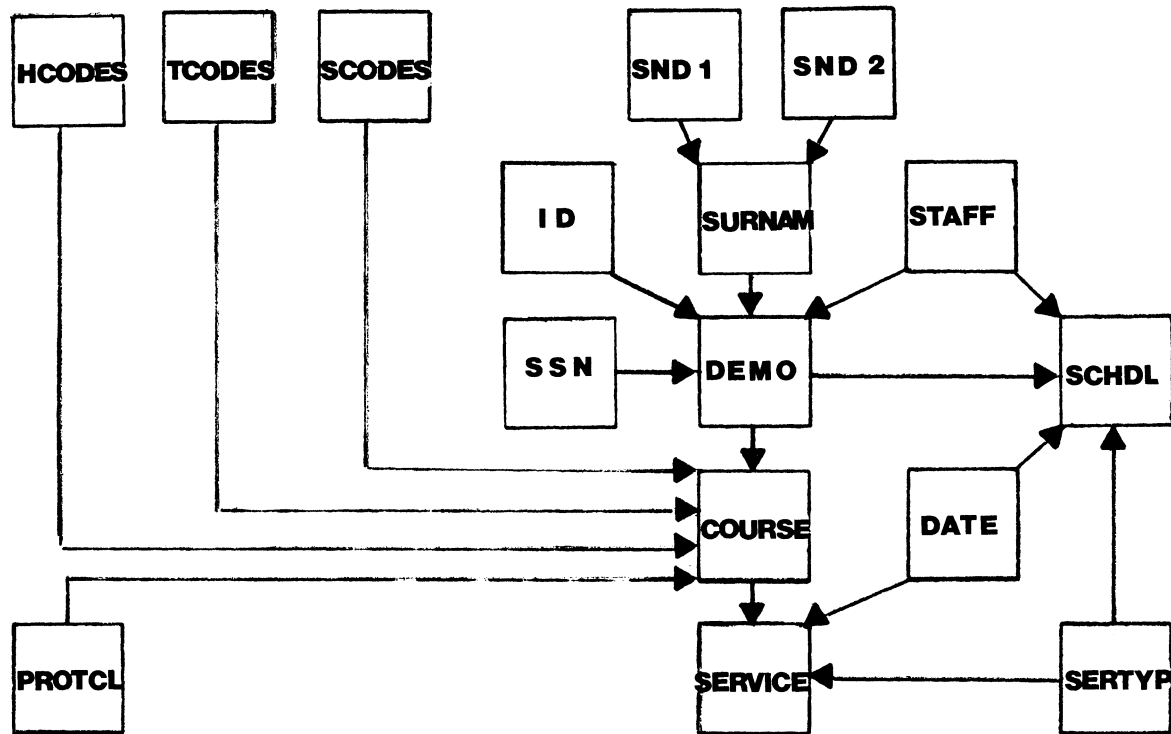
DBMS-11 establishes a User Work Area (UWA) [1] in each database program. In FORTRAN, the UWA is a COMMON block used to transfer information between the database and program. Only the data portion of each database record is transferred into the UWA by the DBM. This prevents corruption of the record prefix by any program ensuring the integrity of the set structure. Database programs need not be connected to the DBM when loading information into the UWA. After all information is collected and placed into the UWA, the DBM is then accessed in Update mode to store the information. When modifying information, the DBM is accessed in Retrieval mode to fetch the information into the UWA. After modification, the DBM is accessed in Update mode to rewrite the modified information. This method reduces the risk of corruption from users aborts by minimizing the actual connection time to the database.

#### Design Overview

Our database is designed to retrieve information quickly, by using sets as fast paths to locate related information (Figure 3). Currently, the design is comprised of four regions each containing a different type of patient information: Demographic, Treatment Course, Provided Service and Scheduling. Frequently used information unique to the patient is placed in the Demographic record. In addition, the demographic record contains pointers to external storage files containing information rarely accessed. Course

fig.3

**OVERALL DESIGN SCHEME**



records contain information that remains constant throughout a course of radiotherapy. All information about a service performed by the department is placed in a service record. Scheduling records are used to schedule patients for the services provided by the department.

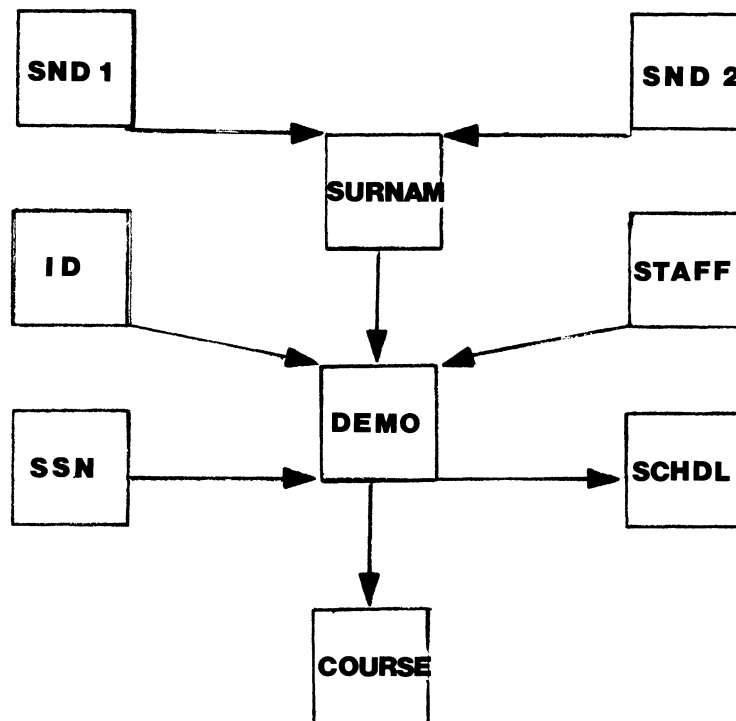
**Demographic Information**

Each patient stored in the database has only one demographic record. Sets for medical history number or hospital ID number {4}, Social Security number, surname and patient's physician provide quick access to the Demographic record (Figure 4). All Treatment Course and Scheduling information is owned by the patient's Demographic record. A user enters name, history number or Social Security number to locate a patient (make the patient's record "current"). If more than one patient is found with the same surname, then the list of matching patients is displayed for selection to the user. Upon selection, the Demographic record DBKEY is remembered so that the application can directly access the patient without further prompting the user.

Data items in the Demographic region:

- o Social Security Number
- o Medical History Number or ID
- o Patient's surname
- o Patient's given name
- o Sex
- o Race
- o Status (Inpatient, Outpatient)
- o Birthdate
- o Address
- o Home and Work telephone
- o Referring physician
- o Betabase access pointer
- o Gammabase access pointer

Locating a patient by name need not be exact. There are two phonetic methods for locating patients with surnames having



similar spelling or pronunciation. Soundex [2] is a method of translating similar names into the same string value. Using a set, all names (members) having the same value (owner) can be quickly listed for selection. The Soundex method which produces a value based on similar spelling, emphasizing the first letter in the surname is used first. This method assumes the user correctly entered the first character of the surname. The Wallace method [3] of pronunciation is used when Soundex fails to locate the patient. The Wallace coding method provides the mechanism to locate the patient using only a vague or phonetic spelling of the surname by encoding the entire surname.

The procedures used to locate a patient until selection:

1. Location by patient name:

- a. User enters best guess of patient name.
- b. If only one patient is found exactly matching user's guess, then user is asked to confirm selection.
- c. All patients with the surname which matches the guess are listed.

d. Patients having the same Soundex value are listed.

e. Patients having the same Wallace value are listed.

f. If no patient is selected, user is permitted to add the patient.

g. The selection procedure will only list patients once. If a patient has the same Soundex and Wallace value as the user's guess, then the patient will be listed with the Soundex matches but not with the Wallace matches.

2. Location by ID or Social Security number:

a. User enters ID or Social Security number.

b. If only one patient is found, then the patient is selected.

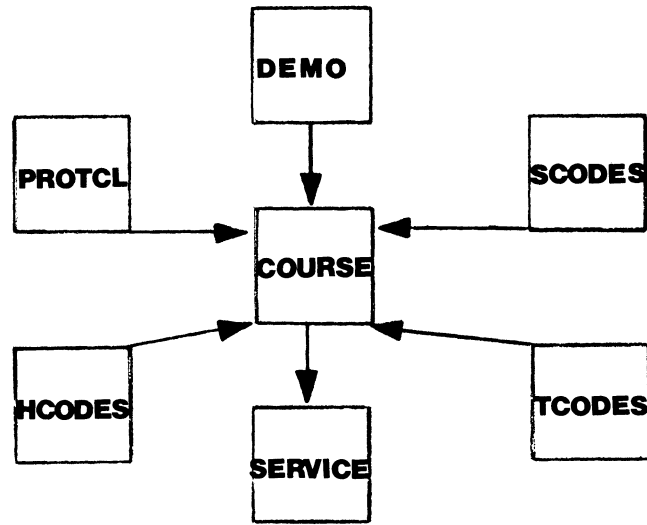
c. All matching patients are listed.

d. If no patient is selected, user is asked for the name of patient.

e. Location by name is attempted.

fig.5

**COURSE DESIGN**



Treatment Course

One Treatment Course record is stored for each course of radiotherapy the patient receives. Information that is constant during the Treatment Course is stored here. The Course record is found through the Demographic, Primary Site, Treated Site, Histology and Protocol sets (Figure 5). The Primary, Treated and Histology sets are coded according to the SNOMED {5} scheme. For each service provided during a course of radiotherapy, a service record is placed in the Service set owned by the Course record.

Data items in the Treatment Course region:

- o Tumor Grade
- o Recurrent Disease
- o Tumor staging
- o Treatment Plan
- o Tumor Therapies
- o Radiotherapies

fig.6

**SERVICE DESIGN**

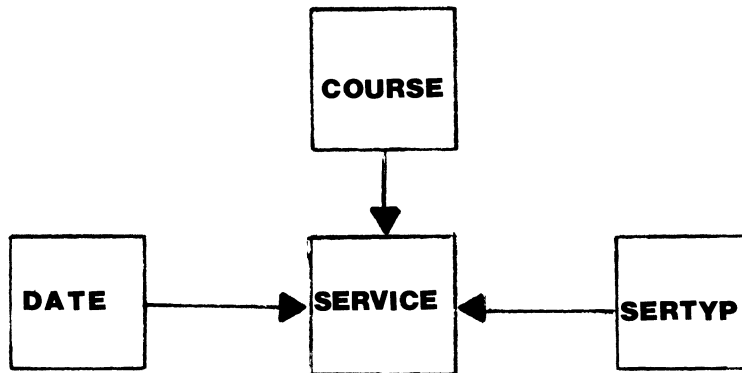
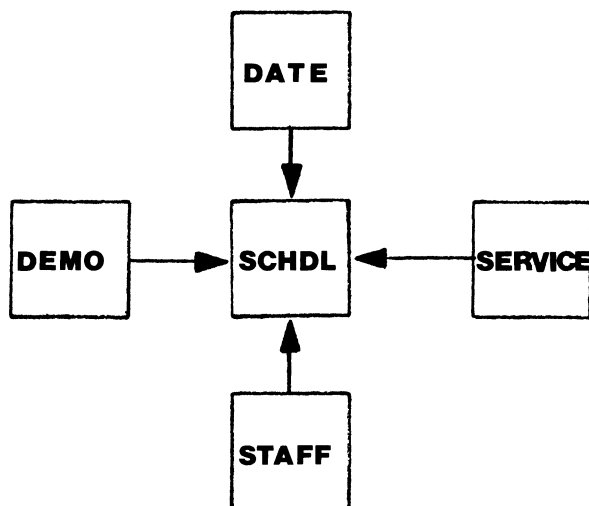


fig.7

# SCHEDULING



### Provided Services

A Service record is created for each service the department provided to a patient. The Service record can be found by either the Course, Date or Service Type sets (Figure 6). All information specific to the service is stored in the record.

Data items in the Service region:

- o Type of Service (ex. Treatment on Linear Accelerator 1)
- o Service Date
- o Delivered Dose

- o Ports

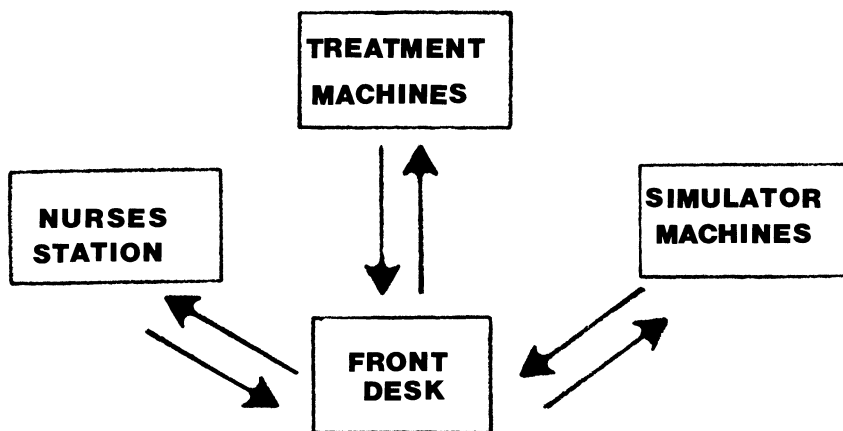
- o Special Devices

### Scheduling

Scheduling of services is a major function of our system and is performed daily. A schedule record containing the time and appointment status is sorted by time in the date set. Schedule entries can be organized by date, patient name, physician, and service type (Figure 7). A service record is created after the service has been scheduled and completed. No service record is created for scheduled events that

fig.8

# TRACKING



did not occur. Examples would be patient cancellation of an appointment or a missed day of treatment. Many schedule formats are used throughout the department and are designed according to the needs of personnel. Printing the various of schedule formats occurs by unloading the schedule records into a file using SORT-11 to sort each schedule.

### Tracking

A system developed to track patients through the department will soon be installed (Figure 8). Each service station and the receptionist will receive a VT220 terminal to interact with the tracking system. The receptionist will inform the system when the patient arrives in the department. The system then displays the patient's name on each of the scheduled service station terminals. Assume a patient arrives for both treatment and a visit with a physician (two different services). His name will appear on both the treatment machine and nurse's station terminals. If treatment is the first available service, then the patient's name will be marked unavailable at the nurse's station. Completion of treatment is indicated by the technician at the treatment machine who describes the treatment to the computer. Now the patient is marked available at the nurse's station to be selected for the physician's examination. When the technician enters information about the service provided, the system creates a service record in the database and generates a billing charge based on the service information entered. The clinic will at all times be aware of the patient's location and progress in his scheduled services. Tracking will account for services provided to the patient both financially and statistically. Additional advantages of monitoring the patients' movement through the department are the reduction in patient waiting time between multiple services and clinic personnel time spent locating patients.

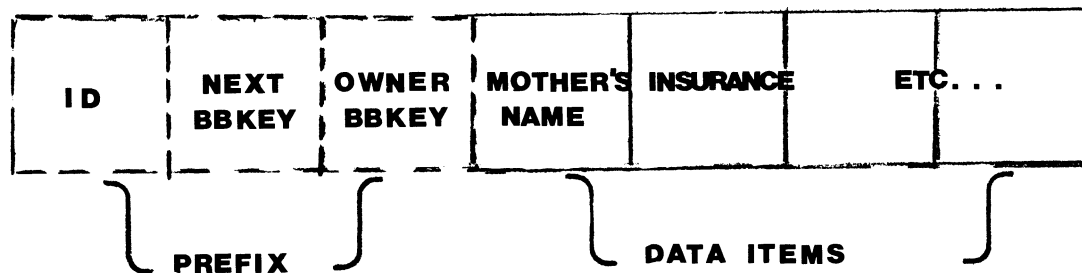
### Betabase

The "Betabase" is a reservoir for infrequently used information. It is also used to contain new data items until the database can be upgraded. Quick access to the Betabase is provided by storing the starting Betabase record address (BBKEY) in the patient's demographic record as the Betabase access pointer. The Betabase is a direct access file of 1024 byte fixed length records. The first record in the Betabase file (Betabase Control Record) is reserved to contain the number of allocated and used records in the file. Allocation size is the number of records reserved for the Betabase by the IAS operating system. Used size is the number of records assigned to contain patient information. Space available for assignment is found by subtracting the Used from Allocation sizes in the Betabase Control Record. The Control record enables automatic expansion and valid BBKEY range checking. All other records are divided into a 24 byte prefix and 1000 byte data area (Figure 9). The prefix contains the patient's history number, owner, and next BBKEY in the patient's set. The data area contains three part data items. A data item is composed of a data type, data, and an end of data marker. A null byte following the end of data marker indicates the end of Betabase information for the patient.

Data items that may appear in a Betabase record:

- o Mother's given name
- o Father's given name
- o Names and addresses of relatives
- o Alternate referring physicians
- o Occupation
- o Insurance information
- o Birthplace

**fig.9 BETABASE RECORD**



## Adding to the Betabase

When adding information to the Betabase, a record (1024 bytes) must be assigned to the patient. Prior to assignment, the Betabase control record is checked for the next available record. If full, the Betabase expands and initializes the expanded area. If this is the patient's first Betabase record, the address is written into the Demographic record of the database. The assigned record prefix is updated to contain the history number and the owner BBKEY.

Each new data item is appended to the end of the last data item stored in the record. If a data item is too large to fit on the Betabase record, then the record is filled with the data item. Another record is assigned to the patient and its BBKEY is placed into the prefix of the filled record before writing the filled back into the Betabase. The data which did not fit onto the filled record is written on the current record. When addition is complete, the current record is written back into the Betabase.

## Modifying a Betabase record

The user selects which data item(s) to modify and enters the modification. Modification of the Betabase record is based on the difference between the lengths of the data item and modification:

- a. If the modification is smaller than the existing data item, the modification overwrites the data item. All data items following the modification are compressed to the end of the modification. A record is deleted if empty after modification.
- b. If the modification is same length as the existing data item, the data item is overwritten by the modification.
- c. If the modification is larger than the existing data item, the data items following the selected data item are moved to make room for the larger modification.

## Deleting a Betabase record

The Betabase is always compressed with all assigned records at the beginning of the Betabase. When a record is deleted this procedure is followed to ensure that empty records are placed at the end of the Betabase:

- a. If the deleted record is the owner BBKEY, then the Betabase access pointer is erased from the demographic record in the database.
- b. The last record assigned in the Betabase is copied onto the deleted record.
- c. The Used Indicator of the Betabase Control record is decremented by 1.
- d. If the moved record is not an owner record, then the Betabase Access pointer is updated to the new BBKEY address.
- e. If the transferred record is an owner, then the owner is located and the set is searched until the record previous to the transferred record is found. The next BBkey is updated to the new address of the transferred record.

Deletion of Betabase records rarely occurs and no patients to date have more than one Betabase record.

## Gammabase

The "Gammabase" is another non DBMS-11 file structure that provides online storage and retrieval of patient chart free text documents. Thus, a physician can review a patient's treatment history without removing the chart from the fileroom. The Gammabase is a direct access file of 1024 byte fixed length records. The first record in the Gammabase is reserved as a Control record indicating the allocation and used size in the same manner as the Betabase. The Gammabase contains two groups of records: Index and Document. Index records contain the starting record address (GBKEY) of all the patient's Gammabase documents. Document records contain the actual document text. All Gammabase records have a 32 byte prefix containing Document record type, Medical History Number and Social Security number. Document record prefixes also contain prior and next GBKEYs to the preceding and following records of the document.

### Chart Documents in the Gammabase:

- o History & Physical: Contains information about initial consultation.
- o On Treatment Visit: Impressions of routine examinations during treatment.
- o Completion Summary: Impressions at the end of the course of radiotherapy.



fig.10 **GAMMABASE INDEX RECORD**

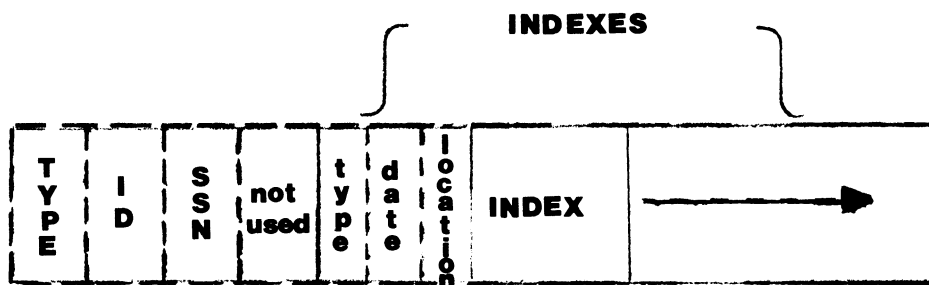
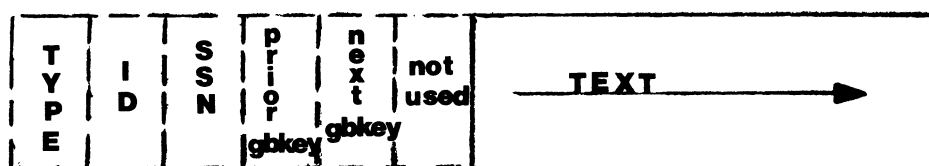


fig.11 **GAMMABASE DOCUMENT RECORD**



- o Follow up: Impressions of post treatment examinations.
- o Reconsultation: Information about a reconsultation examination.
- o Correspondence: All other chart information such as letters to referring physicians.

document format and only removes formfeeds from a document for better video display.

Date Processing

Major functions such as scheduling are based on date. We wished to design a flexible date processor to parse the most common date formats. Forms such as a day of the week or tomorrow are relative to the current system date. Most dates have one of three basic formats dependent upon the number of tokens given by the user:

Index records

Each patient with Gammabase documents has only one Index record (Figure 10). The Index record address is stored as the Gammabase access pointer with the patient's demographic information in the database permitting direct access to the Gammabase. The Index record contains nine byte index pointers to all the patient's documents stored in the Gammabase. This permits a patient to have 110 documents in the Gammabase. An index pointer contains the document type, date and starting address (GBKEY) for a document.

Document records

Document size is unlimited and blocked into records of 990 bytes of text (Figure 11). Documents are entered onto the Digital Equipment Corporation WPS-200 word processor. From here they are loaded into the Gammabase. The Gammabase permits any

1. Single token format:

- a. A Numeric token is translated as the day of the current month.
- b. An alphabetic token can be either a day of the week or a relative date such as Yesterday, Today or Tomorrow. Relative dates are based on the current system date.

2. Two token format:

- a. If both tokens are alphabetic, then the first token must be either Next or Last. The second token is a day of the week such as Next Monday, also relative to the current system date.

b. If both tokens are numeric, then the first token is the month of the current year and then second token is the day of month, such as 5/1.

second tokens are processed according to the Two token format.

c. Regardless of order, the alphabetic token is always the month and the numeric token is the day of month such as May 15 or 5 December.

The tokens can be separated by the following characters: hyphen "-", slash "/", period ".", comma ",", asterisk "\*" or spaces. No multiple separators are allowed, except spaces. A separator with spaces can delimit tokens.

3. Three token format:

a. The third token is always the year. If the year is less than 100, then year is in the current century. First and

Once a date has been parsed, it is encoded into a four byte code to save space. When using a date interval, the end-dates of the interval are converted into Julian numbers [4]. A simple loop can then be used to span the interval.

Examples of valid dates:  
(current date is Tuesday, 9 April 1985)

| Example      | Date                     |
|--------------|--------------------------|
| 15           | Monday, 15 April 1985    |
| Yester       | Monday, 8 April 1985     |
| Today        | Tuesday, 9 April 1985    |
| Tom          | Wednesday, 10 April 1985 |
| Thur         | Thursday, 11 April 1985  |
| Next Fri     | Friday, 12 April 1985    |
| Last Sunday  | Sunday, 7 April 1985     |
| 4 May        | Saturday, 4 May 1985     |
| March 15     | Friday, 15 March 1985    |
| 6/5          | Thursday, 5 June 1985    |
| 4-30-85      | Tuesday, 30 April 1985   |
| 3.11.1985    | Monday, 11 March 1985    |
| May 28, 1895 | Tuesday, 28 May 1895     |
| 4 July 1776  | Thursday, 4 July 1776    |

Privilege Masking:

mask. If the bit is not set, the user's access is denied by the database program.

Both IAS and DBMS-11 provide methods which limit access to the database, particularly important for maintaining the integrity of sensitive information. IAS provides the ability to grant read and write access to the database directory. DBMS-11 maintains a list of valid users and where to direct them when accessing the database. This is referred to as a Redirection Table. The XEQ {6} facility, a program which invokes a task outside the user's directory according to a predefined path, is used to access the database. Only valid users have an XEQ path to the database. Finally, each user has a 32 bit privilege mask. Each database program is assigned a bit in the privilege

Performance Evaluation

Location of a patient in the database is the most critical function. Therefore, performance of the system can be measured as the time used to accurately locate a patient. During a period of six weeks, location times were recorded using the SECNDS {7} function. Location time was measured from the entry of a name or ID until a patient was selected by the location procedures for name and ID. Location time includes the time to display lists of patients and waiting for the user's response over 1200 BAUD lines.

Performance of Location Procedures:  
(Time is measured in seconds)

|                | Name   | ID     |
|----------------|--------|--------|
| Total Measured | 2704   | 9058   |
| # Under 2 sec. | 748    | 8138   |
| Minimum        | 0.62   | 0.49   |
| Maximum        | 467.30 | 307.07 |
| Range          | 466.68 | 306.58 |
| Median         | 8.83   | 0.92   |
| Mean           | 19.41  | 1.37   |
| Standard Dev.  | 34.13  | 4.06   |

The ID location procedure was found to be the more frequently used method to locate a patient by our users. Since each patient is assigned a unique ID by the Medical Records Department, the ID is the most direct path to the patient. The mean and median ID location times are under the two second acceptable standard that was established as a design goal three years ago. If no patient is found by ID, the name location procedure is invoked automatically. Location time by name reflects the time used to display lists of patients having similiar names and waiting for the user to select a patient. For this reason, location times by name are very skewed as indicated by the standard deviation of location time. The median time to locate a patient by name is reasonable because of the tremendous variation in spelling among users.

#### Constraints and Problems using DBMS-11

Like all other systems, DBMS-11 does have constraints and problems arising from either the design or support of the system. We have highlighted some of the more important ones below:

##### Constraints:

- o Database programs cannot exceed 28K (8) words of memory. A 4K word buffer is used by DBMS-11 to communicate and transfer information with the database.
- o A CODASYL database maintains a rigid design. As the Department evolves, changes to the design of the database are inevitable. When the database design is upgraded, all information in the database must be unloaded and reloaded after the upgrade. All database programs must be rebuilt, although no coding changes may be necessary.
- o DBMS-11 provides no mechanism to unload information. You must design a file structure to preserve the logical relationships among records and write a program to unload and reload the database.

##### Problems:

- o DBMS-11 is no longer supported by Digital Equipment Corporation.
- o The DBKEY computed for CALC records is based on the number of pages in the database. Hence, when the database is full, you must unload all information so that the DBKEY can be recalculated when reloading. Recently,

#### Conclusion

Several areas of the department have been significantly affected by the computerized database system since its installation in January 1983. Many more employees are now able to obtain the data they need on an individually determined time scale rather than having to wait their turn for a glimpse at the patient chart. In addition, the automatic printing of office materials containing patient information (index cards, front sheets of charts, schedules) has considerably lessened the amount of repetitive typing of the same data. Finally, the flexibility of the database design has allowed practically any request for a list or summary of data to be tailored to the demands of the individual employee. It is our intention that the future functions of the database system, primarily the patient tracking and billing systems, will continue in this vein by contributing to the improvement of Radiation Therapy operations.

#### Footnotes

{1} "The patient chart" is a paper and cardboard record of the patient's history of encounters with our department. It contains physicians notes, treatment prescription and records as well as the place where all laboratory and Radiology reports are filed.

{2} "IAS", Interactive Application System. An operating system for the PDP-11/70.

{3} "CODASYL", Conference on Data Systems Languages.

{4} "ID", Patient identification assigned by the Department of Medical Records to uniquely distinguish patients.

{5} "SNOMED", Systematized Nomenclature of Medicine [5]. A coding scheme developed by the American College of Pathology and selected by the Department to encode disease.

{6} "XEQ", William Wood, "Computer Program XEQ", The Best of ICR Collection, Institute for Cancer Research, Philadelphia, PA, 1981.

{7} "SECNDS", A function supplied with Digital FORTRAN 77 which computes elapsed time.

{8} "K", A constant equivalent to 1,024.

#### References

1. Digital Equipment Corporation, DBMS-11 Database Administrator's (DBA) Guide, December 1981.
2. "SOUNDEX Foolproof Filing System for finding name in the File", Remington Rand, Brochure LVB801.
3. Curley, Robert F. and Smith, Theodore J., "A Radiotherapy Department Computer Database", Proceedings of the Eighth International Conference on the Use of Computers in Radiation Therapy, July 9-12, 1984, pp. 501-505.
4. "Algorithm 199", Collected Algorithms from ACM, Vol 1 Algorithm 1-220, ACM Inc., 1980, p. 199-P-1.
5. Systematized Nomenclature of Medicine, American College of Pathology, April 1979, 2nd Ed.



# Criteria for Selecting Your Relational Database

Jeffrey S. Jalbert

and

Keith W. Hare

JCC

Box 381

Granville, Ohio 43023

(614)587-0157

Now that you have decided to use a relational database, how do you decide which relational database to use?

This paper describes the features you should expect to find in a fully functional relational database. These features are divided into the following categories:

- General software features
- Security
- Database consistency
- Data integrity
- Performance
- Data dictionary
- AD-HOC query language
- Programming language interface
- Forms interface
- Miscellaneous considerations

Each of these major categories is defined and their important sub-points are noted.

## 1 INTRODUCTION

The term database is one that is often mis-used. In this document we will use the term in its true technical meaning, as recognized in the computer science community. The reasons for wanting a database system are many and compelling. The reasons for using the relational model are just as compelling. Neither of these issues are addressed here.

A relational database (as defined in C. J. Date) is one that has the following characteristics:

1. Relations and associated data structures
2. Query language at least as powerful as relational calculus (such as relational algebra).
3. Does not use funny loops
4. Supports unique keys, foreign keys

The above criteria for being a relational database do not address all of the issues that must be addressed if a relational database is to be a useful tool in a production environment.

In this presentation, we will outline the tool we used to compare several commercial (VAX based)

relational database products. We will display a description of the analysis process and an outline and discussion of those features of a relational database package that we think are useful/helpful/required in a production environment.

## 2 ANALYSIS PROCESS

Judging software is just not a clean process. Competing packages often have different characteristics, ones that are valuable. One then is often in the process of trading off one valuable characteristic for another. In order to be objective in ones choice, the following approach is often used.

### 2.1 Scoring Method

First, one gathers together all the issues that one can think of that one may want in a system. This effort must take into account what is available from different vendors, what is desirable in the target environment, what is practical, what is desired, even if not immediately available, what is strategically appropriate in terms of evolving technology, and just about everything else including the kitchen sink.

These points are then described in more-or-less detail and also organized into sub-groups of issues and items. Each sub-group is awarded a number of points, the total for all sub-groups often being 100. This total really is arbitrary, but the issue here is that the relative weight of each sub-group must be assessed.

There may be some issues that are of over-riding importance. These are go/no-go issues. Really, these act almost as a veto. For instance, if two packages are relatively equal otherwise, but one has the potential to grow into something that is even more desirable, we might not select that package because its performance is just too poor. Certainly performance is one of those go/no-go issues, but only if performance is dramatically different. A ten-percent performance difference in some test may just be a fluke of the test and not to be valid across all uses.

In any event, once the major categories have been formed and their relative importance assessed, specific sub-issues are then assigned to them. These issues are also assigned points which should total the points for the entire sub-group. Once this is done one is ready to evaluate the competing products.

A committee is formed which ranks each product in each category of importance. Points are assigned to the product from zero to the maximum for that category depending on the quality with which the product meets the requirements for that particular category.

Analysis is then simple, total the points, address go/no-go issues and the winner is often obvious. If more than one product receives a similar score then other tools should be brought to bear on the analysis, such as past experience, hunches, "feel", whatever. But, at least this method allows one to more-or-less make the selection process more rigorous.

## 2.2 Garland's Method

In an article titled "Hiring your Database System" in the Fall 1983 DECUS Proceedings, Andrew M. Garland outlines an alternate to the above scoring method.

After discussing the scoring system above, Garland says:

"In this approach, the problem of satisfying all interests is solved by buying the "best" system. Everyone is consulted to approve the list of features and the weightings. The approach is scientific, fair, and no one can object to the result. Essentially this replaces judgement by methodology.

"The difficulty with this approach 'by list' is the assumption that the complexity of a DBMS can be captured on paper, and that the decision can then be made by what is on paper. People are quite good at sifting data and valuing options provided that they use aids such as lists to help their memory. The lists are also useful to others who wish to add to the evaluation. However, the lists are only aids.

The paper cannot make a good decision; it can only record some of the input to a good decision. If the information is restricted from the start to fit onto paper, a bad choice is more likely.

Garland then goes on to recommend the following procedure:

1. Inform yourself about DBMS systems
2. Read the manuals for the candidate DBMS's
3. Get a presentation of the system
4. Try the systems, all of them, on real projects
5. Talk to other users of candidate DBMSes
6. Remember that performance comparisons may be highly mis-leading because of the different ways DBMSes do their work
7. Remember to consider the expert vs. novice user in your analysis.

## 2.3 Our Analysis

We have applied a modified version of Garland's analysis. Each of the products was investigated, both by having local presentations and by talking with other users of the products and by obtaining "subjective" comments from the national VAX community. We also itemized features that we feel are important in a database and tallied which features were present in each product. The table we used for our analysis is included in this report.

We do not include the final results of our analysis. It is not our intent to publicly rate database management systems but rather to present methodology.

## 3 MAJOR CATEGORIES

This section outlines those categories that are of major importance in selecting a database system. These are listed below and explained more fully in separate paragraphs that more completely outline all the finer points that we consider in our analysis.

The categories we determined were:

- I. General Software Features
- II. Security
- III. Database Consistency
- IV. Data Integrity
- V. Performance
- VI. Data Dictionary support (external to Database)
- VII. Quality of AD-HOC Query Language

VIII. Quality of programming language interface

IX. Forms interface

X. Miscellaneous considerations

The meaning of each of these categories is explained below, together with the sub-points that are of importance in each of these major categories.

### 3.1 General Software Features

1. Many users must be able to access any table simultaneously (for update)
2. Datatypes should include, strings, dates, text, fixed and floating point numerics. Fixed and variable length strings should have no "practical" size limits. Funny datatypes such as segmented strings would be useful.
3. Data format independence (can dynamically change size/type of fields)
4. Ability to define tables on the fly (from a program)
5. Ability to define indices on the fly (from a program)
6. Ability to have multiple cursors into a single database simultaneously
7. Ability to access multiple databases simultaneously (single process)
8. Ability to modify table definitions without having to re-load database or table.
9. Unlimited number of columns in a relation
10. Unlimited number of relations in a database
11. Ability to define views
12. Ability to define views composed of both views and relations
13. Concurrent Batch/on-line
14. Record sizes should be unlimited
15. Inner Join
16. Outer join
17. No limit to the number of tables that may be joined at one time
18. Explicit control of start and end of transaction
19. Rollback and Commit
20. Aggregation functions including TOTAL, COUNT, MAX, MIN, AVERAGE and PROJECTION to unique values

21. DDL and DML that adheres, more or less, to the standard relational database syntax (as defined by relational calculus). Consistency of syntax would be useful.

22. DML this is fully functional and accessible from program environment.

### 3.2 Security

Security means the protection of resources (data, software and hardware) from accidental or malicious damage, from disclosure to unauthorized individuals, from unauthorized modification and includes preventing unauthorized users from denying access to authorized users.

Features that we feel are important to have in a database system to support security are:

1. Control access to relations/views on a per-user basis with some kind of pass-through access on views of views
2. Control access to database on a per-user basis
3. Data may not be modified (if desired) external to the programming language environment (where such modification is under prescriptive control)
4. Audit trail facility that can be turned on and off by data base administrator

### 3.3 Database Consistency

A database is in a correct state if it contains the most recent entries and modifications made by any user and it does not contain any information that has been deleted by any user. A database is in a valid state if its information is part of the information in a correct state. This implies that there are no spurious data, although some information may have been lost. A database is in a consistent state if it is in a valid state, and the information it holds satisfies the users' consistency constraints.

Features that we feel are important to have in a database system to support consistency are:

1. Journaling that can be turned on and off by the data base administrator
2. Roll-back incomplete transactions
3. Roll-forward from journal file
4. Automatic recovery in the case of failure
5. Utility to monitor the integrity of the database
6. Quality of integrity of system both to normal update procedures and when a hardware failure (e.g. head-crash) occurs.



7. Deadlock detection, rollback of victim and automatic restart of victim transaction
8. Two-phase locking (no lock may be requested after unlock has taken place)
9. Two-phase commit
10. Database Backup (speedy and efficient) with support for incremental backup/restore.
11. Maturity of software package (so that bugs will have been eliminated)

### 3.4 Data Integrity

Data integrity as used here means that the data stored in the database meets the standards necessary to support the application. We include here such things as having only correct values in data fields, cross-checks of data between different records, and in general, everything that one might require to insure that the data actually stored in the records meets the quality standards demanded by an application.

Heretofore such checking has been performed by programs. Fourth-generation software should have such checking performed by the database itself, and the required standards set by the database design.

Further we should include the notion of triggers in our dialogue so that host-language programs do not have to be written to support semantic knowledge of the database.

Features that we feel are important to have in a database system to support data integrity are:

1. Data verification/validity checks/edits
2. Constraints
3. Ability to check constraints on commit with all updates simultaneously available.
4. Range of values
5. Set of values
6. Uniqueness constraints
7. Statistical constraints
8. Trigger fields (ones which when changed cause some other actions to take place also.)

### 3.5 Performance

Databases have a lot to do. Besides just storing and retrieving data, they are checking the validity etc. of that same data. Given all the requirements, it is no wonder that a database system often has a greater overhead than ordinary file systems. In terms of performance, then, one is

interested in a variety of issues, all of which make the database system faster. The elements we considered important in judging performance are:

1. Overall performance relatively like competitors
2. Fast load of relation
3. Fast unload of database
4. Dynamic index definition
5. Query optimization heuristics
6. Dynamic index maintenance
7. Number of indices per relation
8. Indices may be comprised of many segments in any order
9. Frequency of re-organization that would be necessary to support continued quality performance.
10. Utility to monitor the performance of the database
11. Fast recovery/check-point restart availability
12. Future ability to totally utilize VAX cluster architecture
13. Re-entrant code - (memory efficiency) only one database process for the entire system
14. Locking efficiency (the efficiency with which the database uses its lock manager, fine locks for small transactions and coarse locks for large transactions.)
15. Storage (disk) efficiency

### 3.6 Data Dictionary Support (external To Database)

A data dictionary is almost a database that describes the data that is going to be stored in the database. If you wish, it is called meta-data. All relational database products on VAX have a data dictionary imbedded in them. Often, these dictionaries are not available to other things like compilers of forms systems.

We feel that a dictionary that is fully understood by other software is very important. One should have only one central record definition. If something is changed, one wants to go only to one place to effect that change throughout the entire application. Experience tells us that such changes will occur with absolute certainty. We must minimize now the cost of such future changes. This single issue could cost an organization more in terms of additional time spent on problems than all other costs in an applications. The Data Dictionary is that important.

Features that we feel are important to have in a database system to support a data dictionary are:

1. Dynamic
2. Data dictionary support external to database
3. Security of data dictionary
4. Speed/performance of the data dictionary

### 3.7 Quality Of AD-HOC Query Language

We will not be able to pre-define all reports that are desired from the database. In some application areas, it has been shown that up to 75% of the total number of reports are ad-hoc reports. The tools that are available for making such ad-hoc reports, and indeed the tools for making the planned reports thus become extremely important.

Our model is that the end users themselves will be generating most of their own reports. Given this, we must be particularly concerned that such end-users are capable of using the tools provided. Features that we feel are important to have in an ad-hoc query language are:

1. Non-procedurality of the language (user friendliness)
2. Ability to include files outside of the database in a report
3. Ability to work in a distributed environment
4. Complexity of queries supported
5. Ability to utilize views effectively
6. Ability to utilize the standard forms package
7. Can be restricted from updating data
8. Report interface

### 3.8 Quality Of Programming Language Interface

Even with the features of relational databases, one will inevitably have to write some programs to support an application. It is important that we have the widest range of languages possible and that all are supported by the database system. We feel the following considerations are important when considering programming language support:

1. Program Language interface for VAX-11 BASIC because this was our current shop standard.
2. Program Language interface for VAX-11 PASCAL because we believed we should be considering this language. We were tired of being second guessed by the Computer Scientists.

3. Program Language interface for VAX-11 FORTRAN because it was the first language Jeff learned.
4. Program Language interface for VAX-11 COBOL because our management believes COBOL is right for data processing.
5. Full power of the DML available to host-language programs
6. Full power of the DDL available to host-language programs
7. Can the language(s) call the ad-hoc report writer
8. All supported languages access record definitions from Data Dictionary

### 3.9 Forms Interface

Many new applications are supported ONLY on CRT terminals. Up until now we had been required to support both hard-copy and CRT's. This has naturally led to some backwardness in the look and feel of all of our applications.

In a previous talk (last year in Las Vegas), we outlined the features we thought were appropriate to a forms manager. Without going into details, the products being evaluated all supported their own forms editor. No package completely fills the criteria we had outlined. Lacking perfection, we adopted the following generic forms criteria:

1. Forms available both to host-language and to database utilities
2. Forms use standard data dictionary record definitions
3. Ability to change forms without changing the application programs or packaged queries
4. Judgement of forms package relative to requirements (this is another session)

### 3.10 Miscellaneous Considerations

Several things are not covered by the categories above. These are all mixed together here is a single stew. The degree of importance each of these collected issues is clearly variable.

1. Ability to keep track of performance statistics
2. Ability to store the queries that actually were made to the database to enable future optimization strategy definitions
3. Initial cost

4. Maintenance cost
5. Generalized utility package
6. Vendor training
7. Documentation quality
8. Stability/reputation of vendor
9. User satisfaction
10. User support group(s)
11. Will provide future upgrade to database machine or to be distributed in a network or both.
12. Possibility of future natural-language query system
13. Familiarity
14. Stability of product
15. License agreements that are favorable in a VAX-Cluster environment
16. Ability to install a new version of the database system while continuing in production with the old.

**COMPARISON OF DATABASE FEATURES  
ACROSS ALL DATABASES**

| FEATURE                                | Rdb/VMS | Rdb/ELN | X1 | X2 |
|----------------------------------------|---------|---------|----|----|
| <b>General Software Features</b>       |         |         |    |    |
| Concurrency                            | Yes     |         |    |    |
| Datatypes                              | Yes     |         |    |    |
| Data format independence               | Yes     |         |    |    |
| Define tables on fly                   | Yes     |         |    |    |
| Define indices on fly                  | Yes     |         |    |    |
| Multiple database cursors              | Yes     |         |    |    |
| Multiple databases                     | Yes     |         |    |    |
| Modify table definitions               | Yes     |         |    |    |
| Columns in a relation                  | Large   |         |    |    |
| Relations in a database                | Large   |         |    |    |
| Define views                           | Yes     |         |    |    |
| Define views of views                  | Yes     |         |    |    |
| Concurrent Batch/on-line               | Yes     |         |    |    |
| Record sizes unlimited                 | 64k     |         |    |    |
| Inner Join                             | Yes     |         |    |    |
| Outer join                             | Awkward |         |    |    |
| No. tables in a single join            | 32      |         |    |    |
| Control trans start & end              | Yes     |         |    |    |
| Rollback and Commit                    | Yes     |         |    |    |
| Aggregation functions                  | Yes     |         |    |    |
| DDL and DML                            | Yes     |         |    |    |
| Fully functional DML                   | Yes     |         |    |    |
| <b>Security</b>                        |         |         |    |    |
| Access on per-user basis               | Yes     |         |    |    |
| Access to db on a per-user             | Yes     |         |    |    |
| Data modified only by prog (excl. DBA) | Yes     |         |    |    |
| Audit trail                            | No      |         |    |    |
| <b>Database Consistency</b>            |         |         |    |    |
| Journaling                             | Yes     |         |    |    |
| Roll-back incomplete                   | Yes     |         |    |    |
| Roll-forward                           | Yes     |         |    |    |
| Automatic recovery                     | Yes     |         |    |    |
| Utility to monitor integrity           | Yes     |         |    |    |
| Quality of integrity                   | Yes     |         |    |    |
| Deadlock detection                     | Yes     |         |    |    |
| Two-phase locking                      | Yes     |         |    |    |
| Two-phase commit                       | Yes     |         |    |    |
| Database Backup                        | Yes     |         |    |    |
| Maturity of software                   | Semi    |         |    |    |
| <b>Data Integrity</b>                  |         |         |    |    |
| Verification/validity checks/edits     | Yes     |         |    |    |
| Constraints                            | Yes     |         |    |    |
| Check constraints on commit            | Yes     |         |    |    |
| Range of values                        | Yes     |         |    |    |
| Set of values                          | Yes     |         |    |    |
| Uniqueness constraints                 | Yes     |         |    |    |
| Statistical constraints                | Yes     |         |    |    |
| Triggers                               | No      |         |    |    |

COMPARISON OF DATABASE FEATURES  
ACROSS ALL DATABASES (cont)

| FEATURE                                          | Rdb/VMS   | Rdb/ELN | X1 | X2 |
|--------------------------------------------------|-----------|---------|----|----|
| <b>Performance</b>                               |           |         |    |    |
| Performance like competitors                     | Yes       |         |    |    |
| Fast load                                        | Yes       |         |    |    |
| Fast un-load                                     | Yes       |         |    |    |
| Dynamic index definition                         | Yes       |         |    |    |
| Dynamic index maintenance                        | Yes       |         |    |    |
| No. of indices per relation                      | Infinite  |         |    |    |
| Segmented indices                                | Infinite  |         |    |    |
| Query optimization heuristic                     | Automatic |         |    |    |
| Infrequent re-organization                       | Yes       |         |    |    |
| Monitor performance of DB                        | sort of   |         |    |    |
| Fast recovery/check-point restart                | Yes       |         |    |    |
| Future ability for VAX cluster                   | Yes       |         |    |    |
| Re-entrant code                                  | Yes       |         |    |    |
| Locking efficiency                               | Good      |         |    |    |
| Storage efficiency                               | Good      |         |    |    |
| <b>Data Dictionary support</b>                   |           |         |    |    |
| Dynamic                                          | Yes       |         |    |    |
| Data dictionary external                         | Yes       |         |    |    |
| Security of data dictionary                      | Yes       |         |    |    |
| Speed/performance of DD                          | Yes       |         |    |    |
| <b>Quality of AD-HOC Query Language</b>          |           |         |    |    |
| User friendliness                                | Yes       |         |    |    |
| Include files external to DB                     | Yes       |         |    |    |
| Work in distributed environment                  | Yes       |         |    |    |
| Complexity                                       | Yes       |         |    |    |
| Utilize views                                    | Yes       |         |    |    |
| Utilize standard forms package                   | Yes       |         |    |    |
| Restricted from updating data                    | Yes       |         |    |    |
| <b>Quality of programming language interface</b> |           |         |    |    |
| VAX-11 BASIC                                     | Yes       |         |    |    |
| VAX-11 PASCAL                                    | Yes       |         |    |    |
| VAX-11 FORTRAN                                   | Yes       |         |    |    |
| VAX-11 COBOL                                     | Yes       |         |    |    |
| DML available to programs                        | Yes       |         |    |    |
| DDL available to programs                        | Yes       |         |    |    |
| Call ad-hoc report writer                        | Yes       |         |    |    |
| Access record def from DD                        | Yes       |         |    |    |
| <b>Forms interface</b>                           |           |         |    |    |
| Available to programs and DB utilities           | Yes       |         |    |    |
| Use standard DD definitions                      | Yes       |         |    |    |
| Change forms without changing program            | Yes       |         |    |    |
| Forms package meets DU requirements              | No        |         |    |    |
| <b>Miscellaneous considerations</b>              |           |         |    |    |
| Keep track of performance stats                  | No        |         |    |    |
| Store the queries                                | No        |         |    |    |
| Initial cost                                     | ?         |         |    |    |
| Maintenance cost                                 | ?         |         |    |    |
| Generalized utility package                      | Semi      |         |    |    |
| Vendor training                                  | Yes       |         |    |    |
| Documentation quality                            | Good      |         |    |    |
| Stability/reputation of vendor                   | Very      |         |    |    |
| User satisfaction                                | Yes       |         |    |    |
| User support group(s)                            | Yes       |         |    |    |
| Upgrade to DB machine                            | Yes       |         |    |    |
| Natural-language query                           | Future    |         |    |    |
| Familiarity                                      | Yes       |         |    |    |
| Stability of product                             | Yes       |         |    |    |
| Favorable License agreements                     | Yes       |         |    |    |
| Test DB concurrent w/ Prod                       | Yes       |         |    |    |

Larry R. Creel  
Science Applications International Corporation  
(SAIC)  
Los Alamos, New Mexico

ABSTRACT

This paper describes the implementation of a bar-coded property inventory system at the Los Alamos National Laboratory. It identifies "lessons learned" due to problems encountered during system development. Discussed are bar code scanners, bar code labels, portable data collection, data unloading, file transfer between micro and mainframe, some applicable database management techniques, software control, how much computer proficiency is required of property managers, and a method of assuring the integrity of files that are updated independently from multiple sources.

INTRODUCTION

Maintaining current inventory records on over 100,000 items is a difficult task. This paper describes how database management techniques combined with bar coding can be utilized to reduce errors and increase productivity in the area of property management.

In July, 1984, SAIC was contracted to provide a software system to assist the Property Representatives at the Los Alamos National Laboratory (LANL) in automating their inventory techniques. The fully implemented system, simply put, operates in the following way.

Inventory data is collected with a portable bar code reader.

Collected data is processed against a property database.

Feedback is given to property representatives.

What has been found.

What has not been found.

What was invalid.

Notification of new record arrivals.

Notification of record deletions.

SAIC is completing the final stages of that project at this time. It was found to be a most interesting assignment.

DEFINITIONS

Some terms used in the system description that are specific to the implementation at Los Alamos are also relevant to a "generic" implementation at another site with similar operating requirements.

Lab Database

Property records are maintained in a large database with strict updating controls. Property representatives modify the location fields in this database by providing an input file containing the data to a software program that does the actual update. Because of its size, the database runs on a large computer under the control of a Database Management System (DBMS). At Los Alamos, it is a System 2000 database containing about 100 megabytes of data running on a CDC Cyber 825/NOS computer.

The Lab database represents LANL's official property records and may be updated only by their Administrative Data Processing group.

## DEFINITIONS (CONTINUED)

### Local Database

When a large amount of property is involved, a number of people are assigned responsibility for its whereabouts. Each of these people may be responsible for between three hundred and twelve thousand items. In order for these property representatives to maintain current records on their assigned property, they are provided with their own smaller database extracted originally from the Lab database.

The Local database provides the property representative with a set of data to process against that can be updated freely with no possibility of corrupting the Lab database.

### Global files

Since the Lab database requires location updates to come from a single input file, and there are many property representatives who must contribute to this file, it is referred to as a global file.

Global files serve as accumulators for data from multiple sources.

Integrity protection for these global files is provided by software procedures. When one program is adding data, no other program can access it.

### PROBLEMS TO SOLVE

The method used by the property representatives to provide location information to the Lab database was to complete a handwritten coding sheet containing the property number and the location. This data was keypunched, read into a file on the host computer, and processed against the Lab database. Before any increase in productivity could be realized, inventory data processing via handwritten coding sheets had to be eliminated.

If the coding sheets are eliminated, an inventory data collection and processing system must be developed to replace them. For reasons of accuracy and efficiency, bar coding technology and DBMS technology offer the most attractive means of providing the optimum system.

This system for inventory must also interface with existing update methods of the Lab database. Specifically, this meant the program that modifies the location fields in the Lab database would

not be changed and the new system had responsibility for correctly generating a single properly formatted and accurate location update input file on disk for it to read and process.

### STEPS TAKEN BY LANL

The decision was made to have the property representatives use portable bar code readers to take inventory. Machine readable CODE 39 bar code labels were applied to newly arriving capital equipment and property representatives were given the task of labeling all property that had the old non-machine readable property number labels with CODE 39 labels.

SAIC was contracted to develop an "inventory data system" for the property representatives and to train them in its use.

### ACTION TAKEN BY SAIC TO PROVIDE "INVENTORY DATA SYSTEM"

The approach taken was to provide a local database to process against for each group. This set of data would be managed by a relational DBMS software package that would provide the processing power to do the data reduction. There are several steps involved; some of which were done concurrently to minimize the implementation time.

1) Provide a Local database for each property representative.

For the property representative to provide inventory information to the Lab database, a Local database containing data on assigned property must be available.

A Local database was created and loaded from the Lab database for each property representative. It is managed with the FRAMIS DBMS software to provide the representative with data independence.

FRAMIS is a fully relational DBMS developed by Lawrence Livermore National Laboratory. It is easy to learn and use and runs on VAX/VMS, CRAY/CTSS, and CDC 7600/LTSS. The Local database puts property representatives in direct contact with their data allowing them the complete update privileges they need.

2) A portable bar code reader is programmed to collect data.

A portable bar code reader is used by the property representative to inventory assigned equipment. Bar code readers can be used as INPUT devices for reading either the item's bar code number, its

location, or both. Two programs were needed at LANL.

An "Assignment" program was written to collect an old property number, a new bar code number, and location data. The term "assignment" refers to collecting the CODE 39 bar code label applied to all equipment not already having one. When the decision was made to automate the inventory process, LANL began applying bar code labels to equipment as property numbers. All equipment received prior to bar coding had to have a label applied by the property representative. The old property number and new bar code label had to be collected and stored in the Local database and later provided to the Lab database.

If an item does not possess a CODE 39 label, the representative applies one and records the value in the reader as the "bar code number". If the item possesses a CODE 39 label, the entry for the "property number" and the "bar code number" are the same.

An "Inventory" program was written to collect a bar code number and location data. It is the same as the "Assignment" program except it does not request a value for the old property number.

3) Data from the bar code reader is unloaded into a microcomputer.

Development of communication software is expensive and off-the-shelf software should be used whenever possible. The KERMIT software package developed at Columbia University was found to be satisfactory for the LANL inventory system and was free of charge.

KERMIT is used to transfer data between the portable bar code reader and the microcomputer and between the microcomputer and the host computer. KERMIT can transmit data over telephone lines or over direct RS-232 communication lines.

The data collected in the portable bar code readers are unloaded into a microcomputer for reasons of data integrity and convenience. Groups at LANL are using the IBM PC, DEC Rainbow, or an HP-150.

An editor (VEDIT) on the microcomputer is used to format the data into eighty character records.

After formatting and possibly editing, the bar code reader data is shipped to a host computer for processing against the Local database. A qualified host computer is one that can access the global location update

file and one that is capable of running the FRAMIS DBMS software. At Los Alamos those machines are CDC 7600/LTSS, CRAY-1/CTSS, and VAX/VMS computers.

4) Merge procedures were developed to incorporate the portable bar code reader data into the Local database.

Once the information arrives at the host, procedures process the data against the Local database. The FRAMIS command language is used to query data. To print what was found, the representative may enter

```
PRINT VIEW WHERE INVDTE<>" ";
```

To print what was not found, the representative may enter

```
PRINT VIEW WHERE INVDTE=" ";
```

(VIEW is the name of the table in the Local database that contains the property data and INVDTE is the field containing the date of inventory.)

Listings are printed as a result merging the data from the portable bar code reader into the Local database. One report is a listings of "What did not belong". An item does not belong if no match was found by property number. Such records are posted to a global "no match" file that is later checked against the Lab database to see if the record belongs to another group. This feature allows a property representative to enter a room and inventory every item of capital equipment whether it is assigned to him or not.

5) Correlation procedures were developed to automatically compare Local database records with the Lab database records.

The Lab database is considered to be correct in all respects except for location information which will come from the Local database. Based upon these assumptions:

Current Lab data is extracted from the Lab database.

Records whose locations in the Local database are different from their location in the Lab database will generate a "location update transaction".

"Location update transactions" are written to a "global location update file" that is processed against the Lab database.

Property representatives need to know when new property is assigned to them. They



also need to know when property assigned to them has been decontrolled. Records in the Lab database that are not in the Local database (new property) are automatically added. Records in the Local database that are not in the Lab database are considered "decontrolled" and are transferred to a HISTORY table.

The requirement to send location updates from Local databases to the Lab database had been satisfied, but the problem of associating new bar code numbers with old property numbers still remained.

Procedures were developed to create an update file for the Lab database that associates the new bar code number with the old property number based upon matching the property number and group identifier in the update file with the property number and group identifier in the Lab database. Procedures to process that update file against the Lab database are being completed at this time.

#### ERROR CHECKING PERFORMED ON THE LOCAL DATABASE

A system is as good as the error prevention it provides. When the Local database is updated either by merging inventory data from the bar code reader or by comparison of the Local database with the Lab database, extensive error checking occurs. Among these safeguards are:

Property numbers are required to be unique.

Bar code numbers are required to be unique.

Bar code labels assigned to property numbers in the Local database must agree with those in the Lab database.

No location update transactions are generated for records with incomplete locations.

Records in the Local database are required to be contained in the Lab database. Decontrolled items automatically go into a HISTORY table.

Hardcopy listings are automatically printed informing the property representative rejected records and the reason for their rejection.

#### ASSURING THE INTEGRITY OF GLOBAL FILES

Global are files accumulator files that become input files to the Lab database. They require additions from all property

representatives. A scenario in which data can be lost when two users are writing to the same file is when one gets the file and updates it while another is updating it. Only those updates made by the last person to replace the file will be kept. The updates made by the first to replace the file will be lost.

A method of locking the global files was developed using passwords. A file with a password cannot be accessed unless its password is correctly given.

The global file is stored with a password of "UNLOCKED". A Fortran program retrieves it for updating and replaces it when modifications are completed.

Before retrieving the global file, the Fortran program changes its password from "UNLOCKED" to "LOCKED". If the change is successful, no one else was modifying the file, and processing continues. If the change was unsuccessful (i.e. the password was something other than "UNLOCKED"), someone else was modifying the file, and processing stops and the representative is requested to try again later.

If the password change went okay, the file is modified, replaced, and the password is changed from "LOCKED" to "UNLOCKED". The file can now be updated by another property representative.

#### DOCUMENTATION

Documentation is still being written and revised. A "PROPERTY REPRESENTATIVE'S HANDBOOK OF BAR CODED INVENTORY PROCEDURES" provides a user's guide for the property representative. A programmer's reference manual is being developed to provide system maintenance information.

#### TRAINING

Formal classroom training and on-site training immediately following implementation is provided to the property representatives. Telephone consulting is also available.

## WHAT WAS LEARNED

Many useful facts were discovered; a few were already known and confirmed by this project. Some of the most important are:

Bar code readers must be reliable, reasonably priced, and able to be programmed in a high level language. Bar code readers that can be used for other functions when they are not needed for inventory are an extra benefit.

Bar coding technology decreases the time required for inventory and reduces the number of errors made.

Scanning a bar code label improves the reliability of an inventory because it requires the property representative to physically come in contact with the item.

Computer inexperience by users can be overcome by very detailed instructions (both verbal and written) and one-on-one training using the system. This requires a great deal of patience and understanding from the instructor.

Support for the system from the property representative's supervisor is very important because:

Time and effort is required from the property representative.

A bar code reader, microcomputer, and a communications port must be purchased.

The first inventory using bar code labels may take the property representative longer than using coding sheets because:

He is unfamiliar with the bar code reader.

He must also key in the old property number.

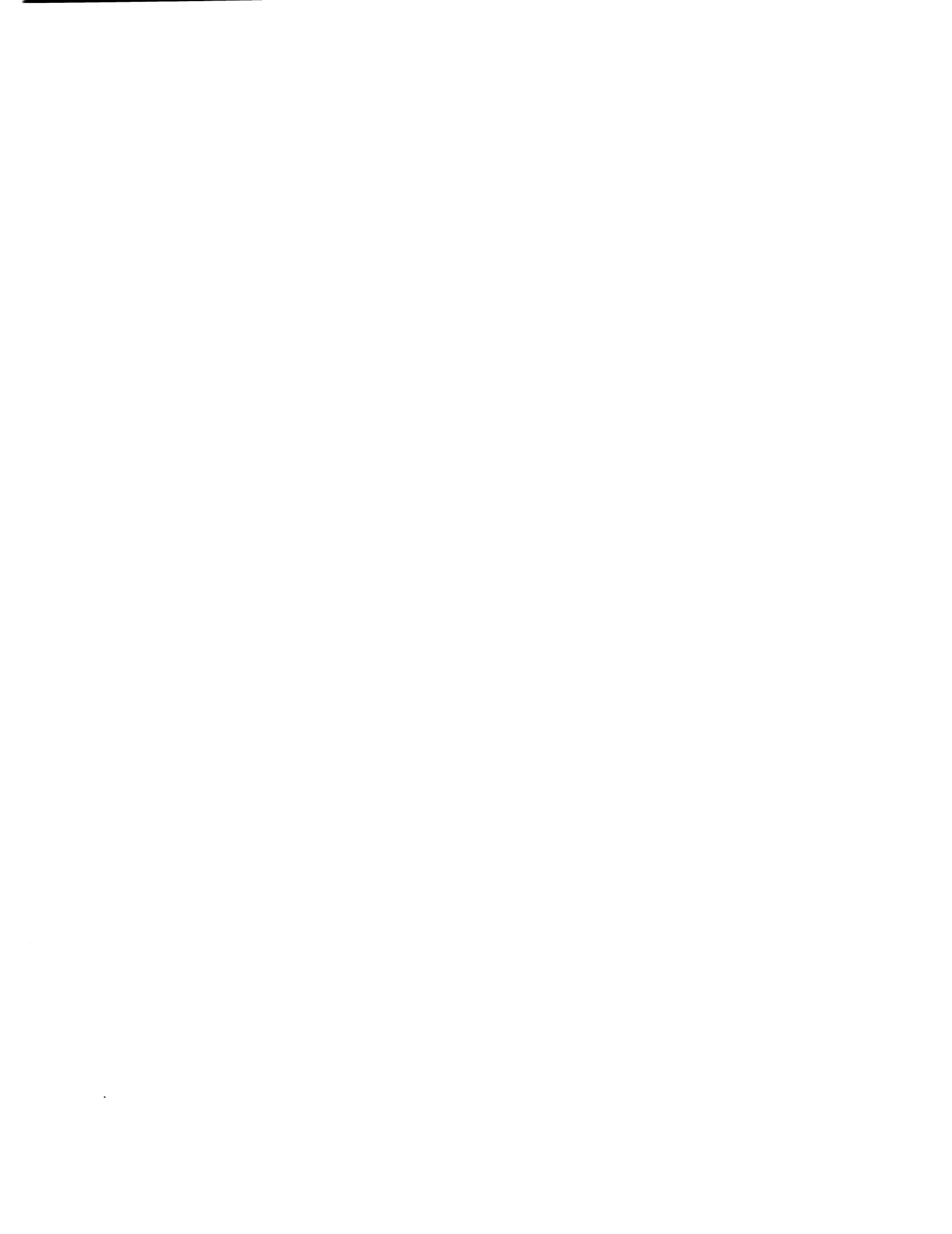
He must make physical contact with the item being inventoried.

Scheduling groups for training and implementation can be very time consuming. People are often too busy to be trained in the new system.

Error checking, prevention and recovery throughout the system is essential.

The majority of property representatives at LANL had very little computer experience, yet most were eager to learn. For a system to be successful, it must be made as simple as possible. Programmers must always be willing to work harder in the software when it will make running and using the product easier.

Our documentation is strictly cookbook and procedure files handle all of the data processing. Time will tell how successful we were at Los Alamos.



CREATING MENU-DRIVEN SYSTEMS  
USING FMS AND VAX DCL

Brian D. Lockrey  
ITT Telecom  
P.O. Box 20345, NW Station  
Columbus, Ohio 43220

Abstract

The VAX/VMS operating system from Digital Equipment Corporation includes a command language interpreter called DCL. An optional software product is also available from DEC, called the Forms Management System. VAX/VMS does not currently provide a facility to use FMS forms within a DCL command procedure. By writing a generalized program, it is possible to create menu-driven systems using FMS forms as menus and the DCL command interpreter as a procedure language to control the presentation of these menus. The purpose of this paper is to describe a prototype program that performs this function.

Introduction

To facilitate using FMS forms with a DCL command procedure, a generalized program has been written which binds FMS field names to DCL symbols. By using this convention, any DCL command procedure can interact with an FMS form on a limited basis without the use of a specialized application program. While the capabilities of such an interface do not provide for all the functionality offered by FMS, the interface provides enough flexibility to create menu-driven systems.

Most menu-driven systems are controlled by a main menu which offers various options to the user. The option selected may invoke another sub-menu or activate the execution of an application program. For example, one menu option may be a request to produce a report on a line printer. This report may actually be created by submitting a batch job that runs Datatrieve, creates a report, and spools the report to the line printer.

Since menus may also be used to run application programs and system utilities, it appears that the logical place to implement the menu driver is from the command language interpreter. Since DCL provides all the necessary functions required to run programs and execute system utilities, the only extension needed is a facility to present the menu on the terminal screen.

The PANEL Program

A program called PANEL was written which can be used to display an FMS form, solicit input from the terminal, and pass the data to DCL. The DCL command procedure may then use the data to make a determination as to what the next action should be.

PANEL communicates with DCL through the DCL symbol table. When the PANEL program terminates, each field name on the form is prefixed with 'P\$' and a DCL symbol is created. The value contained on the form is assigned to the DCL symbol and stored in the symbol table. These symbols may be created either LOCAL or GLOBAL through the use of a command qualifier passed to the PANEL program.

Example 1

The best way to describe the function of the PANEL program is by examples. The first example is a command procedure and FMS form used to enter parameters for the VAX SORT utility. The form allows entry of an input file name, an output file name, and up to five fields on which to sort. This particular menu does not allow for all of the available sorting options; however, it is sufficient for many applications.

Assuming the ASORT command has been predefined in the user's DCL symbol table as follows.

```
$ASORT ::= "@MENU:ASORT.COM"
```

the ASORT command then causes the ASORT.COM procedure to execute (see figure 1). This procedure then calls the PANEL program to display ASORT.FRM (see figure 2) from the MENU.FLB library. After the parameters have been entered into the appropriate fields, the user presses the RETURN key to terminate the PANEL program. Prior to exiting, PANEL creates 10 symbols in the user's DCL symbol table from the corresponding fields in the FMS form. At this point the ASORT command procedure builds a command line and invokes the VAX SORT utility in line 12.

While this example is a simplified application, it should help convey the actual function performed by the PANEL utility. By using a small DCL command procedure and an FMS form, an interface has been created for the VAX SORT utility without writing any additional software.

#### Example 2

The second example is called the Media Control Systems (MCS). This system was designed to maintain a catalog of video and audio tapes. The catalog is stored as an indexed file that may be modified by the user. Various reports may be produced from the records within the file.

This system also uses only one menu screen. From the menu, the user is able to call a program to add, delete, and modify records in the file and then produce reports using the data stored in the file. The entire system is driven by MCS.COM, which uses the form MCSMENU as the main menu (see figure 3). The reports are created by submitting batch files that use the Datatrieve report writer. The report procedures are not shown.

Referring to line 6 in figure 3, the command procedure calls the PANEL program which displays the MCSMENU form from the MCS.FLB forms library. The menu shown in figure 4 contains only one field named OPTION. Upon termination of the PANEL program, the DCL symbol P\$OPTION will contain the data corresponding to the option specified by the user.

The PANEL command in line 10 is used to display the main menu after a particular option has been performed. The /REUSE qualifier instructs PANEL to restore the option field with the data contained in the DCL symbol P\$OPTION. By using this qualifier, the menu is restored to exactly the way it was when the PANEL command was complete at line 10.

The /LAST qualifier is used to pass a string that is displayed on the last line of the form. In this case, a message indicating the action taken by the procedure is displayed. This provides a facility to display error messages or other text to the user. The /BELL qualifier causes the bell in the terminal to ring when the menu is displayed.

#### Example 3

The third example is a menu system developed to assist users wishing to perform various file conversions. The main menu contains nine options. These options include converting carriage control, removing tabs, selecting records, and converting a file between upper and lower case. From the main menu the user invokes one of five utility programs used to perform the requested operation. The utilities are invoked by the command procedure FILECONV.COM shown in figure 5.

Each of these utility programs has different switches and qualifiers which must be known prior to their use. As is often the case, the syntax of the commands is different between each of the five utilities. By using a main menu and a few sub-menus, these differences may be hidden from the user.

The FILECONV.COM procedure displays the main menu, shown in Figure 6, where the user enters the input and output file names. Depending on the requested option, the procedure may display a sub-menu on the lower portion of the screen. The sub-menu used with option 1 of the FILECONV procedure is shown in Figure 7.

These sub-menus enable the user to enter parameters that are specific to the current option. For instance, if the user has selected option 6 to shift a file to the right, the OPTION6 menu contains a field asking the user to enter the number of columns the file should be shifted to the right. Once the user has entered the parameter, the procedure would then call the REFORMAT program to perform the shift operation.

Since the command procedure builds the actual parameters passed to the REFORMAT program, the user is not required to know the syntax of the command string actually passed to the target program. In many cases, such as this one, the user may not even be aware that another program is called to reformat the file.

#### Conclusion

The PANEL program adds to DCL the capability needed to display and interact with menus. Since the menus can easily be created using existing FMS utilities, the programmer is not required to learn additional system commands and editors in order to use this facility. Although the PANEL program does not support all of the features found on other systems, it provides the framework that allows the creation of workable menu-driven systems in a VAX/VMS environment.

```

1: $ PANEL MENU,FLB ASORT
2: $ KEY1 = '/KEY=(POS=' + P$POS1 + ',SIZE=' + P$LEN1 + ')"
3: $ KEY2 = '/KEY=(POS=' + P$POS2 + ',SIZE=' + P$LEN2 + ')"
4: $ KEY3 = '/KEY=(POS=' + P$POS3 + ',SIZE=' + P$LEN3 + ')"
5: $ KEY4 = '/KEY=(POS=' + P$POS4 + ',SIZE=' + P$LEN4 + ')"
6: $ CMD = P$OUTFILE+KEY1
7: $ IF KEY2 .NES. '/KEY=(POS=,SIZE=)' THEN CMD = CMD + KEY2
8: $ IF KEY3 .NES. '/KEY=(POS=,SIZE=)' THEN CMD = CMD + KEY3
9: $ IF KEY4 .NES. '/KEY=(POS=,SIZE=)' THEN CMD = CMD + KEY4
10: $ MSG = 'Sorting File: ' + P$INFILE
11: $ WRITE SYS$OUTPUT MSG
12: $ SORT 'P$INFILE' 'CMD'

```

Figure 1.

Form: ASORT

```

 1 2 3 4 5 6 7 8
1234567890123456789012345678901234567890123456789012345678901234567890

1| ASORT |11
2| |12
3| |13
4| |14
5|Input File: -----|15
6| |16
7|Output File: -----|17
8| |18
9|Position: ----- Length: -----|19
10| |10
11|Position: ----- Length: -----|11
12| |12
13|Position: ----- Length: -----|13
14| |14
15|Position: ----- Length: -----|15
16| |16

1234567890123456789012345678901234567890123456789012345678901234567890
 1 2 3 4 5 6 7 8

```

Figure 2.

```

1: $!
2: $! MCS.COM
3: $! MEDIA CONTROL SYSTEM - MAIN MENU
4: $!
5: $ START:
6: $ PANEL/GLOBAL MCS.FLB MCSMENU
7: $ GOTO SELECT
8: $!
9: $ REDO:
10: $ PANEL/GLOBAL/REUSE/BELL/LAST=&MSG MCS.FLB MCSMENU
11: $!
12: $ SELECT:
13: $ IF P$OPTION .EQS. '1' THEN GOTO MCS_EDIT
14: $ IF P$OPTION .EQS. '2' THEN GOTO MCS_REPORT_1
15: $ IF P$OPTION .EQS. '3' THEN GOTO MCS_REPORT_2
16: $ IF P$OPTION .EQS. '4' THEN GOTO MCS_EXIT
17: $ MSG = 'Illegal menu option, please try again...'
18: $ GOTO REDO
19: $!
20: $ MCS_EDIT:
21: $ IDE MCS
22: $ GOTO START
23: $!
24: $ MCS_REPORT_1:
25: $ SUBMIT/NOPRINT REPORT1.COM
26: $ MSG = 'Report 1 has been submitted...'
27: $ P$OPTION ;== ' '
28: $ GOTO REDO
29: $!
30: $ MCS_REPORT_2:
31: $ SUBMIT/NOPRINT REPORT2.COM
32: $ MSG = 'Report 2 has been submitted...'
33: $ P$OPTION ;== ' '
34: $ GOTO REDO
35: $!
36: $ MCS_EXIT:
37: $ EXIT

```

Figure 3.

Form: MCSMENU

```

 1 2 3 4 5 6 7 8
1234567890123456789012345678901234567890123456789012345678901234567890

11 MEDIA CONTROL SYSTEM 11
12
13 Main Menu 13
14
15 Options available are: 15
16
17 1 - Edit the MCS Database 17
18 2 - Print Media Catalog Report 18
19 3 - Print Media Cross Reference Report 19
20 4 - Exit from MCS 20
21
22 Select Option: _ 22
23
1234567890123456789012345678901234567890123456789012345678901234567890
 1 2 3 4 5 6 7 8

```

Figure 4.

```

1: $!
2: $! FILECONV.COM
3: $! FILE CONVERT UTILITIES - MAIN MENU
4: $!
5: $ ON CONTROL_Y THEN GOTO INTERRUPT
6: $ START:
7: $ PANEL/CLEAR/GLOBAL CONVERT.FLB CONVERT
8: $ GOTO SELECT
9: $!
10: $ REDO:
11: $ PANEL/GLOBAL/REUSE/BELL/LAST=&MSG/CURSOR=OPTION -
12: $ CONVERT.FLB CONVERT
13: $ SELECT:
14: $ MSG = " "
15: $ IF P$OPTION .EQS. '1' THEN GOTO OPTION_1
16: $ IF P$OPTION .EQS. '2' THEN GOTO OPTION_2
17: $ IF P$OPTION .EQS. '3' THEN GOTO OPTION_3
18: $ IF P$OPTION .EQS. '4' THEN GOTO OPTION_4
19: $ IF P$OPTION .EQS. '5' THEN GOTO OPTION_5
20: $ IF P$OPTION .EQS. '6' THEN GOTO OPTION_6
21: $ IF P$OPTION .EQS. '7' THEN GOTO OPTION_7
22: $ IF P$OPTION .EQS. '8' THEN GOTO OPTION_8
23: $ IF P$OPTION .EQS. '9' THEN GOTO OPTION_9
24: $ IF P$OPTION .EQS. 'E' THEN GOTO OPTION_E
25: $ MSG = "Illegal menu option, please try again..."
26: $ GOTO REDO
27: $!
28: $ OPTION_1:
29: $ PANEL/LAST=&MSG CONVERT OPTION1
30: $ PARAM = "?"
31: $ IF P$CONTROL .EQS. 'F' THEN PARAM = "/FORTRAN"
32: $ IF P$CONTROL .EQS. 'L' THEN PARAM = "/LIST"
33: $ IF P$CONTROL .EQS. 'P' THEN PARAM = "/PRINT"
34: $ IF P$CONTROL .EQS. 'N' THEN PARAM = "/NONE"
35: $ IF PARAM .EQS. '?' THEN GOTO ERROR_1
36: $ TRANSFORM 'P$FILEIN' 'P$FILEOUT' PARAM
37: $ MSG = "File has been converted."
38: $ GOTO REDO
39: $!
40: $ ERROR_1:
41: $ MSG = "Invalid option, please try again."
42: $ GOTO OPTION_1
43: $!
44: $ OPTION_2:
45: $ DETAB 'P$FILEIN' 'P$FILEOUT'
46: $ P$FILEIN = P$FILEOUT
47: $ MSG = "File has been detabed."
48: $ GOTO REDO
49: $!
50: $ OPTION_3:
51: $ ENTAB 'P$FILEIN' 'P$FILEOUT'
52: $ P$FILEIN = P$FILEOUT
53: $ MSG = "File has been entabed."
54: $ GOTO REDO
55: $!
56: $ OPTION_4:
57: $ PANEL CONVERT OPTION4

```

Figure 5, Part 1.



```

58: $ REDO_4:
59: $ IF P$LRECORD .NES. "" .AND. P$CRECORD .NES. "" THEN GOTO ERROR_4
60: $ PARAM = "(START:" + P$FRECORD + ","
61: $ IF P$LRECORD .NES. "" THEN PARAM = PARAM + "END:" + P$LRECORD + ")"
62: $ IF P$CRECORD .NES. "" THEN PARAM = PARAM + "COU:" + P$CRECORD + ")"
63: $ SCOPY 'P$FILEIN' 'P$FILEOUT/RECORDS='PARAM'
64: $ P$FILEIN = P$FILEOUT
65: $ MSG = "Records have been selected."
66: $ GOTO REDO
67: $!
68: $ ERROR_4:
69: $ MSG = "Can not specify the last record and a count."
70: $ PANEL/REUSE/LAST=&MSG CONVERT OPTION4
71: $ GOTO REDO_4
72: $!
73: $ OPTION_5:
74: $ PANEL CONVERT OPTIONS
75: $ PARAM = "(" + P$FCOLUMN + "," + P$LCOLUMN + ")"
76: $ REFORMAT 'P$FILEIN' 'P$FILEOUT'/COLUMNS='PARAM'
77: $ P$FILEIN = P$FILEOUT
78: $ MSG = "Columns have been selected."
79: $ GOTO REDO
80: $!
81: $!
82: $ OPTION_6:
83: $ PANEL CONVERT OPTION6
84: $ REFORMAT 'P$FILEIN' 'P$FILEOUT'/SHIFT='P$SHIFT'
85: $ P$FILEIN = P$FILEOUT
86: $ MSG = "File has been shifted right."
87: $ GOTO REDO
88: $!
89: $ OPTION_7:
90: $ PANEL CONVERT OPTION7
91: $ REFORMAT 'P$FILEIN' 'P$FILEOUT'/SHIFT='-'P$SHIFT'
92: $ P$FILEIN = P$FILEOUT
93: $ MSG = "File has been shifted left."
94: $ GOTO REDO
95: $!
96: $ OPTION_8:
97: $ TEXTPROC 'P$FILEIN' 'P$FILEOUT'
98: $ \U
99: $ EOD
100: $ P$FILEIN = P$FILEOUT
101: $ MSG = "File has been converted to uppercase."
102: $ GOTO REDO
103: $!
104: $ OPTION_9:
105: $ TEXTPROC 'P$FILEIN' 'P$FILEOUT'
106: $ \L
107: $ EOD
108: $ P$FILEIN = P$FILEOUT
109: $ MSG = "File has been converted to lowercase."
110: $ GOTO REDO
111: $!
112: $ OPTION_E:
113: $ EXIT
114: $ INTERRUPT:
115: $ MSG = "Operation aborted by Control-Y"
116: $ GOTO REDO

```

Figure 5, Part 2.

Form: CONVERT

```
 1 2 3 4 5 6 7 8
1234567890123456789012345678901234567890123456789012345678901234567890

101
111. Change file carriage control
1212. Detab the input file
1313. Entab the input file
1414. Select range of records
1515. Select range of columns
1616. Shift the records to the right
1717. Shift the records to the left
1818. Convert file to uppercase
1919. Convert file to lowercase
201E. Exit this program
211
221Option: _
231

1234567890123456789012345678901234567890123456789012345678901234567890
 1 2 3 4 5 6 7 8
```

Figure 6.

Form: OPTION1

```
 1 2 3 4 5 6 7 8
1234567890123456789012345678901234567890123456789012345678901234567890

101
111
121
131 Option 1 - Change File Carriage Control
141
151
161The input file should be converted to:
171
181 F - Fortran
191 L - List
201 P - Print
211 N - None
221
231What type: _

1234567890123456789012345678901234567890123456789012345678901234567890
 1 2 3 4 5 6 7 8
```

Figure 7.







ELLIOT F. JAQUITH, JR  
 E. I. du PONT DE NEMOURS & CO.(RETIRED)  
 QUALITY CONTROL - STATISTICS - SOFTWARE  
 CONSULTANT  
 335 OVERLOOK TERRACE  
 HENDERSONVILLE, NORTH CAROLINA 28739

## ABSTRACT

This paper describes how to use a domain table as a bridge between interactive and batch DATATRIEVE when requesting queries on a very large file with variable search parameters. Our very large files queries require at least 45 minutes search time. By using a domain table to hold the variable information entered from the keyboard with interactive DATATRIEVE via FMS screens, parameter entry is reduced to 10 minutes while actual search time will take at least 45 minutes, leaving the terminal free for other tasks. This DOMAIN table is used to supply parameters to a batch program that prepares a special report.

When large files (>100,000 records) are the source of your data and many boolean expressions are used in the search code the time(15 to 45 minutes) for the interactive searches is longer than most computer users will accept. One alternative is the use of the BATCH processor. This means the terminal is available for other tasks while the DATATRIEVE search is going on in the background. A simple DCL command language job batch file works well if the boolean expressions to define the search criteria are the same for each DATATRIEVE search or you are familiar with computer EDITORS and DATATRIEVE. However, most queries are for new or different information. Many users who need this information have little experience or incentive to program in DATATRIEVE. The techniques described will permit DATATRIEVE queries by entering interactively the parameters list and using the batch processor to develop a collection, print, or prepare reports. There are two ways to develop the RSE (Record Selection Expression). The first is where the selection list has a variable number of entries. The second is where the selection list has a fixed number.

To illustrate how to develop the DATATRIEVE PROCEDURES and DCL command files, a simple personnel file will be used. This type of data structure is used in the "DEC" manuals (CHAPTER 6 OF THE INTRODUCTION TO VAX-11 DATATRIEVE) and by other manufacturers of data storage systems to show how their data base systems operate. In the examples the following record structure will be used:

```
DTR> SHOW FIELDS
PERSONNEL
PERSON
 ID <Number, indexed key>
 EMPLOYEE_STATUS (STATUS) <Character string>
 EMPLOYEE_NAME (NAME)
 FIRST_NAME (F_NAME) <Character string>
 LAST_NAME (L_NAME) <Character string>
 DEPT <Character string>
 START_DATE <Date>
 SALARY <Number>
 SUP_ID <Number>
```

The domain is "DEFINED DOMAIN PERSONNEL USING PERSONNEL\_REC ON PERSONNEL.DAT;"

The following will illustrate methods used to query the personnel data base. For a one time search using interactive DATATRIEVE the following commands could be typed directly to select all personnel who work for a supervisor with ID equal 12 and 891

```
DTR> READY PERSONNEL
DTR> FIND PERSONNEL WITH SUP-ID=12,891
[10 records found]
DTR> PRINT ALL
```

| ID    | STATUS      | FIRST NAME | LAST NAME | DEPT | START DATE  | SALARY   | SUP ID |
|-------|-------------|------------|-----------|------|-------------|----------|--------|
| 00012 | EXPERIENCED | CHARLOTTE  | SPIVA     | TOP  | 12-Sep-1972 | \$75,892 | 00012  |
| 00891 | EXPERIENCED | FRED       | HOWL      | F11  | 9-Apr-1976  | \$59,594 | 00012  |
| 32432 | TRAINEE     | THOMAS     | SCHWEIK   | F11  | 7-Nov-1981  | \$26,723 | 00891  |
| 38462 | EXPERIENCED | BILL       | SWAY      | T32  | 5-May-1980  | \$54,000 | 00012  |
| 39485 | EXPERIENCED | DEE        | TERRICK   | D98  | 2-May-1977  | \$55,829 | 00012  |
| 48475 | EXPERIENCED | GAIL       | CASSIDY   | E46  | 2-May-1978  | \$55,407 | 00012  |
| 78923 | EXPERIENCED | LYDIA      | HARRISON  | F11  | 19-Jun-1979 | \$40,747 | 00891  |
| 87289 | EXPERIENCED | LOUISE     | DEPALMA   | G20  | 28-Feb-1979 | \$57,598 | 00012  |
| 87465 | EXPERIENCED | ANTHONY    | IACOBONE  | C82  | 2-Jan-1973  | \$58,462 | 00012  |
| 87701 | TRAINEE     | NATHANIEL  | CHONTZ    | F11  | 28-Jan-1982 | \$24,502 | 00891  |

A similar DATATRIEVE search can be done in BATCH mode by creating a batch job file either with the CREATE command or EDITOR. The following DCL file is developed to search for personnel as above with supervisor's ID equal to 891 and 12. The batch job file is called "SUP.COM" with the following code:

```

$DTR
READY PERSONNEL
FIND PERSONNEL WITH SUP-ID=891,12
OPEN LIST.LOG
PRINT CURRENT SORTED BY LAST-NAME,SUP-ID
CLOSE
EXIT

```

Then using the batch processors as follows:

```
$ SUBMIT/NOLOG_FILE SUP
```

When the batch processor is finished a print file named "LIST.LOG" is created. This can be printed by using DCL TYPE or PRINT command.

```
$TYPE LIST.LOG
```

| ID    | STATUS      | FIRST NAME | LAST NAME | DEPT | START DATE  | SALARY   | SUP ID |
|-------|-------------|------------|-----------|------|-------------|----------|--------|
| 48475 | EXPERIENCED | GAIL       | CASSIDY   | E46  | 2-May-1978  | \$55,407 | 00012  |
| 87701 | TRAINEE     | NATHANIEL  | CHONTZ    | F11  | 28-Jan-1982 | \$24,502 | 00891  |
| 87289 | EXPERIENCED | LOUISE     | DEPALMA   | G20  | 28-Feb-1979 | \$57,598 | 00012  |
| 78923 | EXPERIENCED | LYDIA      | HARRISON  | F11  | 19-Jun-1979 | \$40,747 | 00891  |
| 00891 | EXPERIENCED | FRED       | HOWL      | F11  | 9-Apr-1976  | \$59,594 | 00012  |
| 87465 | EXPERIENCED | ANTHONY    | IACOBONE  | C82  | 2-Jan-1973  | \$58,462 | 00012  |
| 32432 | TRAINEE     | THOMAS     | SCHWEIK   | F11  | 7-Nov-1981  | \$26,723 | 00891  |
| 00012 | EXPERIENCED | CHARLOTTE  | SPIVA     | TOP  | 12-Sep-1972 | \$75,892 | 00012  |
| 38462 | EXPERIENCED | BILL       | SWAY      | T32  | 5-May-1980  | \$54,000 | 00012  |
| 39485 | EXPERIENCED | DEE        | TERRICK   | D98  | 2-May-1977  | \$55,829 | 00012  |

Another way to create the collection would be to replace the above RSE in SUP.COM with a reference to a table. The use of tables in this way is somewhat like a shipping list in a warehouse, where an operator has a list of items which he uses to pick the items to be shipped. An example of a DICTIONARY table

```

DTR> SHOW DEPT-TBL
TABLE DEPT TBL
QUERY_HEADER IS "SUPERVISOR"/"NAME"
"891" : "John Smith"
"12" : "Ray Jones"
END_TABLE

```

Using the interactive command, the following RSE will find all the personnel with the selected supervisor ID in the first position of the table. You note there is no boolean expression when a table does the selecting. The key word in the RSE is "IN". It is this word which causes the search to select only those supervisor's ID which are in the table.

```

DTR> FIND PERSONNEL WITH SUP-ID IN DEPT-TBL
[10 records found]

```

DTR> PRINT ALL

| ID    | STATUS      | FIRST<br>NAME | LAST<br>NAME | DEPT | START<br>DATE | SALARY   | SUP<br>ID |
|-------|-------------|---------------|--------------|------|---------------|----------|-----------|
| 00012 | EXPERIENCED | CHARLOTTE     | SPIVA        | TOP  | 12-Sep-1972   | \$75,892 | 00012     |
| 00891 | EXPERIENCED | FRED          | HOWL         | F11  | 9-Apr-1976    | \$59,594 | 00012     |
| 32432 | TRAINEE     | THOMAS        | SCHWEIK      | F11  | 7-Nov-1981    | \$26,723 | 00891     |
| 38462 | EXPERIENCED | BILL          | SWAY         | T32  | 5-May-1980    | \$54,000 | 00012     |
| 39485 | EXPERIENCED | DEE           | TERRICK      | D98  | 2-May-1977    | \$55,829 | 00012     |
| 48475 | EXPERIENCED | GAIL          | CASSIDY      | E46  | 2-May-1978    | \$55,407 | 00012     |
| 78923 | EXPERIENCED | LYDIA         | HARRISON     | F11  | 19-Jun-1979   | \$40,747 | 00891     |
| 87289 | EXPERIENCED | LOUISE        | DEPALMA      | G20  | 28-Feb-1979   | \$57,598 | 00012     |
| 87465 | EXPERIENCED | ANTHONY       | IACOBONE     | C82  | 2-Jan-1973    | \$58,462 | 00012     |
| 87701 | TRAINEE     | NATHANIEL     | CHONTZ       | F11  | 28-Jan-1982   | \$24,502 | 00891     |

Tables can also be used to form a relationship between one field and another. By using a VIA in the RSE, you can obtain a list of personnel and their supervisor by name rather than number.

DTR> PRINT ALL FIRST-NAME, LAST-NAME, SUP-ID VIA DEPT-TBL

| FIRST<br>NAME | LAST<br>NAME | SUPERVISOR<br>NAME |
|---------------|--------------|--------------------|
| CHARLOTTE     | SPIVA        | Ray Jones          |
| FRED          | HOWL         | Ray Jones          |
| THOMAS        | SCHWEIK      | John Smith         |
| BILL          | SWAY         | Ray Jones          |
| DEE           | TERRICK      | Ray Jones          |
| GAIL          | CASSIDY      | Ray Jones          |
| LYDIA         | HARRISON     | John Smith         |
| LOUISE        | DEPALMA      | Ray Jones          |
| ANTHONY       | IACOBONE     | Ray Jones          |
| NATHANIEL     | CHONTZ       | John Smith         |

However, the DICTIONARY table can't be used to bridge between interactive and batch DATATRIEVE and is only presented to illustrate how tables are used. There is another type of table called a DOMAIN. This is the table type to use as the bridge between interactive and batch DATATRIEVE. The most important feature of the DOMAIN table is it can be used with any DATATRIEVE file. This is the key to how to develop the bridge system. However, it requires more DATATRIEVE code. To create a DOMAIN table in this example for SUP-ID, do the following:

1. DEFINE A DOMAIN

```
DEFINE DOMAIN SEARCH USING SEARCH-REC
ON DEPT.DAT;
```

2. DEFINE A RECORD

```
DEFINE RECORD SEARCH-REC USING
01 SEARCH-REC.
03 SUP-AREA PIC 9(5).
03 AREA PIC X(4).
03 SUP-NAME PIC X(15).
;
```

3. DEFINE A TABLE

```
DEFINE TABLE SEARCH-TABLE FROM DOMAIN SEARCH
SUP-AREA : AREA
ELSE ""
END-TABLE
```



4. DEVELOP A DATATRIEVE PROCEDURE FOR ENTERING THE SEARCH LIST

```
DEFINE PROCEDURE DEMO
DEFINE FILE SEARCH SUPERSEDE
READY SEARCH WRITE
DECLARE NEW_SUP PIC 9(5).
DECLARE NEW_AREA PIC X(4).
REPEAT 100 BEGIN
NEW_SUP=*. "supervisor's number"
NEW_AREA=1
IF NEW_SUP EQ 0 THEN ABORT
IF NEW_SUP NE 0 THEN
BEGIN
 STORE SEARCH USING
 BEGIN
 SUP_AREA=NEW_SUP
 AREA=NEW_AREA
 END
END
END_PROCEDURE
```

5. DEVELOP A DCL COMMAND FILE(ENTRY)

```
CREATE A FILE NAMED NEWSUP.COM

$ COPY SYS$INPUT SYS$COMMAND
 THIS PROGRAM CREATES A LIST OF
 PERSONNEL WHO WORK FOR A SELECTED
 SUPERVISOR

$ASSIGN/USER_MODE SYS$COMMAND SYS$INPUT
$DTR;EXECUTE DEMO
$SUBMIT/NOLOG_FILE SUP
$EXIT
```

6. CREATE A DCL FILE (OUTPUT)

```
$DTR
READY PERSONNEL
FIND PERSONNEL WITH SUP-ID IN SEARCH-TABLE
OPEN LIST.LOG
PRINT ALL
CLOSE
EXIT
```

To illustrate how the file NEWSUP operates type the following command:

```
$@NEWSUP
```

The output is as follows:

```
THIS PROGRAM CREATES A LIST OF
PERSONNEL WHO WORK FOR A SELECTED SUPERVISOR
```

```
Enter supervisor's number: 87289
Enter supervisor's number: 39485
Enter supervisor's number: 0
 JOB entered on queue SYS$BATCH
$
```

In step 4, there is a boolean test for zero. It is this zero which is used to exit the entry phase. An output can be obtained by \$T LIST.LOG

DTR> PRINT ALL

| ID    | STATUS      | FIRST<br>NAME | LAST<br>NAME | DEPT | START<br>DATE | SALARY   | SUP<br>ID |
|-------|-------------|---------------|--------------|------|---------------|----------|-----------|
| 02943 | EXPERIENCED | CASS          | TERRY        | D98  | 2-Jan-1980    | \$29,908 | 39485     |
| 34456 | TRAINEE     | HANK          | MORRISON     | T32  | 1-Mar-1982    | \$30,000 | 87289     |
| 48573 | TRAINEE     | SY            | KELLER       | T32  | 2-Aug-1981    | \$31,546 | 87289     |
| 49843 | TRAINEE     | BART          | HAMMER       | D98  | 4-Aug-1981    | \$26,392 | 39485     |
| 83764 | EXPERIENCED | JIM           | MEADER       | T32  | 4-Apr-1980    | \$41,029 | 87289     |
| 84375 | EXPERIENCED | MARY          | NALEVO       | D98  | 3-Jan-1976    | \$56,847 | 39485     |
| 88001 | EXPERIENCED | DAVID         | LITELLA      | G20  | 11-Nov-1980   | \$34,933 | 87289     |
| 91023 | TRAINEE     | STAN          | WITTGEN      | G20  | 23-Dec-1981   | \$25,023 | 87289     |

When running the example, it is possible to search the personnel data file for any employees who work for a selected supervisor. The number of supervisors is a variable from 1 to 100. In a DOMAIN table the number of fields is not restricted to two and the search field doesn't have to be the first field in the data file. Thus in designing the search file structure, the file can be used with other defined tables for other purposes as described in CHAPTER 9 OF THE DATATRIEVE USERS GUIDE.

There is another way to bridge between interactive and batch when a single value in the search field is desired. As an example, in the personnel data base there is interest in STATUS field for a single selection ie("TRAINEE" OR "EXPERIENCED"). The set up is as follows:

```
1. DEFINE A DOMAIN
 DEFINE DOMAIN SINGLE-SEARCH USING ONE-REC
 ON ONE.DAT;
2. DEFINE A RECORD
 DEFINE RECORD ONE_REC
 01 ONE_REC.
 03 ONE_STATUS PIC 9.
 ;
3. DEFINE A TABLE
 DEFINE TABLE STATUS_TABLE
 "EXPERIENCED" : 1
 "TRAINEE" : 2
 END TABLE
4. DEFINE DATATRIEVE PROCEDURE (ENTRY)
 DEFINE PROCEDURE DEMO2
 FINISH
 DEFINE FILE SINGLE_SEARCH SUPERSEDE
 READY SINGLE_SEARCH WRITE
 DECLARE SINGLE PIC X(11).
 SINGLE=*. "STATUS CATEGORY"
 STORE SINGLE_SEARCH USING
 BEGIN
 ONE_STATUS=SINGLE VIA STATUS_TABLE
 END
 END-PROCEDURE
5. DEFINE DATATRIEVE PROCEDURE (OUTPUT)
 DEFINE PROCEDURE DEMO3
 READY SINGLE_SEARCH
 FIND VALUE IN SINGLE_SEARCH
 READY PERSONNEL
 SELECT VALUE
 FIND PERSONNEL
 WITH STATUS VIA STATUS_TABLE=ONE_STATUS
 OPEN LIST.LOG
 PRINT CURRENT SORTED LAST_NAME
 CLOSE
 END-PROCEDURE
```

```

6. CREATE DCL ENTRY FILE
 $COPY SYS$INPUT SYS$COMMAND
 THIS DEMONSTRATES A SEARCH FOR A SINGLE
 SEARCH PARAMETER

 $ ASSIGN/USER_MODE SYS$COMMAND SYS$INPUT
 $ DTR;EXECUTE DEMO2
 $ SUBMIT/NOLOG_FILE DEMO5
 $ EXIT

7. CREATE A DCL RUNNING FILE
 $ DTR;EXECUTE DEMO3
 $ EXIT

8. TO RUN THE SEARCH
 $ @DEMO4

```

Enter STATUS CATEGORY: TRAINEE

```

 $ JOB entered on queue SYS$BATCH
9. TO PRINT THE OUTPUT
 $ T LIST.LOG

```

| ID    | STATUS  | FIRST NAME | LAST NAME | DEPT | START DATE  | SALARY   | SUP ID |
|-------|---------|------------|-----------|------|-------------|----------|--------|
| 87701 | TRAINEE | NATHANIEL  | CHONTZ    | F11  | 28-Jan-1982 | \$24,502 | 00891  |
| 49843 | TRAINEE | BART       | HAMMER    | D98  | 4-Aug-1981  | \$26,392 | 39485  |
| 48573 | TRAINEE | SY         | KELLER    | T32  | 2-Aug-1981  | \$31,546 | 87289  |
| 34456 | TRAINEE | HANK       | MORRISON  | T32  | 1-Mar-1982  | \$30,000 | 87289  |
| 32432 | TRAINEE | THOMAS     | SCHWEIK   | F11  | 7-Nov-1981  | \$26,723 | 00891  |
| 12643 | TRAINEE | JEFF       | TASHKENT  | C82  | 4-Apr-1981  | \$32,918 | 87465  |
| 91023 | TRAINEE | STAN       | WITTGEN   | G20  | 23-Dec-1981 | \$25,023 | 87289  |

This example presents some unique problems in developing the RSE because the status field is not numeric and therefore a table transition is required to convert this field to a numerical value. If the search field were numeric, the RSE in STEP 5 would be FIND PERSONNEL WITH STATUS=ONE-STATUS and the rest of the steps would not have to reference a table. Another requirement is the SELECT VALUE statement in step 5. Without this SELECT, an error occurs because there is no way to reference the field ONE-STATUS.

There is another technique which is used to help reduce the search time with interactive DATATRIEVE. This is to create a sub-file (small) of the large file. The creation of the small file is done in Batch mode using one or both of the techniques previously illustrated. An example of the code is as follows:

```

DTR> DEFINE DOMAIN NEW USING PERSONNEL-REC ON
DFN> NEW.DAT;
DTR> DEFINE FILE NEW SUPERSEDE
DTR> READY NEW WRITE
DTR> READY PERSONNEL
DTR> FIND PERSONNEL WITH DEPT="E46"
[2 records found]
DTR> NEW=CURRENT
DTR> FIND NEW
[2 records found]
DTR> PRINT ALL

```

| ID    | STATUS      | FIRST NAME | LAST NAME | DEPT | START DATE  | SALARY   | SUP ID |
|-------|-------------|------------|-----------|------|-------------|----------|--------|
| 38465 | EXPERIENCED | JOANNE     | FREIBURG  | E46  | 20-Feb-1980 | \$23,908 | 48475  |
| 48475 | EXPERIENCED | GAIL       | CASSIDY   | E46  | 2-May-1978  | \$55,407 | 00012  |

A new domain called NEW is created with the same record definition as the main (large) file. When the selection is complete and a collection generated, the collection is stored in the domain NEW. This domain NEW can then be used either interactively or in batch mode to develop reports.

With all the techniques presented it is now possible to shorten your actual terminal time to use DATATRIEVE with very large files. All or part of the examples can be used when you develop your own procedures.

B. Z. Lederman

2572 E. 22nd St.  
Brooklyn, N.Y. 11235

Abstract

This session will supply examples and suggestions which go beyond the material in the Datatrieve manuals, and show how various types of problems may be solved using the options available within the record definition. The material will include some comparisons between different approaches to the same problem, the use of VIEWS (which are created from record definitions), and methods of transferring data from one domain to another, which also depends in part upon record definitions.

It is not unreasonable to state that the record definition is the foundation of any Datatrieve application, as it is the reference by which all data is stored and retrieved. Therefore, the first rule for any application is:

KNOW YOUR APPLICATION

from which immediatly follows:

KNOW YOUR DATA

I find that the best way to work out record definitions is with two very simple pieces of equipment: a pencil, and a piece of paper marked off in squares such as graph paper, or a printer form layout sheet, or CRT display form, or an old coding sheet. By marking off the fields, using one square per byte, the number of data items, the length of each field, and it's alignment and relationship to other fields are easily determined. This is especially important with the REDEFINES clause, which will be shown later.

Often, an unsuspected benefit of taking an existing manual operation and implementing it on Datatrieve is that the people involved must sit down and figure out exactly what pieces of data they are dealing with, and in what manner: this is often the first time anyone actually does this, and they are often suprised by the amount of data involved.

Some applications move onto Datatrieve almost automatically. If you are using a pre-printed form (and almost every company has some sort of printed form for orders, absence reports, pencil requisitions, etc.) then one can simply copy the fields into the record definition: there are cases of new users moving applications like this in one day. If your records are not as well organized, then you must analyze them

yourself. If you are using Datatrieve to read an existing file created by some other program, then it is necessary to obtain the file record layout and follow it.

Keep in mind that, while it is nice to get a good record definition at the beginning, it is always possible to define a new domain and record and read the data from the old domain to a new one, so if you have 10,000 records stored, and find you need to add a field, don't panic. Examples of this will be given.

While one could use ADT or follow the simple examples in the manuals and develop many useful applications with Datatrieve, there is a much wider range of applications which may be addressed with a few simple techniques. For example, suppose the YACHTS domain was being used in a show room, where the customers are allowed to look up data at a terminal, but the seller doesn't want the price to appear. Using the simplified record definition for YACHTS, here are two possible solutions.

| Original<br>Definition | Second<br>Definition  | View                               |
|------------------------|-----------------------|------------------------------------|
| 01 BOAT.               | 01 BOAT.              | DEFINE DOMAIN LOOK OF YACHTS USING |
| 06 BUILDER PIC X(10).  | 06 BUILDER PIC X(10). | 01 LOOK OCCURS FOR YACHTS.         |
| 06 MODEL PIC X(10).    | 06 MODEL PIC X(10).   | 03 BUILDER FROM YACHTS.            |
| 03 SPEC.               | 03 SPEC.              | 03 MODEL FROM YACHTS.              |
| 06 RIG PIC X(6).       | 06 RIG PIC X(6).      | 03 RIG FROM YACHTS.                |
| 06 LOA PIC XXX.        | 06 LOA PIC XXX.       | 03 LOA FROM YACHTS.                |
| 06 DISP PIC 9(5).      | 06 DISP PIC 9(5).     | 03 DISP FROM YACHTS.               |
| 06 BEAM PIC 99.        | 06 BEAM PIC 99.       | 03 BEAM FROM YACHTS.               |
| 06 PRICE PIC 99.       | 06 FILLER PIC XX.     | ;                                  |
| ;                      | ;                     |                                    |

The definition on the left is for the whole domain which the showroom owner will use, and is the short definition given in the manual in the optimization chapter. PDP-11 users who are short of pool space should look at this chapter, and compare the short definition, which uses much less pool space by having fewer clauses, with the definition created by the installation package. The customers could use the view on the right, which does not have the price, or a second domain using the record definition in the center could be used to access the SAME FILE as is used for YACHTS. This shows two useful features. First, it is possible to have more than one domain access the data in a single file: this makes it possible to look at the data in more than one way, with more than one record definition. The only restriction is that the user must document the domains accessing each file so that, if it is ever necessary to change a file, the domains to be affected will be known. There is also an important difference between the domains and the view: you cannot store or erase records in a view, but you can do all operations on the second domain. The limitation on views can be bad or good, depending upon the application: in this example, you probably would not want customers to add or erase records, so using a VIEW would be one way of preventing this.

The second domain shows the use of the special field type FILLER to "skip" over data in a record. The data is still there, and may be accessed by the original domain, but not by the second domain: if you also protect the record definition itself to be execute but not read, the user will never see the filler field (it doesn't appear in SHOW FIELDS), and will not know there is data there. This may be extended for use in 'hiding' fields.

```

01 CUSTOMER.
 03 ADDRESS.
 06 STREET PIC X(10).
 06 CITY PIC X(10).
 06 STATE PIC X(2).
 06 ZIP PIC 9(5).
 03 FILL-ENG.
 06 FILLER PIC X(10).
 03 ENG REDEFINES FILL-ENG.
 06 ENGINEER PIC X(10).
;

```

If you normal command PRINT you get:

```

STREET CITY STATE ZIP
130 LIBERTY NEW YORK NY 10006

```

but if you say PRINT ADDRESS, ENG the result is:

```

STREET CITY STATE ZIP ENGINEER
130 LIBERTY NEW YORK NY 10006 LEDERMAN

```

Thus the ENGINEER field is always available, but will not print out unless specifically asked for. This can also be a pool saving technique for very large records, or may be used to control access to information where several users must access the same file by doing something like this:

```

01 ALL-DEPT-REC. 01 DEPT-A-REC. 02 DEPT-B-REC.
 03 DEPT-A. 03 DEPT-A. 03 FILLER PIC X(5).
 06 BUDGET PIC 999. 06 BUDGET PIC 999.
 06 MANAGER PIC XX. 06 MANAGER PIC XX.
 03 DEPT-B. 03 FILLER PIC X(10). 03 DEPT-B.
 06 BUDGET PIC 999. ; 06 BUDGET PIC 999.
 06 MANAGER PIC XX. ; 06 MANAGER PIC XX.
 03 DEPT-C. ; 03 FILLER PIC X(5).
 06 BUDGET PIC 999. ; ;
 06 MANAGER PIC XX. ; ;
;

```

This is a very small example, but it shows how a single file may be accessed by one domain having access to all fields, and by several other domains, accessing only some of the data. Each of the smaller domains can read and write only their own data, and the big domain could be used for report giving all of the data. This is an alternative to having separate domains for each department, and using a view to tie them together for reports. (For PDP-11 and PRO users, each of the smaller record definitions uses less pool than the big definition, allowing more complicated procedures to be used (or more sort space, etc.). If a record definition is very large (hundreds of bytes), then the only way to access it usefully in Datatrieve-11 may be to have more than one domain each access a portion of the record.

Another approach to the same problem would be to use a VIEW. First, a definition may be given for the domain which holds data for all departments.

```

01 BUDGET-REC.
 10 DEPARTMENT PIC XX.
 10 PROJECT PIC X(10).
 10 AMOUNT PIC 9(6)V99
 EDIT-STRING $$$$,$$$$.00.
 10 MANAGER PIC X(10).
;

```

The person in charge of budgets would have full access to this domain, and so could access all of the data.

Each individual department would have their own VIEW defined like this:

```

DEFINE DOMAIN AA-BUDGET OF BUDGET USING
01 AA-BUDGET OCCURS FOR BUDGET WITH
 DEPARTMENT = "AA".
 10 PROJECT FROM BUDGET.
 10 AMOUNT FROM BUDGET.
 10 MANAGER FROM BUDGET.
;

```

This definition should be protected so that the department can execute it, but not read or modify it, otherwise they might want to change the definition to allow access to other departments. Because the selection criteria is fixed, they will see only their own department's data. This configuration would be of greatest use when the different departments must read the information, but only the central controller will enter or erase it.

Another very useful feature of the REDEFINES clause is that it allows one to look at the data in a domain in more than one way within a single domain. An application for this could be a file which has more than one record type in a single file. The author does not recommend this for new applications, but there may be existing files set up like this (COBOL and RPG are often the source) which one would like to access with Datatrieve. Consider a file with data that looks like this:

Key Type

```

0001 N Lederman Bart Z
0001 A 2572 E 22nd New York NY 11235
0001 P 38 DPG 2222 Distributed Proc
0001 T 212-250-2300 718-555-5555

```

A possible record definition is:

```

DEFINE RECORD MULTI-REC
01 MULTI-REC.
 03 KEY PIC 9999 EDIT-STRING ZZZ9.
 03 TYPE PIC X.
 03 NAME-PAGE.
 06 LAST PIC X(14).
 06 FIRST PIC X(12).
 06 M PIC X.
 06 N PIC X.
 03 ADDRESS-PAGE REDEFINES NAME-PAGE.
 06 STREET PIC X(11).
 06 CITY PIC X(10).
 06 STATE PIC XX.
 06 ZIP PIC 99999.
 03 PERSONNEL-PAGE REDEFINES NAME-PAGE.
 06 FLOOR PIC 99 EDIT-STRING Z9.
 06 SECTION PIC XXX.
 06 CLOCK PIC 9999.
 06 DEPARTMENT PIC X(19).
 03 TELEPHONE-PAGE REDEFINES NAME-PAGE.
 06 BUSINESS PIC X(10)
 EDIT-STRING XXX-XXX-XXXX.
 06 HOME PIC X(10)
 EDIT-STRING XXX-XXX-XXXX.
 06 FILLER PIC X(8).
;

```

The first part of the definition is for the fields which do not change (the key and the type, which are common to all records). The next part is the definition for the first record, the name fields. Because this is the first definition (highest in the hierarchy), it is used by default when accessing the data. If the data is printed, the result is

```

KEY TYPE LAST FIRST M N
1 N Lederman Bart Z
1 A 2572 E 22ndNew York NY112 3 5
1 P 38DPG2222Distr ibuted Pr o c c
1 T 21225023002125 555555

```

Notice how all records have been printed as if they were NAME-PAGE as this is the first group in the hierarchy, but because I have the redefined fields, I can also access the data in different ways. Each redefines has it's own group name, which makes access much easier as I can specify a group name for one whole page, and note that a redefines must never be longer than the original field and/or group. In other applications, you should be certain the longest group comes first, or that you fill the first group to be as long as the longest group. It is acceptable for the redefines to be shorter than the original field, but I prefer to fill all groups in for my own reference. In this case, the telephone data is shorter, and the additional length is made up with FILLER. With the redefines, a simple procedure will access the data correctly.

```

DEFINE PROCEDURE PRINT-MULTI
READY MULTI
FOR MULTI BEGIN
 IF TYPE EQ "N" PRINT NAME-PAGE
 IF TYPE EQ "A" PRINT ADDRESS-PAGE
 IF TYPE EQ "P" PRINT PERSONNEL-PAGE
 IF TYPE EQ "T" PRINT TELEPHONE-PAGE
END
END-PROCEDURE

```

When this procedure is invoked, the data prints like this:

```

LAST FIRST M N
Lederman Bart Z
STREET CITY STATE ZIP
2572 E 22nd New York NY 11235
FLOOR SECTION CLOCK DEPARTMENT
38 DPG 2222 Distributed Proc
BUSINESS HOME
212-250-2300 718-555-5555

```

If you have more than one set of records, the following sets will not have headers when they print out (this is the normal way the Datatrieve PRINT command behaves) but the data will print out using the correct field definitions. The same technique may be used to select the proper page for storing, and so on. Individual fields of each page may be accessed simply by using the name of the field, at any time: Datatrieve will go through the record hierarchy, as it would for any other domain, to resolve the field references.

An alternative to the procedure is to define two separate domains for the data.

```

DEFINE RECORD BASE-REC
01 BASE.
 03 KEY PIC 9999 EDIT-STRING ZZZ9.
 03 TYPE PIC X.
 03 LAST PIC X(14).
 03 FIRST PIC X(12).
 03 M PIC X.
 03 N PIC X.
;
DEFINE DOMAIN BASE USING BASE-REC
ON MULTI.SEQ;

DEFINE RECORD OTHER-REC
01 OTHER.
 03 KEY PIC 9999 EDIT-STRING ZZZ9.
 03 TYPE PIC X.
 03 ADDRESS-PAGE.
 06 STREET PIC X(11).
 06 CITY PIC X(10).
 06 STATE PIC XX.
 06 ZIP PIC 99999.
 03 PERSONNEL-PAGE REDEFINES
ADDRESS-PAGE.
 06 FLOOR PIC 99 EDIT-STRING Z9.
 06 SECTION PIC XXX.
 06 CLOCK PIC 9999.
 06 DEPARTMENT PIC X(19).
 03 TELEPHONE-PAGE REDEFINES
ADDRESS-PAGE.
 06 BUSINESS PIC X(10)
 EDIT-STRING XXX-XXX-XXXX.
 06 HOME PIC X(10)
 EDIT-STRING XXX-XXX-XXXX.
;
DEFINE DOMAIN OTHER USING OTHER-REC
ON MULTI.SEQ;

```

Now, I can define a view of these two domains to bring all of the separate records together into what will look like one single record:

```

DEFINE DOMAIN MUL OF BASE, OTHER USING
01 MULT OCCURS FOR BASE WITH TYPE EQ "N".
 06 LAST FROM BASE.
 06 FIRST FROM BASE.
 06 M FROM BASE.
 06 N FROM BASE.
03 ADDRESS-PAGE OCCURS FOR OTHER WITH
TYPE EQ "A" AND OTHER.KEY=BASE.KEY.
 06 STREET FROM OTHER.
 06 CITY FROM OTHER.
 06 STATE FROM OTHER.
 06 ZIP FROM OTHER.
03 PERSONNEL-PAGE OCCURS FOR OTHER WITH
TYPE EQ "P" AND OTHER.KEY=BASE.KEY.
 06 FLOOR FROM OTHER.
 06 SECTION FROM OTHER.
 06 CLOCK FROM OTHER.
 06 DEPARTMENT FROM OTHER.
03 TELEPHONE-PAGE OCCURS FOR OTHER WITH
TYPE EQ "T" AND OTHER.KEY=BASE.KEY.
 06 BUSINESS FROM OTHER.
 06 HOME FROM OTHER.
;

```

The reason for the two domains is to be able to use the key field to tie together the appropriate separate records. The BASE domain will occur once for each group of associated records, and the KEY field will be used to retrieve the other records of the same group. When this view is readied and printed, it looks like this (I have added a second set of data records to show that the view works properly):

| LAST              | FIRST | M N          | STREET      |
|-------------------|-------|--------------|-------------|
| Lederman          | Bart  | Z            | 2572 E 22nd |
| New York          | NY    | 11235 38     | DPG 2222    |
| Distributed Proc  |       | 212-250-2300 |             |
|                   |       | 212-555-5555 |             |
| Hackinbush        | Hugo  | Z            | 11 Julius   |
| Hialeah           | FL    | 33999 13     | MRX 1313    |
| Sales & Promotion |       | 305-555-3131 |             |
|                   |       | 305-555-1476 |             |

It appears wrapped around here as there are only 44 columns on this page, but on 132 column paper, all fields print out with their headers. This view would be very useful in "flattening" the data record so it could be processed as other domains are.

Another use of the redefines can be for break fields.

```

DEFINE RECORD TTN-REC
01 TTN.
 03 PORT PIC 999.
 03 BREAK REDEFINES PORT.
 06 B1 PIC 99.
 06 FILLER PIC X.
 03 GROUP PIC 999.
 03 SWITCH PIC 9.
 03 TRUNK PIC 9999.
 03 COMMENTS PIC X(21).
;

```



Although PORT is an integer number, I am using the DISPLAY data type so I can redefine it as a two digit field. The reason can be seen when reporting the data.

```

DEFINE PROCEDURE RPT-TTN
READY TTN
REPORT TTN ON TTN.RPT
SET REPORT-NAME="Show Breaks with Redefines"
SET COLUMNS-PAGE=50
PRINT COL 1, PORT, COL 8, GROUP, COL 20,
 SWITCH, COL 24, TRUNK, COL 40, COMMENTS
AT BOTTOM OF B1 PRINT COL 1,
"-----"
END-REPORT
END-PROCEDURE

```

When this procedure is invoked, the resulting report is:

Show Breaks with Redefines

Page 1

| PORT  | GROUP | SWITCH | COMMENTS |
|-------|-------|--------|----------|
| 040   | 347   | 4 1154 | 88/89    |
| 041   | 347   | 4 1155 | 88/89    |
| 042   | 347   | 4 1156 | 88/89    |
| 043   | 347   | 4 1157 | 88/89    |
| 044   | 347   | 4 1417 | 88/89    |
| 045   | 347   | 4 1440 | 88/89    |
| 046   | 347   | 5 4521 | 88/89    |
| 047   | 347   | 5 4522 | 88/89    |
| ----- |       |        |          |
| 050   | 347   | 5 4523 | 88/89    |
| 052   | 131   | 7 5311 | TYPE0    |
| 053   | 131   | 7 5312 | TYPE0    |
| 054   | 131   | 7 5313 | TYPE0    |
| 055   | 131   | 7 5314 | TYPE0    |
| 056   | 131   | 7 5315 | TYPE0    |
| 057   | 131   | 7 5316 | TYPE0    |
| ----- |       |        |          |
| 060   | 131   | 7 5317 | TYPE0    |
| 061   | 131   | 7 5320 | TYPE0    |
| 062   | 131   | 7 5321 | TYPE0    |
| ----- |       |        |          |

As may be seen, by having a field which acts on the first two digits of PORT, it is possible to put in a break line every 'n' entries, something which would be difficult otherwise. (Incidentally, the ports are numbered in octal, which is why there is no port 048 or 049, but the system works just as well in decimal.) Not shown is a LINES-PAGE command: if you want groups of records to print out on successive pages with the breaks aligned, you will have to set the number of lines per page to match the group breaks. It may be noted that there is no SORTED BY clause in the report statement: it is not necessary to sort the data if it is already in the proper order, as it will be if a sequential file is reported in it's present sequence, or if an indexed file is reported in the order of it's primary key. (One of the uses for indexed files can be to keep data ordered.) If there is no SORTED BY clause, Datatrieve will issue a warning that unsorted records

are being reported, but will then report and allow a break on any field: if there is a SORTED BY clause, then only fields which were sorted can have breaks, and in this case I want the report in the order of PORT, not the order of B1. Reports also come out faster if you don't have to sort the domain first, and exhaustion of sort pool space is avoided.

A commonly used feature is variable length records, as in the FAMILIES domain. While certainly useful, variable length records have some drawbacks, including more difficult access to the fields in the variable length portion, and having to know the maximum number of variable occurrences when defining the domain. There is an alternative approach using two files and a view. First, for comparison, is a shortened record definition for the FAMILY domain.

```

DEFINE RECORD SHORT-FAMILY-REC
01 FAMILY.
 03 FATHER PIC X(10).
 03 MOTHER PIC X(10).
 03 NUMBER-KIDS PIC 99 EDIT-STRING Z9.
 03 KIDS OCCURS 0 TO 10 TIMES
 DEPENDING ON NUMBER-KIDS.
 06 KID PIC X(10).
 06 AGE PIC 99 EDIT-STRING Z9.
;

```

```

DEFINE DOMAIN FAMILY USING
SHORT-FAMILY-REC ON FAMILY.DAT;

```

The alternative uses one domain for the fixed data, and one for the variable occurrence data, with one field in common to tie the two together.

```

DEFINE RECORD PARENT-REC
01 PARENT.
 03 KEY PIC 999 EDIT-STRING ZZ9.
 03 FATHER PIC X(10).
 03 MOTHER PIC X(10).
;

```

```

DEFINE RECORD OFFSPRING-REC
01 OFFSPRING.
 03 KEY PIC 999 EDIT-STRING ZZ9.
 03 KID PIC X(10).
 03 AGE PIC 99 EDIT-STRING Z9.
;

```

```

DEFINE DOMAIN PARENT USING
PARENT-REC ON PARENT.DOM;

DEFINE FILE FOR PARENT KEY=KEY(NO DUP);

DEFINE DOMAIN OFFSPRING USING
OFFSPRING-REC ON OFFSPRING.DOM;

DEFINE FILE FOR OFFSPRING KEY=KEY(DUP);

```

The two domains are then connected with a view:

```

DEFINE DOMAIN HOUSEHOLD OF PARENT,
OFFSPRING USING
01 HOUSEHOLD OCCURS FOR PARENT.
03 FATHER FROM PARENT.
03 MOTHER FROM PARENT.
03 KIDS OCCURS FOR OFFSPRING WITH
OFFSPRING.KEY EQ PARENT.KEY.
06 KID FROM OFFSPRING.
06 AGE FROM OFFSPRING.
;

```

When printed, the HOUSEHOLD domain looks just like the FAMILY domain, without the NUMBER-KIDS field.

| FATHER   | MOTHER   | KID        | AGE |
|----------|----------|------------|-----|
| JIM      | ANN      | URSULA     | 7   |
|          |          | RALPH      | 3   |
| JIM      | LOUISE   | ANNE       | 31  |
|          |          | JIM        | 29  |
|          |          | ELLEN      | 26  |
|          |          | DAVID      | 24  |
|          |          | ROBERT     | 16  |
| JOHN     | JULIE    | ANN        | 29  |
|          |          | JEAN       | 26  |
| JOHN     | ELLEN    | CHRISTOPHR | 0   |
| ARNIE    | ANNE     | SCOTT      | 2   |
|          |          | BRIAN      | 0   |
| SHEARMAN | SARAH    | DAVID      | 0   |
| TOM      | ANNE     | SUZIE      | 6   |
|          |          | PATRICK    | 4   |
| BASIL    | MERIDETH | BEAU       | 28  |
|          |          | BROOKS     | 26  |
|          |          | ROBIN      | 24  |
|          |          | JAY        | 22  |
|          |          | JILL       | 20  |
|          |          | WREN       | 17  |
| ROB      | DIDI     |            |     |
| JEROME   | RUTH     | ERIC       | 32  |
|          |          | CISSY      | 24  |
|          |          | NANCY      | 22  |
|          |          | MICHAEL    | 20  |
| TOM      | BETTY    | MARTHA     | 30  |
|          |          | TOM        | 27  |
| GEORGE   | LOIS     | FRED       | 26  |
|          |          | JEFF       | 23  |
|          |          | LAURA      | 21  |
| HAROLD   | SARAH    | HAROLD     | 35  |
|          |          | CHARLIE    | 31  |
|          |          | SARAH      | 27  |
| EDWIN    | TRINITA  | ERIC       | 16  |
|          |          | SCOTT      | 11  |

This view can be readied, printed, reported, examined, etc. just like the FAMILY domain, but there are several important differences. First, because they are two separate domains, it is not necessary to know how many occurrences there are for the variable portion, and there is no limit to the number of variable records. Also, because there are two separate domains, it is possible to work on each portion separately, and protect each portion separately. We had an application where the fixed portion was the basic information on a communications circuit (the customer name, location, date due, and so on), and the variable portion

was a record entered each time a workman attended to the circuit: the domains were protected so the workmen could read the fixed portion but never had write or modify access to it, but could write to the variable domain. This mixed protection cannot be done on a single domain with variable occurs. Another benefit is in Datatrieve-11, where pool space may be saved by readying only the portion required for a given application, rather than always having to use the larger single domain.

One drawback is that one cannot STORE to a view, but a simple procedure solves this:

```

DEFINE PROCEDURE STORE-FAMILY
DECLARE PROMPT PIC X.
DECLARE KEY-FIELD PIC 9999.
READY PARENT WRITE
READY OFFSPRING WRITE
KEY-FIELD = MAX KEY OF PARENT
KEY-FIELD = KEY-FIELD + 1
WHILE *."Y to store a family" CONT "Y" BEGIN
 STORE PARENT USING BEGIN
 KEY = KEY-FIELD
 FATHER = *.FATHER
 MOTHER = *.MOTHER
 END
 PROMPT = *."Y if there are any kids"
 WHILE PROMPT CONT "Y" BEGIN
 STORE OFFSPRING USING BEGIN
 KEY = KEY-FIELD
 KID = *."Kid's name"
 AGE = *.AGE
 END
 PROMPT = *."Y for another kid"
 END
END
FINISH
RELEASE KEY-FIELD
RELEASE PROMPT
END-PROCEDURE

```

Note that a temporary field KEY-FIELD is used to obtain the key (either by prompting or as is done here, by obtaining the last key previously used), and this field is used to store the same value in both domains, thus insuring the fields will match. The prompting for repeats could easily be made more sophisticated than this simple example

The field KEY is used to tie the two domains together, but need not appear in the final view. I have used the name KEY for this field to emphasize the fact that this is a good application for a keyed field in an indexed file. Note the condition on matching KIDS, where the two KEY fields are matched: if KEY were not a keyed field in domain OFFSPRING, then for every record in PARENTS, EVERY RECORD IN OFFSPRING WOULD HAVE TO BE SEARCHED FOR A MATCHING FIELD. While it is possible for the variable length portion domain to be a sequential file, don't complain when it takes several hours to print out a few records. In the fixed

length portion domain, it is not absolutely necessary for the KEY field to actually be a key, but making it so takes advantage of another aspect of keyed fields, that of preventing duplicates. If there were duplicates in PARENTS, than one set of KIDS would be assigned to more than one set of PARENTS, a confusing situation to say the least. In most applications, it is desirable for each set of variable records to be assigned to one fixed record, though the use of a VIEW will allow you to have multiple associations, one more possible advantage of the VIEW over the variable occurs for some applications A similar use of key fields not shown here is the NO CHANGE attribute, which can prevent a field from being modified, regardless of what access or privileges a user has to that domain. This can be very useful in protecting data from accidental or deliberate modification.

The technique of tying two domains together with a matching field can be extended to do something else the occurs clause can not do: a domain (view) with more than one variable portion. To the previous definition, I will now add:

```

DEFINE RECORD ANIMAL-REC
01 PETS.
 03 KEY PIC 999 EDIT-STRING ZZ9.
 03 NAME PIC X(10).
 03 SPECIES PIC X(10).
;

DEFINE DOMAIN ANIMALS USING
ANIMAL-REC ON ANIMAL.DOM;

DEFINE FILE FOR ANIMALS KEY=KEY(DUP);

DEFINE DOMAIN HOUSEHOLD OF PARENT,
OFFSPRING, ANIMALS USING
01 HOUSEHOLD OCCURS FOR PARENT.
 03 FATHER FROM PARENT.
 03 MOTHER FROM PARENT.
 03 KIDS OCCURS FOR OFFSPRING WITH
 OFFSPRING.KEY EQ PARENT.KEY.
 06 KID FROM OFFSPRING.
 06 AGE FROM OFFSPRING.
 03 PETS OCCURS FOR ANIMALS WITH
 ANIMALS.KEY EQ PARENT.KEY.
 06 NAME FROM ANIMALS.
 06 SPECIES FROM ANIMALS.
;

```

The same rules about keys, etc. apply to this view. I have chosen to match the PETS with the PARENTS, but I could have added an additional field to the OFFSPRING domain if I had wanted PETS to be matched to OFFSPRING. When printed, the first few entries look like this:

| FATHER | MOTHER | KID    | AGE | NAME     | SPECIES |
|--------|--------|--------|-----|----------|---------|
| JIM    | ANN    | URSULA | 7   |          |         |
|        |        | RALPH  | 3   | GAYLORD  | CAT     |
|        |        |        |     | FREDDY   | BIRD    |
| JIM    | LOUISE | ANNE   | 31  |          |         |
|        |        | JIM    | 29  |          |         |
|        |        | ELLEN  | 26  |          |         |
|        |        | DAVID  | 24  |          |         |
|        |        | ROBERT | 16  | RAGMOMMA | DOG     |
| JOHN   | JULIE  | ANN    | 29  |          |         |
|        |        | JEAN   | 26  |          |         |

It is not necessary for every entry in the PARENTS portion to have either OFFSPRING, or ANIMALS, or both, and neither OFFSPRING nor ANIMALS require the presence of the other. However, because of the way HOUSEHOLDS was defined, there must be a PARENT record if an OFFSPRING or ANIMAL record with the same key is to appear in the view. If PETS had been matched to OFFSPRING, then an OFFSPRING would have to be present for PETS to appear

An alternative method of combining data from two domains in VAX-Datatrieve (and DTR-20) is to use the CROSS statement. Instead of the VIEW joining PARENT and OFFSPRING, I could use something like this:

```
PRINT PARENT CROSS OFFSPRING OVER KEY
```

This would give me:

| KEY | FATHER   | MOTHER   | KEY | KID        | AGE |
|-----|----------|----------|-----|------------|-----|
| 1   | JIM      | ANN      | 1   | URSULA     | 7   |
| 1   | JIM      | ANN      | 1   | RALPH      | 3   |
| 2   | JIM      | LOUISE   | 2   | ANNE       | 31  |
| 2   | JIM      | LOUISE   | 2   | JIM        | 29  |
| 2   | JIM      | LOUISE   | 2   | ELLEN      | 26  |
| 2   | JIM      | LOUISE   | 2   | DAVID      | 24  |
| 2   | JIM      | LOUISE   | 2   | ROBERT     | 16  |
| 3   | JOHN     | JULIE    | 3   | ANN        | 29  |
| 3   | JOHN     | JULIE    | 3   | JEAN       | 26  |
| 4   | JOHN     | ELLEN    | 4   | CHRISTOPHR | 0   |
| 5   | ARNIE    | ANNE     | 5   | SCOTT      | 2   |
| 5   | ARNIE    | ANNE     | 5   | BRIAN      | 0   |
| 6   | SHEARMAN | SARAH    | 6   | DAVID      | 0   |
| 7   | TOM      | ANNE     | 7   | PATRICK    | 4   |
| 7   | TOM      | ANNE     | 7   | SUZIE      | 6   |
| 8   | BASIL    | MERIDETH | 8   | BEAU       | 28  |
| 8   | BASIL    | MERIDETH | 8   | BROOKS     | 26  |
| 8   | BASIL    | MERIDETH | 8   | ROBIN      | 24  |
| 8   | BASIL    | MERIDETH | 8   | JAY        | 22  |
| 8   | BASIL    | MERIDETH | 8   | WREN       | 17  |
| 8   | BASIL    | MERIDETH | 8   | JILL       | 20  |
| 10  | JEROME   | RUTH     | 10  | ERIC       | 32  |
| 10  | JEROME   | RUTH     | 10  | CISSY      | 24  |
| 10  | JEROME   | RUTH     | 10  | NANCY      | 22  |
| 10  | JEROME   | RUTH     | 10  | MICHAEL    | 20  |
| 11  | TOM      | BETTY    | 11  | MARTHA     | 30  |
| 11  | TOM      | BETTY    | 11  | TOM        | 27  |
| 12  | GEORGE   | LOIS     | 12  | JEFF       | 23  |
| 12  | GEORGE   | LOIS     | 12  | FRED       | 26  |
| 12  | GEORGE   | LOIS     | 12  | LAURA      | 21  |
| 13  | HAROLD   | SARAH    | 13  | CHARLIE    | 31  |
| 13  | HAROLD   | SARAH    | 13  | HAROLD     | 35  |
| 13  | HAROLD   | SARAH    | 13  | SARAH      | 27  |
| 14  | EDWIN    | TRINITA  | 14  | ERIC       | 16  |
| 14  | EDWIN    | TRINITA  | 14  | SCOTT      | 11  |

By specifying the fields I want to print in the CROSS statement I could suppress the KEY fields, but I let them print out this time to show what is happening. Note that the way this particular CROSS statement was entered results in picking up only those parents who have at least one offspring. The CROSS statement can often be thought of as a short way to do a VIEW, and regarding it as such may help you see what it is doing. Note particularly what was said before about OFFSPRING having to be keyed for fast retrieval: you can see here that the second domain listed in the CROSS statement is taking the same place as the second domain in our VIEW example; therefore, it should also be keyed if the CROSS statement is to execute quickly. Just as with the view, if both domains are keyed it doesn't matter which comes first, but if only one is keyed it should given last in a CROSS statement for best results. Because the CROSS can be implemented in a single statement, it is easier to use during interactive sessions, and when you are investigating relationships between various domains and groups of data. If you find a particular relationship which you expect will be used many times, you may want to convert your CROSS into a VIEW, as this will fix the relationship and you will be able to ready the domain and print records without having to remember what the joining conditions are.

Lest it be thought that I am completely against the use of the OCCURS clause, I will now show a good use for it: de-blocking records. With some other languages, a person will sometimes write more than one logical record or associated group of data into what the operating system and Datatrieve consider to be a single record. An example file containing some names, looks like this:

Dump of DB2:[300,3]DEBLOCK.SEQ;1

Record number 01. - Size 512. bytes

```

000 W o l f J F l y
020 w h e e l
040 M a h a t m a K J e e
060 v e s
100 H u g o Z H a c
120 k i n b u s h
140 O t i s C r i
160 b b l e c o b l i s
200 S Q Q u a
220 l e
240 S a m G r u
260 n i o n
300
320
340
360
400
420
440
460
500

```

This data is all one physical record as there is no separation between logical records (each name), but it can be "de-blocked" with the proper record definition.

```
DEFINE RECORD DEBLOCK-REC
01 DEBLOCK-REC.
 03 FIELDS OCCURS 16 TIMES.
 06 FIRST PIC X(12).
 06 MI PIC X.
 06 LAST PIC X(14).
 06 FILLER PIC X(5).
```

;

Note that there is no "fixed" portion to this definition: everything is in the "variable" portion (inner list) of the record definition. When a domain with this record definition is readied and printed, it looks like this:

| FIRST   | MI | LAST          |
|---------|----|---------------|
| Wolf    | J  | Flywheel      |
| Mahatma | K  | Jeeves        |
| Hugo    | Z  | Hackinbush    |
| Otis    |    | Cribblecoblis |
| S       | Q  | Quale         |
| Sam     |    | Grunion       |

The blank lines at the bottom are due to the fact that the entire inner list is printed by default, and the unused entries are filled with blanks. This record definition can be made to print better looking data with the redefines and computed by expressions.

```
DEFINE RECORD DEBLOCK-TWO
01 DEBLOCK-TWO.
 08 FIELDS OCCURS 16 TIMES.
 16 DUMMY.
 24 FILLER PIC X(32).
 16 N REDEFINES DUMMY.
 24 FIRST PIC X(12).
 24 MI PIC X.
 24 LAST PIC X(14).
 16 NAME PIC X(29)
 COMPUTED BY FIRST||" "||MI||" "||LAST.
```

;

Once again, space is allocated with filler, then redefined with the real fields to "hide" them. The COMPUTED BY may surprise persons who expect that only math functions can be used with this clause, but it works just as well with character strings. The effect of this is to squeeze down the three fields to a single field with blank spaces removed.

## NAME

```
Wolf J Flywheel
Mahatma K Jeeves
Hugo Z Hackinbush
Otis Cribblecoblis
S Q Quale
Sam Grunion
```

Processing the name in this way removes one of the objections many people have to computer output; that it looks too "regimented" or too ridgedly organized. Removing the blanks makes the names look more as they would when written or typed by a person: simple little things like this can significantly improve the appearance of a report. The use of the COMPUTED BY in the record definition allows us still to enter the first and last names separately so we can, for example, sort the data by last name or otherwise access the individual fields, and then use the concatenated name where desired.

It was stated that if a record definition is found not to meet the requirements of an application, it can be changed. The rules for changing an existing record definition (without having to change the domain or file used by the domain) reduce to a simple requirement: the length of the record cannot change. This condition results in the following rules:

Field names, group names, query headers, query names, and edit strings may always be changed, as they do not affect the length of the record, but watch out for any views which access that record definition: if you change a field or group name, any views which use that field must be changed so that their corresponding group and field names match.

Query headers, query names, and edit strings may be added or deleted.

Group names may be added or deleted, provided they do not cause REDEFINES or OCCURS boundaries to be crossed.

REDEFINES fields or groups may be added or deleted.

As a general rule, PICTURE cannot be changed, and elementary fields cannot be added or deleted. There are a few cases where a change can be made, very, VERY, VERY carefully, when the length of a field will not change. For example:

```
03 A PIC 999. can be changed to
03 A PIC XXX.
```

but the data can no longer be used as a number, nor can leading zeroes be suppressed. Going the other way, from PIC XXX to PIC 999 may cause very strange numbers to appear.

03 A PIC 99 USAGE IS INTEGER.

can change to

03 A PIC 999 USAGE IS INTEGER.

but not to

03 A PIC 9999 USAGE IS INTEGER.

because the first two are 2 bytes in length and the last is 4 bytes in length.

Fields can be combined if the total length is the same:

03 CITY PIC X(10).

03 STATE PIC X(2).

can both be replaced with

03 CITY-STATE PIC X(12).

There is one change which is always allowed, and that is to replace any field or combination of adjacent fields with FILLER of the same length.

A VALID IF statement can also be added, deleted, or modified. It should be noted that VALID IF applies only when storing or modifying data with Datatrieve: it does not check data which is already in the domain. Thus it is entirely possible to store data in a field, then add a VALID IF clause which makes that data invalid: no more data of that kind may be added, but the existing data will be unchanged. This is similar to the condition stated above where PIC XXX could be changed to PIC 999: if the data in the field happens to all be numeric digits, the data will be correct, but if there is anything else (including leading blanks), Datatrieve will assume they are supposed to be numeric digits, and this will result in some strange numbers being printed out. Datatrieve will not check the existing data, but will not allow you to modify or store a new field with non-numeric data.

If all of this scares you, it should be noted that the worst that can happen is that if you do change the length of the record definition, when you READY the domain you will get an error message telling you that the record lengths don't match, or you will read the data and get strange results. As long as you extract a copy of your definition before you start, and ready the domain for read only the first time after you change the definition and look at the data, it is very unlikely that you can damage your data, and the worst that can happen is you will have to make additional corrections to your record definition, or go back to the old one.

When changes are required which will not fit the above rules, such as adding an additional elementary field, then a new file

will be needed to hold the new length record. One way to transfer information is shown under "Creating New Domains from Old" in the Datatrieve manual. This basically depends on the FOR statement, which reads the old domain one record at a time. Suppose I defined an address file like this:

```
DEFINE RECORD OLD-ADDR-REC
01 OLD-ADDR-REC.
 03 NAME PIC X(20).
 03 STREET PIC X(20).
 03 CITY PIC X(10).
 03 STATE PIC X(2).
;
```

and I have stored some data,

| NAME               | STREET           | CITY     | STATE |
|--------------------|------------------|----------|-------|
| B. Z. Lederman     | 2572 E. 22nd St. | Brooklyn | NY    |
| Hugo Z. Hackinbush | 11 Julius Ct.    | Hialeah  | FL    |

and then I realize I forgot the Zip code. I can define a new record and corresponding domain:

```
DEFINE RECORD NEW-ADDR-REC
01 NEW-ADDR-REC.
 03 NAME PIC X(20).
 03 STREET PIC X(20).
 03 CITY PIC X(12).
 03 STATE PIC X(2).
 03 ZIP PIC 99999.
;
```

with an additional field and an increased field size for CITY, and move the data.

```
READY OLD-ADDR
READY NEW-ADDR WRITE
FOR OLD-ADDR STORE NEW-ADDR USING
 NEW-ADDR-REC=OLD-ADDR-REC
```

and the data will now be

| NAME               | STREET           | CITY     | STATE | ZIP   |
|--------------------|------------------|----------|-------|-------|
| B. Z. Lederman     | 2572 E. 22nd St. | Brooklyn | NY    | 00000 |
| Hugo Z. Hackinbush | 11 Julius Ct.    | Hialeah  | FL    | 00000 |

Note that ZIP was filled with default characters, which in the case of numeric fields is zero, and that the city field has been moved. When one says USING new=old, Datatrieve will match up fields with the same name in moving data: any new fields get zeroes or blanks. If the new fields are at the end of the record, it would be faster to move the data outside of Datatrieve using one of the RMS utilities (CNV or IFL on the 11, CONVERT on the VAX), or SORT, any of which can pad the new records and will transfer data between files faster than Datatrieve. If you are removing a field from the end of the definition, the same rule applies, as the utilities will truncate long records. If you are changing fields in the middle as was done here when the size of CITY was increased, then the SORT utility can be used, but if it is a one time only

change, it will be easier (if slower) to use Datatrieve than to set up the sort commands. For example, suppose I decided I needed a second address line.

```

DEFINE RECORD NEW-ADDR-REC
01 NEW-ADDR-REC.
 03 NAME PIC X(20).
 03 STREET PIC X(20).
 03 SECOND-LINE PIC X(20).
 03 CITY PIC X(12).
 03 STATE PIC X(2).
 03 ZIP PIC 99999.
;

```

Using the same statement as before for conversion yields:

| NAME               | STREET           | LINE | CITY     | STATE | ZIP   |
|--------------------|------------------|------|----------|-------|-------|
| B. Z. Lederman     | 2572 E. 22nd St. |      | Brooklyn | NY    | 00000 |
| Hugo Z. Hackinbush | 11 Julius Ct.    |      | Hialeah  | FL    | 00000 |

with the new line filled with blanks. One can do more than take the defaults, however

```

DEFINE RECORD NEW-ADDR-REC
01 NEW-ADDR-REC.
 03 ENTRY PIC 99.
 03 NAME PIC X(20).
 03 STREET PIC X(20).
 03 CITY PIC X(12).
 03 STATE PIC X(2).
 03 ZIP PIC 99999.
 03 ENTRY-DATE USAGE IS DATE.
;

```

This time I want to add an entry number, and a date, so I have to give Datatrieve a few more commands.

```

DEFINE PROCEDURE CONVERT-ADDR
READY OLD-ADDR
READY NEW-ADDR WRITE
DECLARE COUNTER PIC 99.
COUNTER=0
FOR OLD-ADDR BEGIN
 COUNTER=COUNTER + 1
 STORE NEW-ADDR USING BEGIN
 ENTRY=COUNTER
 NAME=NAME
 STREET=STREET
 CITY=CITY
 STATE=STATE
 ENTRY-DATE="TODAY"
 END
END
RELEASE COUNTER
FINISH
END-PROCEDURE

```

Here I have a temporary variable which will automatically count up the number of entries and store it in the new domain: also, the ENTRY-DATE will automatically be time stamped (on the PDP-11, it will have only the date, not the time) as entered. Note there is still no ZIP=... command: as I have no ZIP data in the old domain, I will let the new domain be filled with the



default value of zero. I could also put in a prompt here and have someone enter the zip code during conversion, but on a long domain this would be very tedious, so it is better to convert automatically and modify the individual zip codes later. The new data is

| ENTRY | NAME               | STREET           | CITY     | STATE | ZIP   | ENTRY DATE |
|-------|--------------------|------------------|----------|-------|-------|------------|
| 01    | B. Z. Lederman     | 2572 E. 22nd St. | Brooklyn | NY    | 00000 | 11-Mar-83  |
| 02    | Hugo Z. Hackinbush | 11 Julius Ct.    | Hialeah  | FL    | 00000 | 11-Mar-83  |

One can go on from here to make more elaborate changes if desired: the basic idea is that Datatrieve can be made to convert data from one domain to another if it should happen that a record definition needs to be changed, and this can be simplified though a careful choice of field names. As another example, suppose I had to change 5 digit zip codes to 9 digit.

```
DEFINE RECORD OLD-ADDR-REC
```

```
01 OLD-ADDR-REC.
 03 NAME PIC X(20).
 03 ZIP PIC 9(5).
```

```
;
```

```
DEFINE RECORD NEW-ADDR-REC
```

```
01 NEW-ADDR-REC.
 03 NAME PIC X(20).
 03 ZIP-CODE.
 06 ZIP PIC 9(5).
 06 PLUS4 PIC 9(4).
```

```
;
```

Because I have defined the field ZIP in both domains, the data can be directly transferred.

```
FOR OLD-ADDR STORE NEW-ADDR USING
 NEW-ADDR-REC=OLD-ADDR-REC
```

| NAME               | ZIP   | PLUS4 |
|--------------------|-------|-------|
| B. Z. Lederman     | 11235 | 0000  |
| Hugo Z. Hackinbush | 33999 | 0000  |

It would probably be better if PLUS4 had an edit string to suppress zeroes, and there was a way to print out the dash that joins the two parts of the ZIP code.

```
DEFINE RECORD NEW-ADDR-REC
```

```
01 NEW-ADDR-REC.
 03 NAME PIC X(20).
 03 ZIP PIC 9(5).
 03 DASH PIC XXX COMPUTED BY " - "
 QUERY-HEADER "".
 03 PLUS4 PIC 9(4) EDIT-STRING Z(4).
```

```
;
```

| NAME               | ZIP   | PLUS4 |
|--------------------|-------|-------|
| B. Z. Lederman     | 11235 | -     |
| Hugo Z. Hackinbush | 33999 | -     |

The "DASH" field was needed because ZIP and PLUS4 are numeric, and the hyphen would be treated as a minus sign: since ZIP codes are usually positive numbers, the hyphen would not print out. (I haven't tried making the PLUS4 field a negative number: if you're curious, you might try it.) The zip suffix doesn't print because of the zero suppression, but the data in the domain looks like this:

```
B. Z. Lederman 112350000
Hugo Z. Hackinbush 339990000
```

Just to give one last example, I will show how to get the data from FAMILIES to the two domains PARENT and OFFSPRING shown earlier.

```
DEFINE PROCEDURE CHANGE-FAMILY
DECLARE COUNTER PIC 999.
COUNTER=0
READY FAMILY
READY PARENT WRITE
READY OFFSPRING WRITE
FOR FAMILY BEGIN
 COUNTER = COUNTER + 1
 STORE PARENT USING BEGIN
 KEY = COUNTER
 FATHER = FATHER
 MOTHER = MOTHER
 END
 FOR KIDS STORE OFFSPRING USING BEGIN
 KEY = COUNTER
 KID = KID
 AGE = AGE
 END
END
FINISH
RELEASE COUNTER
END-PROCEDURE
```

As may be seen, there is a "loop within a loop". The FOR FAMILY moves through the domain and picks up each set of parents, and within this the FOR KIDS moves through the inner list to pick up each kid within a family. The use of the declared variable COUNTER insures that each offspring has the same key as the corresponding parent.

A final note: the length of each field is important when using the REDEFINES clause, when the data must be read by other programs, or when data is to be transferred between different systems (for example, from a VAX to a PDP-11 or PRO). There is a hidden "gotcha" that should be kept in mind:

```
DEFINE RECORD GOTCHA
01 GOTCHA.
 03 A PIC X.
 03 B PIC 99 USAGE IS INTEGER.
;
```

It might be thought that this record is 3 bytes long, and on a VAX it is, but on a PDP-11 it isn't: it is 4 bytes long,

because of something called word alignment. INTEGER, REAL, DOUBLE and DATE must start on a word boundary, which is an even number of bytes: if they don't, Datatrieve inserts a hidden byte to align the data; in this case, between fields A and B. This byte takes up space in the record but can never be accessed, unless you change the definition. As a general principle, space should not be wasted, and if the application must be transported between a PDP-11 and a VAX the records must match, so in this instance, it would be better to reverse the order of the fields, so the INTEGER WOULD BE ON AN EVEN BOUNDARY, AND THE RECORD WOULD THEN BE 3 BYTES LONG; OR PUT AN EXTRA ONE BYTE FIELD BETWEEN A and B and use the space to store some other data field; or use the ALIGNMENT clause, which will make the VAX force word alignments in the same manner as the PDP-11, which will insure compatibility.







# DEVELOPING AN APPLICATIONS LIBRARY ON THE VAX -- SOME OBSERVATIONS

John M. Anderson  
University of North Carolina at Wilmington  
Wilmington, North Carolina

## ABSTRACT

Moving from a remote-batch processing computing environment to an on-campus interactive computing environment at a rapidly growing state university has its problems. From an instructional point of view, one of the most significant problems is providing easy access to a library of useful and reliable programs. With the arrival of a new VAX came the problems associated with developing such a library.

## THE PROBLEM

It took five years for the University of North Carolina at Wilmington to complete the first phase of its process to upgrade computing on its campus. The process, begun in 1980, led to the acquisition of two VAX 11/780's to replace the remote batch connection which had served as the primary source of computer support since 1969.

In June 1984, along with the new VAX hardware came the demand for software to support instruction and research activities. During the painfully long process to acquire the VAX, academic departments had turned to microcomputers for help with immediate computing needs. By the time the VAX arrived, the microcomputer had gained an enthusiastic following. Easy-to-use software and relatively low cost were strong selling points and had convinced many new users that microcomputers were the best alternative. Where once there had been agreement on the direction for improving computing on the campus, there is now uncertainty and a new group of users competing for scarce computer funds.

From my point of view as an instructor, the problem was one of choosing a direction that would meet the computing needs of my students. Given the limited funds for computer equipment in the School of Business Administration, the choice was simple: since there wouldn't be a significant number of microcomputers available for student use any time soon, the VAX would have to do.

## THE VAX CONFIGURATION

The academic VAX has six megabytes of main memory, two RA81 disk drives, an RA60 disk drive, and a TU80 tape drive. Forty-eight DZ11 and DMF32 ports are available for terminal and printer hookups. Each building on the campus has at least one multiplexer which is connected by telephone cable to switching multiplexers located in the computer center.

Three public terminal rooms are staffed and maintained by the campus computer center and provide access to the system sixteen hours per day, seven days a week. In addition, several departments have terminals and microcomputers in laboratories which are tied into the campus network through the multiplexers in their buildings.

A number of dial-in lines are supported for off-campus use of the VAX. The old remote-batch-IBM link can be accessed through HASP+ on the VAX.

Running under VMS, the system supports FORTRAN, BASIC, PASCAL, COBOL, and PL/1 compilers in addition to DATATRIEVE, CAS, SAS, SPSS, and SIMSCRIPT. Access Technology's 20/20 spreadsheet is under evaluation and will probably be available on the system by July.

## THE NEED FOR APPLICATIONS SOFTWARE

In spite of the array of general-purpose software purchased with the VAX, there was a need for more specific applications software to support classroom instruction in the School of Business Administration. Four categories of need were identified: (1) computer-aided instructional packages; (2) drill packages; (3) problem-solving packages; and (4) classroom demonstration packages.

Prior to the VAX, categories (1), (2), and (4) were not adequately addressed because of lack of hardware. A few terminals tied to a remote HP2000 allowed some problem solving along with some BASIC programming instruction. But, most of the University's computer funds were devoted to maintaining a batch system with all of its paraphernalia. With the VAX, however, came the potential for providing ample support in many more areas. All that was needed was the software.

## THE BUSINESS LIBRARY CONCEPT

Since all of our computing had been done on someone else's computer until the VAX arrived, we had the advantage in not having a commitment to a base of applications software. Energy that would otherwise have been devoted to converting old software could be devoted to the selection and development of new software.

Teaching quantitative methods and computer applications without adequate computer support for more than a decade gave me plenty of time to dream about my "ideal" program library for instructional support. Some of the important characteristics of such a library were:

1. It must be easy to use.

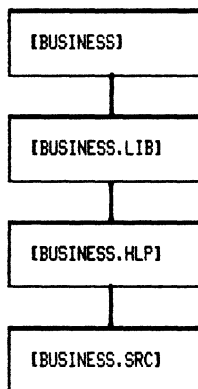
2. It must be self-documented.
3. It must support both keyboard and file input.
4. It must allow hardcopy output.
5. The programs must be tested and reliable.
6. The program/user interface must be consistent.
7. There must be a user-feedback mechanism to allow correction of problems with the system.

Two alternatives were considered for developing a library with those characteristics. The first was to request that the campus academic computer center staff do the work. The second alternative was to do-it-yourself. Because of time requirements and staff limitations, the second alternative was chosen.

### ORGANIZING THE LIBRARY

A main directory called [BUSINESS] was set up for my use by the system manager. Three sub-directories were then created: one called [BUSINESS.LIB]; a second called [BUSINESS.HLP]; and a third called [BUSINESS.SRC]. (See Figure 1.)

Figure 1



The main directory is used for development. Prospective programs are stored there until they can be tested and enhanced for addition to the library. When a program is completed, it is compiled and linked from the main directory. The image is then copied to the .LIB directory and the source copied to the .BAS directory. The .DEV file is deleted as are the .EXE, .LIS, and .OBJ files. A symbol is defined in my LOGIN.COM for a cleanup command file which can be invoked in each directory by simply typing CLEANUP.

The [BUSINESS.LIB] directory contains the LIBRUN command procedure and the executable images in the library. A separate directory is used to keep the "finished products" separate from the "raw materials".

Finally, the [BUSINESS.HLP] directory contains all files related to the online help system. Each program in the image directory has a corresponding help file in the .HLP directory.

### SUMMARY OF TASKS REQUIRED

Several tasks were accomplished after the characteristics and organization of the library were defined:

#### Type BL for Easy Access

A system-wide symbol was defined to allow novice VAX users to get to the library easily. The symbol "BL" was defined in the system manager's login command file, thereby enabling the student to type BL to gain access to the library main menu. (See Figure 2.) Specifically, the symbol was defined as "@[BUSINESS.LIB]LIBRUN". The LIBRUN command file does some accounting, sets the default directory, and displays the library system's main menu. The command file is in the library and can be changed at any time by the library administrator.

Figure 2

## BUSINESS LIBRARY MAIN MENU

#### OPTIONS:

- <1> HELP
- <2> SEE THE DIRECTORY OF PROGRAMS
- <3> EXIT / RETURN TO HOME DIRECTORY
- <4> RUN A PROGRAM
- <5> COMMENT ON LIBRARY
- <6> RUN SPREADSHEET TEST PROGRAMS

Enter choice number and press <RETURN>:

#### Online Help

An online help system was developed which enables the student to select a HELP option from the main menu. A HELP menu is then displayed which provides information on how to run a program, how to return to the home directory, how to set up a data file, how to use the COMMENT utility, and how to get the documentation on the library programs. (See Figure 3.)

Figure 3

#### LIBRARY HELP SYSTEM

#### - M A I N M E N U -

Type the number of your choice and press <RETURN>:

- <1> How to RUN a library program.
- <2> How to return to your own directory.
- <3> Information on the library programs.
- <4> Setting up a data file.
- <5> Using the COMMENT utility.
- <RETURN> Exit the HELP SYSTEM.

Choice?

If the documentation option is selected, a documentation menu is displayed and the user is given the option of having the documentation file dumped to his/her home directory for printing.

Figure 4

## DOCUMENTATION MENU

Type the number of your choice and press <RETURN>.

```
<1> ASSIGN -- The assignment algorithm
<2> CENLIM -- Sampling distribution generator
<3> MEANS -- Descriptive statistics
<4> MULREG -- Multiple linear regression
<5> SIMPLEX -- Simplex algorithm
<6> TRANSP -- Transportation algorithm
<7> TRANDRILL -- Practice transportation problems
<8> APDRILL -- Practice assignment problems
<9> PERT -- PERT/CPM algorithm
<10> FORECAST -- Forecasting module
<11> SIMDRILL -- Practice linear programming problems
<12> EQO -- Economic order quantity analysis
<13> BINOM -- Binomial distribution generator
```

<RETURN> to get back to the HELP SYSTEM MAIN MENU.

What is your choice?

### Online User Feedback

A COMMENT utility was developed which permits the user to send brief comments to the library administrator. This mini-mail utility has an advantage over the MAIL system, namely, the messages are restricted to three lines. The utility of this utility is uncertain at the moment.

### Consistency Among Programs

A series of "standard" modules in BASIC were developed to ensure some measure of consistency among the library programs. For example, every program is introduced the same way when invoked -- i.e., a credit banner is displayed. Every program will prompt the same way for instructions on input/output options. A standard edit routine for keyboard input is added to every program. A standard closing is tacked onto the end of each program. And, most significantly, a standard set of function definitions is tacked onto the beginning of each program to allow cursor control and graphics.

### BASIC Source Language

BASIC was chosen as the source language for the programs in the library because it is widely known and because a large base of BASIC interactive educational software exists for use with microcomputers. A set of FORTRAN-to-BASIC translation modules was developed to convert suitable public domain software.

### 10,000 Lines of Code and Growing

Finally, a list of programs needed for the library was developed. Work on the library began in October and is ongoing. An estimated 10,000 lines of finished and documented code have been entered into the library to date. At least that many lines remain in the queue for testing and enhancement. The finished programs include several in each of the four categories listed earlier. All of them support instruction in management science and statistics.

Many of the programs were first developed for use on microcomputers, but the VAX has proven to be more effective than microcomputers to support my classes. All hardware problems are handled by someone else, and there is no diskette preparation and distribution overhead.

## NEED FOR SYSTEM MANAGER'S HELP

There was a problem at first in establishing the library as a legitimate entity. There might be some resistance to corrupting the system manager's login command file with user-suggested symbol definitions, but the ease-of-use criterion makes it worth fighting for both a simple library command and a clear directory name. Overworked system managers can easily get caught up in optimizing their own workload at the expense of the user. It took some talking, but the idea of system-wide access to a user-developed applications library was finally reluctantly accepted.

## BE PREPARED TO SUPPORT USERS

In proposing the idea of a user library with system-wide access, there is an implied acceptance of responsibility for user support at some level. Several problems have come up concerning this issue. When the library was first set up, the idea was to keep the users in the image directory until they finished running their programs. At that time, they were given instructions on how to return to their home directory. Many students found it easy to get into the library, but forgot how to get out. Their solution was to logoff and complain the next day that they got trapped in the library. Using the SYS\$LOGIN logical name, the library main menu command procedure was improved to include an option to exit the library and return to the home directory. The standard rule has become, "When you don't know what else to do, type BL to get back to the main menu."

## BROWSING WILL TAKE PLACE

Just after the library was released for general access, an idle computer science student had been browsing and TYPEing files. He had managed to work his way into one of the library subdirectories and had TYPED a file which contained a menu screen. The reverse video was turned on and a large flashing warning message was displayed. Not knowing how to turn off the reverse video and the blinking, he sent a message to me through the library's comment utility admonishing me to turn off the reverse video in such files.

I suppose Murphy has already said something like it, but if it's possible to use a system for other than its intended use, someone will do it. It may not be possible to anticipate all problems, but it helps to be aware that there are users who like to explore. And sometimes that exploring might be helpful. So far the exploring has been harmless curiosity, I'm pleased to report.

## PROGRAM TESTING

It's common sense to thoroughly test programs before putting them in the library, but how do you know when they are thoroughly tested? This is an area with some potential for real problems. If one program is put into the library with serious flaws, that's the one everyone will try and its flaws will contaminate the perceived reliability of the entire library. So it is important that every program be subjected to some minimal level of testing. That might slow down the development process, but it will save a lot of complaints and maintenance in the long run.



Not too long ago, a request was made by a faculty member to enter a contributed program into the library. After a few runs of the program, it was clear that it would not catch obvious input errors. The program was set aside until time can be found to debug it. The issue of who's responsible for maintaining contributed programs remains unresolved. Since the work on the library is entirely volunteer, the only way to control program quality is by controlling which programs enter the library.

Another program was "thoroughly" tested and had run successfully for several years on a microcomputer. After converting it to the VAX and testing it again, it was released to the library. Needless to say, one of the first students using the program had problems. The problem was quickly solved, but Orkin's law still holds: no program is ever bug-free! The question of how much testing is enough is still unanswered.

### LIBRARY USAGE

To my surprise, the library concept received only a lukewarm initial response from the faculty in the School of Business. There are many factors which could account for that, not the least of which is their confidence in the person developing the library. But, I think that response was related primarily to a reluctance to use the new computer system. As mentioned above, the University was IBM-batch-oriented until ten months ago, and less than one-fourth of the faculty used the computer for either teaching or research. So, in my view, the problem is first to get them on the computer, and then to work on library acceptance.

Meanwhile, the library is being used by students in my courses and other quantitative courses in the School of Business. The number of library accesses by students and faculty in other departments is increasing. Students in the terminal rooms see business students running the programs and try them out for themselves.

### STUDENT EVALUATION

My students were asked to evaluate their experience with several of the library programs used in support of topics in management science. As you might expect, the responses ranged from "ho hum" to enthusiastic. Although the responses could hardly be labeled unbiased, on the whole they were quite favorable and a number of valuable suggestions were made for improvements.

Upon closer examination, it was found that many of the neutral responses were from students who had been pressed for time and viewed the computer exercises as "extra" work. The more enthusiastic responses were from students who had spent time with the programs and felt that their understanding of the topics had been enhanced as a result. Indeed, the results on a quiz showed a clear correlation between using the drill programs to study the assignment and transportation problems, and the students' grades. Again, it is recognized that there are problems with this type of interpretation, but I am encouraged enough to continue with the development efforts.

### FUTURE DEVELOPMENT

The DAL (Digital Authoring Language) compiler and the REGIS graphics editor offer real promise for developing some exciting courseware. All of the software in the library to date is designed for use on DEC VT100 and VT220 terminals and takes advantage of the special graphics characters, bold, underline, reverse video, blink, cursor posi-

tioning, and other capabilities which can make the programs interesting to the user and more understandable. Full-color, graphics-enhanced versions of some of the programs will be developed in DAL for use with VT240/VT241 terminals which are scheduled to be added to the terminal rooms in the near future.

Recently, a new use for the library emerged as a result of an aborted software demonstration. To make the demo version of the software available for faculty to explore on their own, it was added to the main library menu. Faculty were able to type BL to get to the library main menu and select option 6 to get to the software. All of the setup required to use the software was handled by the main menu command procedure and was transparent to the user. This role as the friendly frontend for users to access programs not in the library might become significant as other departments add useful software to the system.

One of the major goals for the library is to support instruction with a set of flexible classroom demonstration modules. Several distribution generators are in the library now -- sampling distribution generator, binomial distribution generator, and Poisson distribution generator. An inventory simulator and queueing simulator are under development. The list of such modules seems endless in the quantitative area and will ultimately include modules in the non-quantitative areas as well.

### A COMMENT ON DIRECTION

Several of my colleagues have suggested that the VAX is not the way to go -- microcomputers are "where it's at!" I agree with the part about microcomputers and that's precisely why my efforts are aimed at the development of a VAX-based library. I'm betting that the trend toward multiuser microcomputer systems will continue and that Digital's systems will be among the best. With any luck, our School of Business Administration will install its own multiuser micro-VAX system. By that time, the library will be a proven and accepted software system, waiting to be moved onto the micro-VAX. Only time will tell.

## INTRODUCTION TO MICROCOMPUTERS FOR ADULTS

Dr. Richard L. Kopec  
Incarnate Word College  
San Antonio, Texas 78209

### ABSTRACT

A continuing education course designed for adults is described in this article. Organization, publicity, fees, content, texts, teaching strategy, scheduling, and evaluation are discussed. Personal observations by the instructor about the success of different teaching formats are included.

This paper will discuss the organization and presentation of a continuing education course covering microcomputers that has been taught by the author for the past two years at Incarnate Word College in San Antonio, Texas. Offered through the Continuing Education Department, this course serves to introduce adults to the operation and application of microcomputers both at home and in the workplace. By presenting a thorough but elementary explanation of machine architecture and operation, memory organization and utilization, and by editing and executing simple BASIC programs, the primary objective of the class is to dispel the myths and fears that frequently haunt many of the adults taking the course.

Offered about three times a year, the course is targeted toward educating both working and retired adults from widely divergent backgrounds in the greater San Antonio area. Many of these people are retired military personnel or wives of retired military officers and most have college degrees in their own right. Parents with children already using a microcomputer at home have expressed a desire "to keep up with" the younger generation. There have also been a few teenagers in course and some employees from local business firms. The latter are often enrolled in the course by the office manager for purpose of acquainting them with computer basics in anticipation of the purchase of a business computer that these same employees will be expected to operate. All have an intense desire to learn as much as possible about the operation of a microcomputer since many own or are planning to buy a microcomputer in the near future, either for themselves or for relatives. This course also serves as invaluable preparation for those planning to take more advanced coursework in the computer area.

Advertising is handled jointly by the CE department and the public relations office. About three times a year (January, May and September), the CE department mails a detailed brochure to about 6000 people. As names are added to the list, a weekly mailing continues through the year as well. This brochure contains course information for all courses offered by the CE Department including course fees, meeting times, instructors, prerequisites (if any) and a registration form to be filled out and mailed back. The PR people assist by placing advertisements in several of the local newspapers and on local radio and television stations. Many of the enrollees have taken the course on recommendation of previous attendees, so word of mouth is also an important source of advertising.

### **FLEXIBLE SCHEDULING**

The cost of the course is based on the total number of hours the class meets, which, in turn, is determined by the availability of the instructor. Typically the course will meet for a total of 12 to 15 hours at a cost of about \$8 per hour. Although some prospective students have expressed some dismay over the tuition cost, the class has rarely been underenrolled and frequently has a waiting list.

The meeting times and days of the week vary, again as required by the instructor and desired by the continuing education program director. The course has been taught using a variety of different formats: two successive Saturdays meeting a total of seven hours each day including a one hour lunch break, a combination of four 3 hour sessions consisting of two weekday evenings (usually Tuesdays) coupled with Saturday mornings over a two week period, three hour classes twice a week (on weekday evenings) for two weeks, and three hour periods once a

week (also on weekdays) for four to five weeks. The latter three formats appear to be the most desirable for both the instructor and the students, since the seven hour (including lunch) Saturday meeting tends to be too grueling for the older participants (not to mention the instructor). Most of the adults also reach "information overload" shortly after lunch so the instruction time is not well utilized.

#### CE CREDIT AWARDED

All attendees are awarded 0.1 hours of continuing education credit for each hour the course meets. Certificates are mailed to each participant upon completion of the course. Total enrollment is limited to no more than the number of microcomputers available (currently 13). The college has recently purchased 10 additional units so it is expected that the course enrollment will increase accordingly. Doubling up students on the micros is not acceptable to the instructor, because either one of the pair does all the actual programming while the second only watches or there is some friction between the partners over use of the machine!

The course content is not formally specified since the instructor attempts to tailor the material presented to suit the needs and desires of the class. The course itself is logically divided up into 4 or 5 three hour sessions. The first hour and half is devoted to a lecture type presentation, followed by a 15 minute break, and concluded with a lab session lasting about one hour and a half. A typical course outline is given below.

#### COURSE OUTLINE

##### Session 1:

###### Lecture

- a) View "The Computer"
- b) "Buzzwords"
- c) Computer architecture
- d) Basic hardware
- e) ASCII coding

###### Laboratory

- a) Demonstrate use of hardware
- b) Booting a disk
- c) Use available software

##### Session 2:

###### Lecture

- a) Disk operating system
- b) Memory organization and function
- c) Programming in BASIC

###### Laboratory

- a) Copying, editing, and running simple BASIC programs

##### Session 3:

###### Lecture

- a) BASIC programming continued
- b) Other programming languages

###### Laboratory

- a) Copying, editing, and running simple BASIC programs

##### b) Using the printer

#### Session 4:

##### Lecture

- a) Graphics
- b) Review of BASIC programming language
- c) Overview of programming considerations

d) Demonstration of canned software featuring various peripheral devices available

##### Laboratory

- a) Develop simple graphics programs

#### Session 5:

##### Lecture

- a) Applications
- b) Commercially available software
- c) Telecommunication
- d) Future trends
- e) Computer literature
- f) Models available
- g) What to look for when purchasing microcomputers

##### microcomputers

- h) Wrap-up, evaluation

##### Laboratory

- a) Free time

An average class session always begins with an informal lecture on the topics listed above, except on the first day. At the first session, a two part slide presentation with accompanying audio commentary on videocassette is shown first. Part one of the video features some historical background on the development of modern computers and the people who pioneered them. The second part of the videocassette describes how a modern computer works and serves as a lead in to the instructor's lecture. The ideas introduced in the film are developed further and applied to the particular computer system being used in the laboratory (in this case Apple II+'s and Apple IIe's).

The first laboratory session is designed to help the class to overcome their initial fear of the computer and, most importantly, their conviction that they will "press the wrong key" and destroy the computer. During the first lab, the students are introduced to the essential hardware components and how they function. The interior components of the Apple are also discussed including the purpose of the expansion slots and peripheral boards present. They are each given a copy of the system master disk and a tutorial lab manual(2) oriented specifically to the Apple computer, then essentially turned loose. The instructor then visits the students individually during the lab sessions to help solve immediate problems and explain certain aspects of the computer responses encountered.

#### TRIAL & ERROR APPROACH

No specific explanation concerning the behavior of the computer is given in lecture before the first lab session other than the

minimum essential information required to operate the machine. More explicitly, no explanations are given in reference to diagnostic machine responses such as the infamous "SYNTAX ERROR" message that is frequently encountered. All of this explanatory information appears in the tutorial manual so detailed explanation of these phenomena are deferred until the next lecture session.

During the second lecture the instructor begins to discuss the peculiar computer responses observed when the various commands were attempted. Such explanations include, for example, a detailed discussion of the difference between the number zero and the letter "0", the importance of the "RETURN" key, the difference between "HOME" and "NEW", and similar ideas. This approach is sometimes initially very frustrating to the class at first, since they feel totally in the dark about what they are doing and why the system responds as it does. However, subsequent sessions that explain these mysteries are much more effective after the class has already seen the machine's error messages. This brute force approach works well because the students now have some first hand experience to relate to the lecture discussion, even though many of the adults are initially very discouraged.

#### TOPICS STRESSED

Succeeding sessions begin with a quick review of previous material before continuing with new topics. BASIC is the only programming language used in the laboratory although other languages may be discussed in the lectures. As the class progresses, the concepts of an operating system, structured programming, reserved words, variables, memory organization, and role of the user are developed. A thorough explanation of computer "buzzwords" such as RAM, ROM, CPU and similar technical jargon is also covered. Low resolution graphics is usually presented and, if there is sufficient interest, high resolution graphics is also covered.

During the final meeting of each session, the lecture portion of the course will usually last longer than the allotted hour and a quarter period, and it may be given during the last half of the session rather than the the first. This is the most flexible session of all since the lecture material is oriented towards the interests of the class. It will usually cover a brief review of available computer systems and then branch off into such areas as telecommunications or popular software packages. When possible, local vendors are sometimes brought in to demonstrate their latest technology (such as the Apple MacIntosh).

The final lab session is completely unstructured. The class is encouraged to finish up the material from the lab manual or to create their own BASIC programs. On

very rare occasions, someone may even write an assembly language program!

#### NO GRADES ASSIGNED

Attendance is taken, but no homework or exams are given, nor is a grade assigned. The major course goal is merely to get these adults to feel comfortable using the computer and put them into a position where they are more knowledgeable about computers in general. Giving exams and grades would be counterproductive to achieving these ends because many of the adults fear they would not do well academically and would therefore be less likely to enroll in the course.

In addition to the laboratory manual that is loaned to each student, a paperback book that deals with the topic of computer literacy is also given to each student. Although not formally used in the course (no reading assignments are usually given in the book), this book serves as a reference guide for the class and supplement to the lectures. This book will usually contain a very broad and inclusive discussion on microcomputer, programming, peripherals, literature, and reviews on currently available systems. Although the material is presented to some depth, all of these books are written for a computer novice and were selected precisely for this reason.

#### TEXTS UPDATED FREQUENTLY

A number of different books(3) have been used over the past two years for this purpose. Since the personal computer market is so volatile, books that reflect the most recent innovations are constantly being sought to replace whichever one is currently in use. (The next course offered in July, 1985 will use Computer Wimp.) The cost of this book, typically \$10 to \$20 is included in the course fee. The text used in the laboratory, however, is collected during the final class meeting and recycled for the the next group. Since this text is oriented to a specific computer and operating system, its scope is much too limited to be of general interest to a majority of the enrollees. However, a copy of this tutorial manual is made available through the school bookstore for any who desire to purchase a copy.

At the conclusion of the last session, an evaluation of both the instructor and the course is taken. Each student is asked to complete an evaluation form with eight questions relating to course content, instructor effectiveness, and personal satisfaction. The results to date have been extremely positive on all questions asked. A sample of the questionnaire is reproduced below:

#### EVALUATION FORM

##### Rating

- 5 Outstanding - demonstrates superior accomplishment
- 4 Above Average - exceeds normal requirements
- 3 Average - meets normal requirements
- 2 Below Average - improvement needed
- N Insufficient opportunity to observe

#### Questions

- A. Content was as described in brochure
- B. Subject matter adequately covered
- C. Content presented was suitable for my background and experience
- D. Instructor knowledgeable in content area
- E. Program was well paced within allotted time
- F. Participants were encouraged to take an active part
- G. The program met my individual objectives
- H. How would you rate this program in relation to other programs you have attended?

A section for comments is also included. Many of the students who complete the evaluation have requested the addition of an advanced course that complements the introductory course. In response to this demand, The college is currently considering a course designed around a popular integrated software package which contains word processing, spreadsheet, and data base capabilities.

The Continuing Education Department also offers a variety of courses oriented to primary and secondary school children, particularly during the summer months. Further information about these computer courses, which are not taught by the author, can be obtained from the IWC Continuing Education Department.

#### REFERENCES

1. "The Computer", Hawkhill Associates, Inc., Madison, Wisconsin
2. Adults Can Touch Computers (formerly titled Adults Can Touch Too), Shillingburg, Love Publishing Co., 1983.
3. Understanding Computers, Hopper and Mandell, West Publishing, 1984.  
Your First Computer, Zaks, Sybex, 1980.  
The Personal Computer Book, McWilliams, Ballantine Prelude Press, 1982.  
Computer Wimp, Bear, Ten Speed Press, 1983.





We don't sit here and say, "You've gotta program an Assembly because it's the only way it will be fast enough." Ninety percent of the time it will be fast enough, and you can, with minimal effort ( and I'll show you in a few minutes about minimal effort) fix it to run fast enough.

Figure 7 is an example of "C" code written for test in simulation. This is a code fragment of a piece of our current product. Its function is to put a telephone line on hook, and fills the FIFO buffer going out to that telephone line with SILENCE. Is that a hardware oriented enough application? This runs in simulation on the VAX. Lets examine this code and note the use of macros. The coding standard requires that all macros have the first letter capitalized, and defined constants are in all upper case.

Take a look at the lines numbered with (3). Those are macros that actually go out and touch the hardware. For example, `Select_window` macro selects a particular memory bank to be mapped into physical address space. The `Fill` macro on the next line fills that bank of memory with a constant SILENCE. The `Wr_chn_cmd` macro writes a command to that IO channel. On the VAX, these macros turn into the print statements that print their parameters and say, "This is what I just tried to do with the hardware." In some cases more detailed simulation can be used. For instance the `Select_window()` macros could actually use a large array of VAX memory to simulate the bank switching. In the case of the `Write_chn_cmd`, the macro actually sets the status appropriately in a simulated set of registers. Thus a `Read_chn_status` macro can return the appropriate channel status.

Using macros in this fashion provides a number of benefits. Foremost, software development is decoupled from hardware development. Testing can begin well before there is operational hardware. Second, the process of writing such macros increases the understanding of the programmers of how the hardware is supposed to work.

One might ask, "Can I simulate interrupts with this sheme?" The answer is yes. The program that this example code comes from used three interrupts. The way you do it is to write the interrupt handler in "C" and surround it with just enough Assembly to fix the stack, set up for "C" and call the handler. The way you run it in simulation is, to wrap a driver routine around the interrupt handler, instead of the Assembly. The driver sets the simulated hardware registers and calls the interrupt handler. Since the interrupt handle is in "C", it cannot tell how it was called, via assembly routine linked to an interrupt vector, or by a test driver. Since all hardware manipulation is done with macros the interrupt handler cannot tell if it is in simulation or not.

A couple of other things about what we do in the simulated environment. The lines marked with (1) are Enter and Exit macros that are required on all subroutines in the system. My coders get a template to create a module. The Enter and Exit are already there, as are the formatted comment block. They do a global replace on a string "<NAME>" with the module name, and the template is filled in

The lines marked with (2) are intermediate printouts. They implement the intelligent trace by compiling to print statements. For instance, the first macro, `Variable`, translates to a print for the variable `chn_ptr` with a format of four hex digits. The `Comment` macro translates to a print statement using the provided format statement to print the optional variable. This is the form in which all comments in the code are made. This results in a very descriptive program trace. To lower the burden of using this format for comments, we have built the comment macro into our language sensitive editor. I type the first three letters, and the editor inserts a prototype `Comment` macro or `Variable` macro.

You will note that there are only 3 executable lines in the module. The rest of it is macros and, by the way, most of those do not even compile on the target environment.

#### Other Tools

We use a number of tools in conjunction with those detailed above. The following sections discuss these tools.

Structured Analysis And Design - We use structure analysis and design. Unfortunately, I have not seen any automated tools for SA that I think are worth the price. So we do it manually. The end result is that it is not updated after development begins. It is just too costly to do manually.

EMACS Editor - We use EMACS as an editor with some language-sensitive extensions. We have built a package that supports coding in "C". In particular, it corrects most of the misspellings and gives you some of the construct prototypes. We've thought about doing a fullblown language-sensitive editor in EMACS, and it would be possible. My philosophy at this point is, knowing how many MLISP programmers I have available, the \$10,000 to buy it from DEC is the least cost solution.

Forms Library And RUNOFF - The projects have a central forms library. This library holds forms for such items as:

- C Programs
- C Subroutines
- Documents
- Letters
- Envelopes



change at 11 where he replaced three lines of code.

The ACTIVITY HISTORY section provides a list of all actions taken under this PPR. Notice the second to last line, another team member, CHILL, commented on the problem. All other actions were involved in implementing the fix. As you notice, I'm falling down on my job, because I haven't closed it.

The final section is the TEXTUAL COMMENTS section. Each textual comment is time-tagged and the kind of comment is noted. If you'll notice, look at some of the hours on some of the comments, the time of day they were made. These are tame. There are a lot of comment tagged 03's and 04's in front of them.

This problem is somewhat interesting. The problem was found by David Castles. The module was written by Craig Hill. Craig Hill was a little unhappy when somebody reported a problem on his software because he had never had that happen to him before. I had to remind him about egoless programming and point out to him that nobody was saying that he was a poor programmer. We had just found this problem and we needed to get it fixed. By the way, David Castles, who is on a separate contract, went off and fixed it for him. So Craig should be quiet and be thankful. It was Craig's problem, and Castles fixed it for him, free.

### Testing In Simulation

The final major technique that we use is testing in simulation. The following are some of the issues to be considered when building a product for test in simulations:

- Design for Test in Simulation
- High Level Language
- Macros for all External Interfaces
- Portable I/O Package
- Intelligent Trace Via Macros
- Efficiency Thru Measurement

The key point in using simulation for testing is commitment. You have to decide to do it. You have to really say, "I'm going to do that." You can't just say, "Yeah, that's a nice idea and if we can do it when we get to the point of testing, fine." You've got to say up front, "I'm going to design it to be tested and validated in simulation."

If you design for simulation I don't care what it is, it can be made to run in simulation. One of the projects I will describe later is a device driver. It is interrupt driven. It fields three separate interrupts and does all of its processing in response to these interrupts and hardware status. It was tested on the VAX in simulation.

High level language is virtually a requirement for testing in simulation. It should be a requirement regardless of testing methodology, but that's another issue. Assembly doesn't run in simulation very well. It's really hard to run Assembly language code for the 8086 on a VAX. It's much easier to write everything in high-level language. Besides you benefit from much better programmer productivity and fewer errors.

The simulation environment typically changes all of the external interfaces of a product. Therefore, use macros to hide these interfaces. In particular, you don't touch the hardware without a macro. My contractors didn't understand this. They're used to barefooting an IBM PC, and they say, "Gee, you mean I can't just POKE this location and have something happen?" And I say, "Sure, you can. Just put a macro around it." I don't want to see a poke. (Actually, you can't do a poke in "C" but the moral equivalent can be achieved.) Also, don't touch the operating system without a macro. This is true for most operating system services. The Vax has an entirely different set of system services than the PS-DOS does. Use a macro to hide the difference.

We use a portable IO package; in particular, we use the UNIX(tm) V7 package supplied by Whitesmith. VAX11C supports identical I/O routines, and we haven't had any problems compiling and executing identical code under both environments. However, it did require a little bit of work in some of the headers to get that to work.

Another benefit of test and debug in simulation is that we can implement intelligent trace. What I mean by intelligent trace is that we insert a number of macros that print information. Those macros, in the target environment, are defined to be nothing. So they cost nothing. On the VAX they produce an annotated trace of the actual execution, the intermediate results, routine calls, etc. Whatever I'd like to have. I can run that test on VAX in batch. Then look over the batch log at my leisure. This permits me to do a detailed analysis in a non-realtime fashion. Note this also fits nicely into the DEC/TEST MANAGER scheme.

Finally, we do not code for efficiency. Gee, who said that? Did I say that about microprocessors? Everybody knows that you have to constantly worry about efficiency on a micro, don't they. I repeat, we don't code for efficiency. We write code for maintainability. Then when we get it in the environment we figure out if it's fast enough. If it's not fast enough, we measure what's broken and fix it. We don't sit here and say, "Every routine's gotta run blazing fast", because every routine doesn't.

-----  
 | PPR REPORT |  
 -----

project = NITA

|      |                     |            |  |        |
|------|---------------------|------------|--|--------|
| PPR# |                     | TITLE      |  | STATUS |
| 0050 | FIELD MISSPELLED IN | HARDWARE.H |  | OPEN   |

|         |          |                   |     |          |
|---------|----------|-------------------|-----|----------|
| DATE    | AUTHOR   | SHORT TITLE       | PRI | ASSIGNEE |
| 5/16/85 | DCASTLES | BOARD -> REGISTER | 9   | DCASTLES |

|                      |              |
|----------------------|--------------|
| SYSTEM CONFIGURATION | TEST CASE ID |
| Not Applicable       | Not App.     |

----- EFFECTED MODULES -----

| PPR  | MODULE     | GEN | DELTA | STATUS   |
|------|------------|-----|-------|----------|
| 0050 | HARDWARE.H | 010 | 0010  | REPLACED |
| 0050 | HARDWARE.H | 010 | 0000  | CANCELED |
| 0050 | HARDWARE.H | 011 | 0003  | REPLACED |

----- ACTIVITY HISTORY -----

| PPR#  | WHO      | MODULE     | DATE    | ACTION       |
|-------|----------|------------|---------|--------------|
| 00050 | DCASTLES |            | 5/16/85 | CREATED      |
| 00050 | DCASTLES | HARDWARE.H | 5/16/85 | RESERVED MOD |
| 00050 | DCASTLES | HARDWARE.H | 5/16/85 | REPLACED MOD |
| 00050 | DCASTLES | HARDWARE.H | 5/16/85 | RESERVED MOD |
| 00050 | DCASTLES | HARDWARE.H | 5/16/85 | CANCELED MOD |
| 00050 | DCASTLES | HARDWARE.H | 5/16/85 | RESERVED MOD |
| 00050 | DCASTLES | HARDWARE.H | 5/16/85 | REPLACED MOD |
| 00050 | CHILL    | n/a        | 5/16/85 | COMMENTED    |
| 00050 | CHILL    | n/a        | 5/16/85 | COMMENTED    |
| 00050 | DCASTLES | n/a        | 5/16/85 | REQUEST CLS  |

===== TEXTUAL COMMENTS =====

COMMENT made by CHILL on 1985-05-16 23:11:46.50:  
 The statements concerning the near-term volatility of hardware.h ....

COMMENT made by CHILL on 1985-05-16 21:36:41.64:  
 The recent developments concerning the header file hardware.h appear to be causing much grief for all concerned. I agree that we should address the issue in an expedient manner....

CREATE made by DCASTLES on 1985-05-16 09:48:00.85:  
 The field board -> registers was spelled board -> register in the macro Oldval in hardware.h causing myriad compile time errors. I have generated this PPR so that I may go and fix the file....

Figure 5  
 Example of PPR Report

operation where we put contractors on an hourly basis, then assigning them a problem report to solve immediately means an expenditure of money. Somebody should approve that report before they run off and spend money; otherwise, we're giving them a blank check.

Back to the flow, the developer is going to fix the problem. In the process, he may RESERVE, REPLACE and UNRESERVE any number of CMS elements to accomplish the fix. Each of these actions is recorded and assigned to the PPS record for that problem.

Sooner or later, he finishes the problem and forwards it to me for review. This time I review what was fixed and any comments he may have made. I can now detect whether or not he has updated his documentation, because all of the documentation is in the CMS library. There will be a record in the problem report that he checked out that documentation to fix it. If he didn't fix it I can immediately reassign the problem to him and say, "Now, fix your documentation." One of the key problems in traditional development is that the documentation is never updated. It is just forgotten. Finally, when I am convinced that all of the work on the problem has been completed, I assign it to a CLOSED state.

One other function is the COMMENT facility. Any member of the staff can COMMENT on any problem. This provides a formal means to capture all of the discussion that occurs around the solution of a problem.

Figure 4 lists the command options that are available under PPS:

```

CREATE
FETCH ppr_num
FINISH ppr_num
$ PPS COMMENT ppr_num
CLOSE ppr_num
REASSIGN ppr_num user_id
ASSIGN ppr_num user_id
SHOW report_type

```

Figure 4  
PPS Command Options

I've talked about about the CREATE option. The FETCH option retrieves a formatted copy of a PPR from the data base with the associated text comments. Figure 5 is an example of a formatted PPR report. Another option I haven't mentioned is the SHOW option. This command option permits limited queries of the PPR data base for the unwashed user. More complex reports are generated with DTR commands directly. Some of the options under SHOW are:

- Show all OPEN PPRs
- Show all PPRs assigned to a person
- Show all PPRs with a search string in the title

Figure 6 presents the options to CMS trapped by MOCK\_CMS.

```

$ MOCK_CMS | CREATE ELEMENT file-spec
RESERVE element ppr_num
REPLACE element ppr_num
UNRESERVE element ppr_num

```

Figure 6  
MOCK\_CMS Command Options Summary

The CREATE we talked about, RESERVE and REPLACE both require a PPR number. The PPR number can be a list. Reservations will be made under each PPR number given. One of the interesting things about REPLACE is that it checks for corresponding reservations. When you say:

```
$ CMS REPLACE MUMBLE.C "5,6,7"
```

MOCK\_CMS makes sure that you indeed did check MUMBLE.C out under 5, 6 and 7. It also checks to be sure that there are no outstanding reservations other than 5, 6 and 7. If it's checked out under 8, 9 and 10 MOCK\_CMS will go ahead and replace it for you, but MOCK\_CMS will RESERVE MUMBLE.C again under those PPR numbers to permit you to work on MUMBLE.C under the remaining PPR's. So, if you would want to stop in the middle of fixing six problems to checkpoint yourself, you can. The UNRESERVE option is to undo incorrect reservations.

Let's look back at figure 5 and examine a PPR report. This is a real problem that I pulled out of the PPS data base before I left. The report has four sections.

The first section contains the information we collect at creation time and placed in the DTR database. As you can see, it is general information. The problem is currently in the OPEN state. It was created by user named DCASTLES, and it was opened on the 16th of May. Since the actual problem was a coding error, it wasn't discovered during tests or by the review, and there's no test case or configuration involved.

The EFFECTIVE MODULES section is a list of any module that was RESERVED under this problem. In this particular case, it was just a single module, HARDWARE.H, which is a "C" header file. We tracked the generation number and, also, the size of the change. If you notice the change from nine to ten, which is on that first line, changed ten lines of code when it was replaced. Then he made another minor

We also use DTR to format special reports. We can ask for: show me all the open problems, show me all the problems assigned to Joe Blow, etc.

The following is our concept of work flow under this system:

1. Developer produces code and places in CMS library using create command.
2. Test team finds a problem.
3. Test team uses PPS command to create a problem report.
4. PPS administrator reviews the problem and assigns it for work.
5. Developer uses MOCK\_CMS to RESERVE and REPLACE modules to solve the problem.
6. Developer requests closure of a problem with the PPS FINISH command.
7. The PPS administrator reviews the fix and either accepts or rejects the request for closure.

Figure 3 depicts this flow. Let's discuss each of these steps in more detail.

First, the developer places code into the CMS library using CMS CREATE. MOCK\_CMS intercepts the CMS CREATE and makes some group assignments based on the file type. The user is given the option to assign other groups and is helped through the process. This help encourages group assignments to be made. For each linkable

module we have a separate group, and each source file is assigned to at least one these groups. The test teams only take their input from the project libraries. This applies not just to the CMS library, but that EXES has been generated using MMS or a similar process that is directly traceable to the source.

When the test team finds a problem (hopefully, they don't do this very often), they use PPS to create the problem report. They are queried for a title and some other information, and finally they make textual report. That report is mailed off to the entire project staff. It is also sent to a reviewer which, in our shop, is me. It also can go directly into fix. 95% of all problems that we discover during development are bonafide problems, and the guy who needs to fix them knows it's his responsibility to fix them. He doesn't need to be told, so he can just go ahead and fix it. He doesn't have to wait for me to get back from DECUS to start fixing the problem.

The review is really there to cut out bogus problems or ones out of scope of the contractors. For example, it would be nice if we could play PACMAN on the color screen while the data base search is going on. Cute idea, but it is out of the scope of the contract. I would immediately close that one. When we get into a maintenance

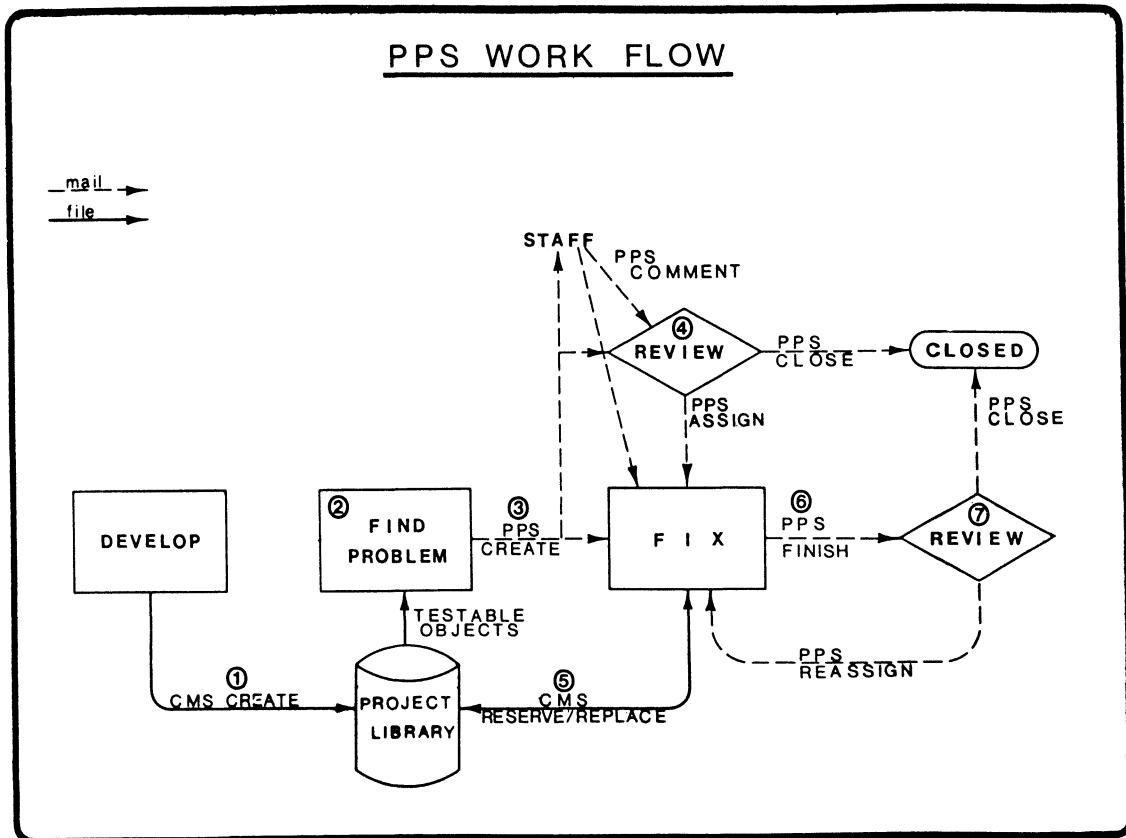


Figure 3  
Work Flow under PPS

has no way to judge the validity of a reason. I do that manually. See the above comment on manual systems.

PPS captures metrics. What do I mean by metric capture? First metrics are some measurable quantities of interest in the development process. We use the PPS system as an edge to trigger the measurement of this information and to store it. Some of the metrics of interest are:

1. How long the developer had a module checked out of the library for a particular problem.
2. How many lines of code changed in a module for a particular problem.
3. How many modules did a particular problem effect.

Those are some that we're collecting right now. We could collect others. We just haven't sat down to think about what we want to collect.

We built PPS on standard utilities that we had -- EMACS, DATATRIEVE, MAIL, and CMS. Figure 2 is an overview of how the project problem system is built. We have two front ends. We place a command procedure, MOCK\_CMS, in front of CMS, and prevent all users from direct access to CMS. This interlocks CMS activity with the PPS sytem. MOCK\_CMS reports to the PPS data base any requests of interest made to the CMS library. Resuests of interest are CMS

RESERVE, REPLACE, and UNRESERVE commands.

The PPS command provides options to CREATE reports, ASSIGN, and CLOSE them. PPS uses mail to send out problem reports to all of the project team. There is a distribution list that controls who gets this mail and the list is project specific. There is also a project specific administrator list based on who is administering the PPS system. Some reports are mailed only to the administrator.

The PPS system uses DATATRIEVE (DTR) to collect the various data. The PPR record itself is a project problem report that contains information such as a title for the problem, a priority, a test environment that exhibits the problem, etc. The RESERVATIONS part of the file contains records for all modules that are currently checked out of the CMS library for every problem report. The HISTORY file tells you action taken on a problem.

EMACS is used as an editor to capture textual comment information. Rather than have you type in at DCL level we throw you into EMACS as an editor (our whole shop runs EMACS), and it creates the textual comments. EMACS also provides multiple windows. For entering textual comments, one window is setup with the formatted report of the entire PPR, including all previous comments, and another window is setup to accept the text of the comment.

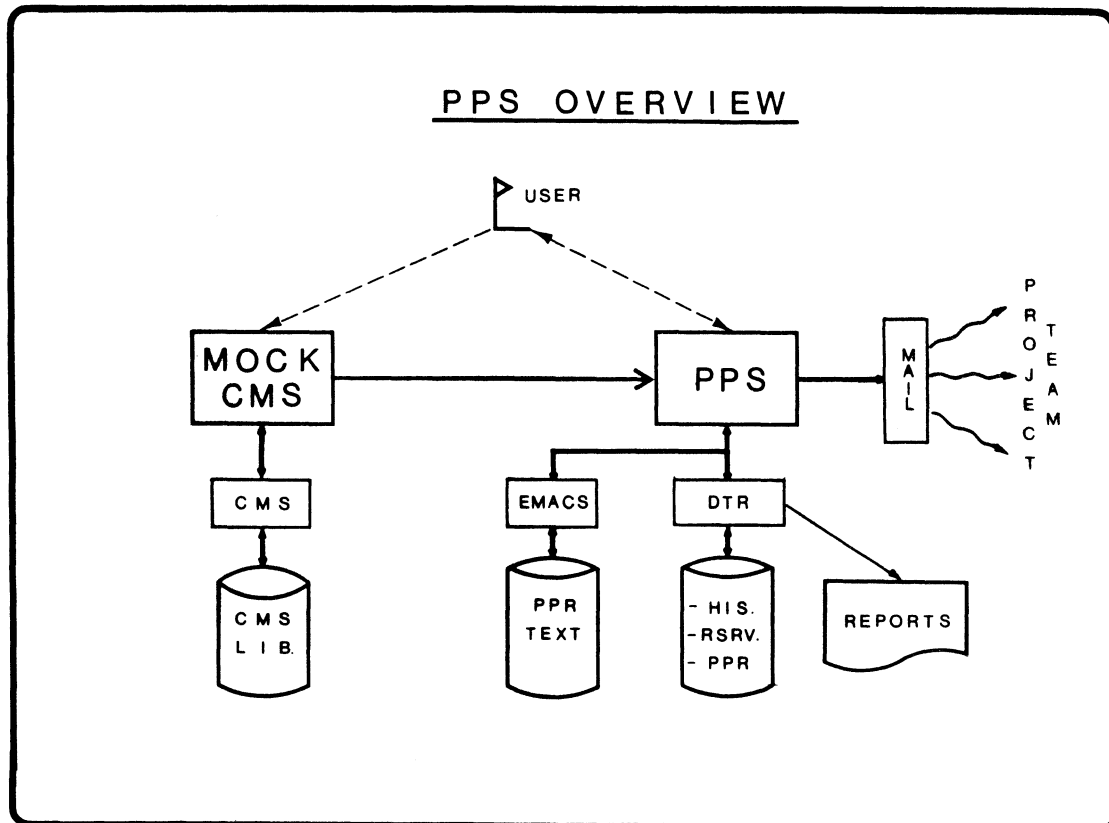


Figure 2  
Overview of PPS System

listings. The source code build procedures are setup to place a new listing in this directory whenever the project object libraries are updated. Thus this set of listings tracks the objects and executables.

The [PROJ\_ID.PPR] contains the project problem reports. In there is two separate directories. [PROJ\_ID.PPR.RECORD] contains the DATATRIEVE (DTR) the files for the problem report data base. The [PROJ\_ID.PPR.TEXT] contains the textual comments.

This structure permits us to write command procedures and software tools to manipulate the components of a project, without making them project specific. This approach is part of my attempts to eliminate one time tools. The biggest problem in reusing tools is that people tend to build to much project specific knowledge into them.

Invocation Mechanism - The project invocation mechanism is the command:

```
$ WORK_ON proj_id
```

For instance, I might enter:

```
$ WORK_ON META
```

The command performs the following functions:

1. WORK\_ON undefines any current project I'm working on. This provides a clean slate between projects.
2. WORK\_ON defines logical names for most of the directories defined in the standard project form.
3. WORK\_ON redefines any VMS commands that might be special for a project. This is accomplished using a project specific command procedure (in [PROJ\_ID.ADMIN]) thus keeping WORK\_ON totally project independent. For example, my current project redefines the CC command from being the standard VAX "C" compiler to a custom command procedure that includes object library update.
4. WORK\_ON defines the generic cross commands. Again this implemented as a project specific command procedure invoked under WORK\_ON. The standard commands we currently use are:
  - XCC - "C" cross compiler
  - XLINK - cross linker
  - XLIB - cross librarian.
  - XASM - cross assembler
  - LINT - "C" syntax checker.
5. WORK\_ON sets the process name. If you do a SHOW USER DCL command, it will show your name, followed by the project that you're working on. If I had

executed the command WORK\_ON META, my process name would be changed to "SCHORNAK\_META". Now everybody on the system knows what I'm working on.

6. WORK\_ON defines the CMS library. This is a standard CMS SET LIBRARY command for [PROJ\_ID.SOURCE].
7. WORK\_ON does a DCL SET DEFAULT to a private-user directory. This directory is the directory dedicated to work by an individual on the project. That way, if you decorate a tree or in any way leave the project, other project personnel can at least find all of your work on a specific project.
8. WORK\_ON defines other project specifics. This is an escape clause to get anything that I have to do that isn't standard. Currently none of our projects use this feature.

### Project Problem System (PPS)

The heart of this environment is the problem tracking system. It tracks all problems. It gives us a automated system for tracking product problems. Normally a developer or tester is working with a product. He sees a problem. As any responsible engineer, he writes the problem down with a description, test setup, possible causes, and recommended solutions. (Sure he does.) But even if he does, where does that peice of paper endup half the time. LOST, that's where. Then that problem shows up about six months later after you have delivered the first one thousand units. Of course management understands none of this and fires you. The objective here is that we don't lose any problems. We may not fix them all, but we at least want to know they exist.

We collect all the dialogue about a problem. In our shop, we don't have a whole lot of meetings in the hall. The ones that we do have are very limited, because not everybody is at the office. The contractors are not on site . The guy who does code reviews is in California. Thus problems are discussed almost exclusively by electronic mail. Now you think, how easy to retain all the discussion, its all entered mail, I just have to file it. WRONG. In previous projects, before PPS was implemented in its current state, we did that. The electronic mail had to be manually collected and retained. Since as is the case with must manual systems, most of the mail was lost. Now, we have a tool to automatically keep them as a permanent part of the project record.

PPS records all of the modifications that anybody makes and why they made them. There's some trust here, because the interlock cannot be automatic. We automatically catch all modifications, but the reason may not match since the machine

service to code review, and expect managers, or other developers to take time from their other activities to do the review. This approach gives the reviewer no incentive to do an adequate job.

Unit and system testing is done in simulation. Remember, we're targeting towards microprocessor operating environments. It's extremely difficult to debug in that environment. It's not so bad to run validation, but it's terrible to debug. There are some things that just can't be tested in simulation. For those, we do the target environment specific tests. Also we revalidate in the target environment. An independent test team performs all of the simulation and validation testing. An aside, the independent test person, resides in California, our shop is in Atlanta. I have met him twice in person, yet he is extremely effective in the testing role.

Now, the best part of all -- deliver, and sell. Finally, we go into maintenance and enhancement, which just repeats 3. through 6. all over again, or, actually, 1. through 6. all over again. The development environment I will describe provides automated support for items 3. through 6. and 8. on the above list.

#### THE DEVELOPMENT ENVIRONMENT

Our development environment is made up of a number of components that we have integrated. The following sections discuss these components.

#### Standard Project Form

The standard project form defines a set of facilities and structures that must be present in every project. This enforced through the use of a single project creation command. This command sets up a project in standard form and creates reasonable defaults for all options. Let us examine these facilities in detail.

Directory Structure - Figure 1 is a summary of the directory structure for the standard form.

The project is a top-level directory. The name of that top-level directory is the project ID. For instance, if I have a project called META, the top-level directory name will be [000000.META]. By the way, META was the project we used to develop the tools that do the project system. Each of the subdirectories has a specific purpose and contains a specific type of file. The subdirectories also give us the control we need to use Access Control Lists correctly.

[PROJ\_ID.ADMIN] contains a number of procedures and the mail files. Source is a straight up CMS library, but it's known to

```
DISK$PROJECTS:[proj_id]
|
|--[ADMIN] = command procs and mail
|
|--[SOURCE] = CMS library
|
|--[PUBLIC] = public versions
|
|--[OBJECT] = object libraries and
 executables
|
|--[HEADER] = "C" include files
|
|--[DOC] = Document reading library
|
|--[LISTING] = Code listing files
|
|--[PPR] = PPR admin files
 |--[RECORD] = DTR data base for PPS
 |
 |--[TEXT] = PPS comment text files
```

Figure 1  
Standard Project Directory Form

be in [PROJ\_ID.SOURCE] and that's computable ahead of time.

The [PROJ\_ID.PUBLIC] contains all of the executable versions. For instance, if you're using this development environment to develop something that runs on the VAX, you might put the image there. (Note that the VAX can be considered as just another target since no target specific information is permitted in the development environment.) You might have a special compiler for this project. You would put the compiler in this directory.

The [PROJ\_ID.OBJECT] contains all object files, including object libraries, EXEs, maps and the like. This directory also contains the VAX version, for execution in simulation, as well as the down-loadable target version.

The [PROJ\_ID.HEADER] contains the "C" "include" files. If we had any Assembly language "include" files they'd be there. As I said before, I try not to have any of those, so at present the only files there are "C" headers. We use the CMS /REFERENCE\_COPY option to keep the headers updated.

The [PROJ\_ID.DOC] directory contains all of the documents of the project. This is the reading library for documents. That includes the design specifications, the functional requirements, operators' manuals, hardware specifications. The DOC directory contains the documents after processing by RUNOFF, so they are formatted. The RUNOFF source to the documents is in the CMS library for the project.

The [PROJ\_ID.LISTING] contains all of the listings. It's a reading file for

product specifications, was to write the specifications for a development environment. The following sections describe the goals setup for this environment.

Bullet Proof Products - Above all else, we have to have bullet-proof products. If the products don't work it doesn't matter how good my development system might be. We'd be out of business by the next DECUS.

Self-Contained Projects - Projects are to be self-contained. We wanted to be able to take one point on the system and say, "Follow this tree and you'll find everything." We didn't want any pieces of magic buried in some command file that SYSLOGIN puts in for you. We didn't want all kinds of pieces of magic that the project really depends on, and that no one is aware of.

Tracability - We want all project related activity to be traceable. We want records of every module change, every technical discussion, and every problem discovered during the entire life cycle of the project. The environment needs to automatically capture this information, since developers and maintainers are not dependable enough to do it manually.

Automated - The environment has to be automated. With six people, it is hard to enforce anything. So we wanted automated checks and records. No human interlocks. I'm here at DECUS. I've got five people in Atlanta, hopefully working their fingers to the nubs. When I get back, I can examine the records, that the environment has captured, to determine what has been accomplished. The records were maintained automatically. It will keep track of every line of code written, and every problem found.

Standard Form - A common structure is required for all projects. This enables a tool developed in one project to be used without change on another. General purpose tools can be designed to run against any project. The standard form defines a set of functions and structures that every project will embody.

Ease Of Operation - The environment must be easy to operate. I've got contractors working for me. What incentive do they have to work my system if it's hard? For one thing, they'll charge me more money. The other thing is, they just won't do it. There's not really not much I can do about it after the fact, two weeks from the deadline. If they haven't followed the rules, what am I going to do, shoot them? The environment must sell itself by providing more benefit to the user than cost. If we make the right methods the path of least resistance, the developers will naturally use them.

No Human Interlocks - No human interlocks are permitted. A developer, working at 3 a.m., who runs into a need for something out of a human librarian is going to give up and go to bed himself, when that librarian is in bed, unable to satisfy his request. Thus the environment is designed to support 24 hour, 7 day operation. Any developer can serve himself as librarian. The environment ensures that records are kept, but no one person is required.

Management Visibility - The environment must provide Management Visibility. In product development it is notoriously hard to track progress, particularly in the integration phase. This system provides a number of measures of progress, that are available to management.

Metric Capture - We use the environment to do metric capture. The metric capture hasn't been thought through as well as I'd like. But we are collecting some elementary measures, and we have the hooks to capture others.

#### DEVELOPMENT MODEL

Let me briefly examine our method, or our model, for development.

1. Functional Requirements
2. Structured Analysis and Design
3. Coding
4. Code Reveiw
5. Unit Test and System Test in Simulation
6. Target Environment Specific Tests
7. Deliver, Sell, or Use
8. Maintain and Enhance

It is pretty standard. We go through functional requirements, which, in our case, become Request for Proposals (RFP) to contractors. The contractor arrangement and a formal RFP help us resist the natural impulse to change the specifications during development. On the current effort we have made only two changes in the specification, both for fatal ommisions. I give this arrangement a great deal of the credit for the high productivity we have experienced on our current project. Then structural analysis and design is used to turn that set of functional requirements into a software design. The design is documented in a design document.

The code is developed in a top down fashion, using a packaging concept to break the work down into managable units. The code is placed in the CMS library immediately after it compiles without error.

Once code is in the library a code review takes place. We have a permanent person whose job is primarily to do code review. This results in very good code reviews. It's his job, and he has a vested interest in doing it right. Too many shops play lip



Clifford J. Schornak, II

Innovative Technology, Inc.  
1000 Holcomb Woods Parkway, Suite 422  
Roswell, Ga.

#### ABSTRACT

This paper discusses the development environment used at the author's company. This company develops microprocessor based products with software and hardware components. The company has a unique organization and makes heavy use of telecommuting in its operation. The development environment consists of a number of automated tools and simulation techniques. The results of the application of this environment for two projects are presented.

#### INTRODUCTION

I am Director of Development for Innovative Technology, Inc. The company is about 3 years old, but has been active six months. In fact, I think my first official duty was to come to the last DECUS. In that time we've developed and displayed at Comdex, our first product, an peripheral to the IBM PC. We developed that product using the methodologies presented here.

The following topics are discussed:

1. Our corporate culture and facilities
2. The development environment we've made to support that corporate culture
3. The results of two projects developed under that environment.

Although Innovative Technology, Inc. is only six months old, three of the people involved, have been working together now for 2 1/2 years. The methodologies you see here are the third generation system that we have tried to put together during that time. There has been a lot of thought put into this environment and quite a bit of operational experience.

#### Corporate Environment

In general, we develop hardware and software products. That is, the product itself has a component that is hardware and is software, so we're not just a software shop. Typically, they're small projects -- small in terms of what some people report around here. We usually have two man-months of effort as a low end and four man-years for the maximum. Typically, a

microprocessor is the target operating environment, either as a dedicated stand-alone piece of equipment, as a microcontroller, or as imbedded in another piece of equipment such as an IBM PC.

Each employee has a terminal on his desk at the office, and a terminal, printer and 2400 baud modem at home. This is standard equipment issue for any employee. It is typical to see two and three people logged in at 2 o'clock in the morning.

We do all of our development with contract labor. Right now there are six employees of the company, and we have ten contractors. My department, Product Development, has five contractors, and only one employee, me. These contractors work from their offices, using a standard issue workstation.

We have a VAX 750 for a development engine. The system is equipt with 300 megabytes of disk, 4 megabytess of memory, and all the data communications you can imagine.

We are targeting for a number of different microprocessor environments. If we did development on the target machines, which is the tendency for most people who do development, we would be throwing away suites of tools every other day. Each project would be a whole new suite of tools. That is awfully expensive. Further, development on typical microprocessors results in a host of problems inherent in PC based development.

#### Goals For The Environment

When we formed Innovative Technology, the first thing I did, before writing the







The "0" is used to show that the dot is not printed and the "a" is used to show the dot is printed.

To use this table the top row of 3 dots should be placed on the top of the 3 dots down the left side to get the full six dot pattern. The character at the intersection of the line and row will make the composite pattern.

I have found that the easiest way to create images with these characters is to first create a grid that corresponds to the dot spacing. The six vertical dots occupy 1/12 of an inch or 72 dots per inch. If I select <Esc>P9q as the horizontal density then I can use a square grid. If I select <Esc>Plq then the horizontal axis must be double the vertical to accomodate twice as many dots per inch.

I draw the image on the grid and mark to dots I want to print. Then I divide the grid into six dot rows. Using the table above I can select the characters to replicate the image on the LA-100.

Our logo **MITRE** for example.

#### DOT PATTERN SET

The characters used for printing are formed much the same way as described above. A typical character cell is composed of a 9 by 15 matrix of dots. Usually only the set 8 by 10 are used to form the character. If you look at a down line loadable character set, it will consist of repeated patterns like the following selection from the Equation and Greek character set:

| These codes         | Print as |
|---------------------|----------|
| <ESC>P1;1;1;4{1     |          |
| KScCcSK?/????@????; | ▽        |
| Wcd~dcW?/?ABA???    | ⊕        |
| ?@~@B??/?ABA???     | Γ        |
| OGGO__0?/????????;  | ~        |
| GCCGOOG?/@@@@@@?;   | ≈        |
| wCQQQCw?/?@AAA@??;  | ⊙        |
| ?CgOgC??/?@??@??;   | ×        |
| ?oKAKo??/BA??AB?;   | △        |
| <ESC>\              |          |

The first 8 characters form the top half of the image and the characters following the "/" form the bottom half of the image.

The escape sequence on the first line instructs a device (VT220 in this case) to load the characters into a font position. From 1 to 94 images can be defined this way. The numbers in the escape sequence tell the device to (in sequence) use font 1, the starting character is 1, to erase only the characters being reloaded (1), the matrix size 4 means 7 by 10, the width defaults to 80 columns, and the final 1 indicates a text cell.

So far as I know you cannot down-load fonts to LA-100's. But it is useful to understand how the printer might handle printing images with this technique.

| Character Set              | Name | Code     | Value |
|----------------------------|------|----------|-------|
| USASCII                    | B    | <Esc>P6q | 110   |
| Italian                    | Y    | <Esc>P7q | 94    |
| Equation and Greek         | >    | <Esc>P8q | 83    |
| Digital VT100 line drawing | 0.   | <Esc>P9q | 74    |

These Escape sequences load the character sets into the G0 - G3:

```
<Esc>(B loads G0 with B USASCII
<Esc>)0 loads G1 with 0 line drawing
<Esc>*> loads G2 with > Greek
<Esc>+Y loads G3 with Y Italian.
```

It does not matter where these are installed in the printer. If it is not there the default character set will be loaded.

Then, if I have done my homework right, this sequence

```
<Esc>*B<Esc>+0<Esc>n<Esc>|
```

will permit me to print normal ACSII characters with the seven bit codes and the line drawing set with eight bit codes without making any other changes.

To do the same thing in just a seven bit environment I would have to switch the GL set as follows

```
<Esc>*B<Esc>+0 to load G2 & G3
```

then

```
<Esc>n to print USASCII
<Esc>o to print lines.
```

This might seem like a lot of trouble. If I have only one or two characters I want to add to my text I could do this

```
<Esc>0x (a single character from G3)
```

where x is the line character that I wanted to include in my text. This would have to be repeated for each character I wanted to substitute. As you can easily see, this could become quite messy and very difficult to edit. For complicated images I recommend you place them in a separate file and include them in your text only when you want to print them.

### GRAPHICS

The LA-100 has a graphics mode that allows you to control the printing of each dot in the print head. You can also control the horizontal density of the dots. The sequences that places the printer in graphics mode are:

| Code     | dots per inch |
|----------|---------------|
| <Esc>P1q | 132           |
| <Esc>P2q | 330           |
| <Esc>P3q | 220           |
| <Esc>P4q | 165           |
| <Esc>P5q | 132           |

In this mode spaces and carriage returns are ignored. To return to normal text mode you would use

```
<Esc>\
```

The characters sent to the printer in graphics mode have different functions than they do in text mode. The characters that control the position where the next dot is placed are:

```
$ return to left margin
- same as $ but down 1/12 inch
? move 1 dot to right
!22? move 22 dots to right.
```

The "!" character is used to introduce a repeat of the next valid character some number of times.

The characters starting with "@" through "~" (in octal sequence) are used to determine which of the top six dots are to print. This is done by subtracting octal 77 from the binary value of the character. The dots are printed from top down if the corresponding bit is a 1 using the least significant bit first. The patterns are shown below:

| DOTS | 1 | 2 | 3 | 4 | 5 | 6 |
|------|---|---|---|---|---|---|
| 1    | 0 | 0 | 0 | 0 | 0 | 0 |
| 2    | 0 | 0 | 0 | 0 | 0 | 0 |
| 3    | 0 | 0 | 0 | 0 | 0 | 0 |
| 4    | 0 | ? | @ | A | B | C |
| 5    | 0 |   |   | D | E | F |
| 6    | 0 |   |   |   |   |   |
| 4    | 0 | G | H | I | J | K |
| 5    | 0 |   |   |   |   |   |
| 6    | 0 |   |   |   |   |   |
| 4    | 0 |   |   |   |   |   |
| 5    | 0 | P | Q | R | S | T |
| 6    | 0 |   |   |   |   |   |
| 4    | 0 |   |   |   |   |   |
| 5    | 0 | W | X | Y | Z | [ |
| 6    | 0 |   |   |   |   |   |
| 4    | 0 |   |   |   |   |   |
| 5    | 0 |   | a | b | c | d |
| 6    | 0 |   |   |   |   |   |
| 4    | 0 |   |   |   |   |   |
| 5    | 0 | g | h | i | j | k |
| 6    | 0 |   |   |   |   |   |
| 4    | 0 |   |   |   |   |   |
| 5    | 0 | o | p | q | r | s |
| 6    | 0 |   |   |   |   |   |
| 4    | 0 |   |   |   |   |   |
| 5    | 0 | w | x | y | z | { |
| 6    | 0 |   |   |   |   |   |

increment the line counter. Failure to do this will cause the top margin to creep in one direction or the other on subsequent pages. These partial line commands are used primarily for sub and superscripting where they would normally be used in pairs anyway.

### TEXT PRESENTATION

Now that you know how to position your text on the page, you are ready for some other interesting escape sequences.

| ON         | OFF        | USAGE          |
|------------|------------|----------------|
| <Esc>[2"z  | <Esc>[1"z  | letter quality |
| <Esc>[?29h | <Esc>[?29l | font pitches   |
| <Esc>[?7h  | <Esc>[?7l  | auto wrap      |
| <Esc>[20h  | <Esc>[20l  | line feed      |
| <Esc> G    | <Esc> F    | C1 transmit    |
| <Esc> 7    | <Esc> 6    | C1 receive     |

The first set of codes allow you to dynamically switch the printer between letter and draft print quality. This can be useful if the printer is not located near you. To make use of the automatic switching between draft and letter quality the manual controls on the printer must be disabled.

When you use the sequence <Esc>[?29h the printer will not accept horizontal pitches greater than 12 characters per inch. If you need 13.2 or 16.6 characters per inch, you must precede these with <Esc>[?29l.

The autowrap feature controls what happens when you try to print beyond the right margin. With autowrap on <Esc>[?7h printing continues on the next line. When off <Esc>[?7l printing of any remaining characters is terminated until a new line is started with a line feed character.

The line feed feature controls what happens when a line feed character (octal 12) is sent to the printer. When on <Esc>[20h the next character is printed at the left margin on the next line down. When off <Esc>[20l the next character is printed at the current position on the next line down.

The C1 transmit and receive codes instruct the LA-100 to accept eight bit control codes. To function properly you will also need to set the printer to eight bit communication environment (this is the eighth switch in the "A" bank of switches inside the printer - see Table 3-2 in the "Operator Guide). In this mode the printer will print seven bit or eight bit characters depending on what is received.

Turning C-1 transmit and receive off causes only the control codes to be sent in seven bit mode. The printable characters will still be sent in eight bit mode. Also turning off the eight bit communications environment causes the data to be sent in seven bit mode (of course).

There are two main advantages to using the eight bit environment. The number of characters required to be transmitted to the printer is less and the number of selectable printable characters is doubled (188 vs 94).

You may think of these as two 94 byte buffers (I use the word buffer very loosely here, it might even be a process or a pointer.) which the documentation refers to as GL and GR. GL refers to the seven bit codes (octal range 40 - 177) and GR to the eight bit codes (octal range 240 - 377).

You may be curious as to how these characters are formed and get loaded into GL and GR and selectively printed. I'll tell you as much as I can; but be forewarned that what follows is only my understanding of what happens and is not necessarily technically correct.

There are four intermediate buffers called G0, G1, G2, and G3. These can be loaded with the dot pattern corresponding to the code (7 or 8 bit) that the character translates to. These patterns are referred to as DPS (Dot Pattern Set) in the documentation. I'll get to how these are formed later.

Any of the buffers G0 - G3 may be selected for seven bit printing environment GL. There are two ways to do this:

| Code    | Octal Code | Selects    |
|---------|------------|------------|
| SI (^O) | 017        | G0 into GL |
| S0 (^N) | 016        | G1 into GL |
| <Esc>n  | 033 156    | G2 into GL |
| <Esc>o  | 033 157    | G3 into GL |

or for a single printable character

|        |         |     |
|--------|---------|-----|
| <Esc>N | 033 116 | G2  |
| <Esc>O | 033 114 | G3. |

You can only select G1 - G3 for printing in the eight bit environment. There are two ways to do this:

| Code   | Octal Code | Selects    |
|--------|------------|------------|
| <Esc>~ | 033 176    | G1 into GR |
| <Esc>} | 033 175    | G2 into GR |
| <Esc>  | 033 174    | G3 into GR |

or for a single printable character

|     |     |     |
|-----|-----|-----|
| SS2 | 216 | G2  |
| SS3 | 217 | G3. |

So - you think this is confusing and complicated? You might ask what is in G0 - G3 and how does it get there? Depending on what LA-100 you have, it came with one or more character sets in read-only memory. You may obtain other character sets in either chip or cartridge form. Each of these character sets has a unique name. A partial list of these names can be found in Table 3-2 of the Programmer Reference Manual. The ones I will use here are:

down <Esc>[1e

|                  | partial     | full        |
|------------------|-------------|-------------|
| <u>direction</u> | <u>line</u> | <u>line</u> |
| up               | <Esc>L      | <Esc>M      |
| down             | <Esc>K      | <Esc>D      |

(Where: l=line, n=number of lines)

The escape sequences to define the page length might look like this:

<Esc>[1z<Esc>[66t

The first code tells the LA-100 that the number of lines per inch is 6. The second code indicates that there are 66 of these lines on a page. This is equivalent to eleven inches. You might additionally want to tell the LA-100 that the first print line is line 6 and the last line to be printed on is line 60. To do this you would code:

<Esc>[6;60r

You will recognize this as the same code that sets the scroll region for the VT-100 terminals (with slightly larger values). Using this command will cause the LA-100 to immediately reset top of form where it is now and then advance the paper to print on the sixth line. As a precaution you should check the setup on the printer personally.

If the paper is not where you expected you can manually reset top-of-form with the following steps:

1. stop the printer queue
2. submit your print request to the queue you intend to use
3. halt the printer with the off-line button
4. position the paper to print just below the physical top-of-form
5. press the "SET TOF" button
6. restart the printer
7. start the queue to the printer.

I strongly recommend that you define and use the /FORMS\_TYPE= qualifier for queues that go to LA-100 printers.

There is no simple way to recover the original top-of-form without manually resetting the printer. If you are adventuresome you might try the following:

<Esc>[132d<Esc>[60t  
<Esc>[132d<Esc>[66t

This sequence assumes the vertical pitch was set at 6 lines per inch and the top and bottom margins were set to 6 and 60 respectively. The first code <Esc>[132d forces the LA-100 to the top of the next page (line 6). The next sequence redefines the page length to 60 lines and then the top and bottom margins to line 1 and line 60. This should cause the next code <Esc>[132d to position the paper at the

first print line below the perforation. The following codes should then cause the LA-100 to properly align the paper with its definition of the form. Try it and see for yourself.

Setting the vertical pitch does not change the top and bottom margins. Changing the vertical pitch also does not change the top and bottom margins. You may notice that the next print line (after a pitch change) is not where you expected it to be. The LA-100 has a simple rule to follow. That rule is

The next print line is an integer multiple of the pitch from the top-of-form.

Vertical tabs are set by default at every line. This makes the CR LF and VT act the same. Setting vertical tabs therefore has no effect until you clear all vertical tabs with <Esc>[4g. Setting and clearing vertical tabs at the active line can be done with these codes:

<Esc>3 set  
<Esc>[lg clear

If you have a defined set of tabs you might want to use the following:

<Esc>[4g<Esc>[10;20;30;40v

This would clear all vertical tabs and then set tabs at lines 10, 20, 30, and 40. The actual position of the tabs would then depend on the vertical pitch at the time the VT was sent to the printer.

You have a rich set of vertical control codes available on the LA-100. If you are clever you can create some very entertaining printing. Beware though -- the codes that cause the printer to reverse index do not work well unless the paper is on the platen as opposed to the forms tractor.

You can set the vertical position within the top and bottom margins with this code: <Esc>[ld where l is the line number. As we saw earlier an attempt to go beyond the bottom margin will set the form to the top margin on the next page. Again the actual position will be determined by the vertical pitch in effect at the time the sequence is sent to the printer. The following two sequences do not produce the same results:

<Esc>[1z<Esc>[12d two inches from top  
<Esc>[4z<Esc>[12d six inches from top.

Armed with both horizontal and vertical control, you can now determine precisely where the next character will be printed.

Unlike the relative horizontal codes, the relative vertical codes can go in either direction. There are also partial line up and partial line down commands. These must be used in equal numbers since they do not



In the first case the <<fontl>> may appear anywhere in the text. The <<file=0;fontl>> must appear on a line by itself. (Note that the << >> are used to represent a single <> combination.)

### HORIZONTAL PAGE CONTROLS

Pitch: <Esc>[nw

| <u>cpi</u> | <u>n</u> | <u>cpi</u> | <u>n</u> |
|------------|----------|------------|----------|
| 10         | 1        | 5          | 5        |
| 12         | 2        | 6          | 6        |
| 13.2       | 3*       | 6.6        | 7*       |
| 16.6       | 4*       | 8.25       | 8*       |

\* Use pitch code <Esc>[?29l  
not <Esc>[?29h

Width: <Esc>[c;n"s  
Margins: <Esc>[c;cs

Tabs:            single        multiple  
set:            <Esc>l        <Esc>c;c...u  
clear:          <Esc>[0g    <Esc>[2g

Position:  
absolute: <Esc>[c`  
relative: <Esc>[na

(Where: c=column, n=number of columns)

The escape sequence to set the left margin of the paper 3 inches to the left might look like this:

<Esc>[lw<Esc>[30;l15s<Esc>[30`

Where the first code sets the horizontal pitch to ten characters per inch, the second code sets the left margin at column 30 and the right margin at column 15. The last code is optional and forces the next printable character to go in column 30.

If you wanted to change the horizontal pitch to 13.2 characters per inch, you would use this sequence:

<Esc>[?29l<Esc>[3w<Esc>[40;l52s<Esc>[40`

Here you will notice that the first code tells the LA-100 that it is to use all pitches. This is necessary prior to setting the pitch to 13.2 characters per inch. Notice also that the left margin is now at column 40. This is still about 3 inches from the left edge of the platen.

Horizontal tabs are set at every 8 columns by default. They can be set with <Esc>l (one, not lower case L). This code sets a tab at the active column. You may find this quite useful if you want several lines of text to all start at the same place, but the place is relative to where the next printable character falls in the text. To be sure this is the only active tab you can use this sequence <Esc>2g<Esc>l, which

first clears all horizontal tabs and then sets a single tab at the active column.

If you require several tabs to be set at predetermined positions, this sequence might be useful to you:

<Esc>[l0;20;30u

Here we set tabs at columns 10, 20, and 30. The number of tabs is a variable; for example, the sequence <Esc>[l;5;7;9;11;13u is also correct.

To selectively clear a tab you can set the active column at the tab and use <Esc>0g.

There are two other codes that will help you control the horizontal position of your text. We have already visited the absolute position sequence <Esc>[c`, but it needs a little more explanation of its function.

The "c" is a decimal number that specifies the column relative to the leftmost printable position (absolute zero). The physical position is calculated by dividing "c" by the horizontal pitch specified by the sequence <Esc>[nw. This answer is in inches.

If you specify "c" outside the bounds determined by either <Esc>[c;cs or <Esc>[c;n"s then "c" will be set to the closest legal margin.

The other escape sequence you might use is one that repositions the active column to the right by the number of columns specified. This code is:

<Esc>[na

where "n" is the number of columns displaced to the right of the current active print position. Again, if this moves the print position beyond the right margin then the print position is set at the right margin

### VERTICAL PAGE CONTROLS

Pitch: <Esc>[nz

| <u>lpi</u> | <u>n</u> | <u>lpi</u> | <u>n</u> |
|------------|----------|------------|----------|
| 6          | 1        | 2          | 4        |
| 8          | 2        | 3          | 5        |
| 12         | 3        | 4          | 6        |

Lines: <Esc>[nt  
Margins: <Esc>[l;lr

Tabs:            single        multiple  
set:            <Esc>3        <Esc>[l;1...v  
clear:          <Esc>[lg    <Esc>[4g

Position:  
absolute:        <Esc>[ld  
relative: up    <Esc>[lA

## PAPER ALIGNMENT

Even before we moved the paper to the center of the platen, we noticed that the printers would not maintain top-of-form synchronously with the perforations in the paper. One document might leave the paper somewhere other than at the top-of-form. The next document would then start where the other left off.

To further compound our problems, we noted that some users were sending binary files to the printers. Aside from making a mess and a lot of noise, the binary codes occasionally matched something the LA-100 understood. These matched codes would cause unpredictable results. The worst case was when the left and right margins were redefined as adjacent columns. With auto wrap still on, it printed everything in those two columns.

## FONT AND PITCH CHANGES

The MASS-11 product has a set of printer personality tables. In these tables are the codes (escape sequences ie. terminal commands beginning with octal 33) that set the printer to the desired characteristics for the document being printed. This is where we defined the left and right margins for the new position of the paper. You can also define the codes to set the printer for the page length, top and bottom margins, print quality, font selection, and vertical or horizontal pitch selection.

As you might suspect, there are a lot of different escape sequences that need to be specified to set the LA-100 for all the options it supports. Our problem was to fit these sequences into the space provided by the MASS-11 printer personality tables. This turned out to be a not so trivial a pursuit.

Having established what we considered a reasonable set of escape sequences in the printer personality tables, we discovered that the LA-100 would not maintain the prescribed attributes for more than one page.

It appears that the LA-100 would reset certain parameters after it received a form feed instruction. In particular we noticed that the printer would revert to left margin of one (not 30 as we needed for the position of the paper). Also we noted that the font and horizontal pitch would change somewhat unpredictably. Most often though it would change to draft quality and ten pitch.

Now we needed to tell the LA-100 the set up escape sequences at the top of every page. This is a function that is not supported by the printer personality tables.

## SOLUTION

To solve these problems we had two issues that we had to deal with. First, MASS-11 does not permit you to enter the <ESC> character into the text directly. Second, we still needed to be considerate of the users and not impose an unnecessary burden on them.

There are two different methods that can be used to enter the escape sequences into your documents.

1. MASS-11 provides for character substitution at print time. Using this capability you can sacrifice one character and MASS-11 will replace it with the <ESC> character prior to sending it to the printer.

2. MASS-11 provides for graphics integration with text. In this situation you can simply code your escape sequences in a file using EDT. Then tell MASS-11 where to include these in your documents.

Both of these methods have their advantages and disadvantages. Using character substitution you can have more direct control over the escape sequences you can use. Standard sequences can be placed in a list document and included in your document by using the list processing features of MASS-11. This method is easier to setup and maintain. It also makes it easier for the users as it functions very much like the font and pitch commands native to MASS-11.

The disadvantages of the character substitution are that merging documents in MASS-11 creates additional processing overhead and printing delays. It also makes it more difficult to use list processing in the document (you must merge the document twice). It is sometimes inconvenient to sacrifice one character all the time; but MASS-11 permits you to select the character you want to substitute, or select a second substitute character.

Using graphics integration frees you from the problems of character substitution and provides relatively clean and faster processing text.

The disadvantages of graphics integration are that it is somewhat more awkward to work with and does not give you the degree of direct control that is sometimes handy. This method is also more difficult to setup and maintain because each escape sequence must be placed in a separate file and (for the users convenience) given a logical name equivalence.

From the users perspective the two methods appear similar:

<<font1>> for character substitution

<<file=0,font1>> for graphics.

# THE LA-100 AS A SHARED RESOURCE

Richard G. Fulton  
The MITRE Corporation  
Houston, Texas

## ABSTRACT

This paper describes how you can make effective use of the Letterprinter 100 as a shared resource in a text processing environment. The examples show how MASS-11 and the LA-100 can be made to work together to produce consistent quality papers with minimal burden to the users. You will learn what does and does not work, and how to integrate graphics images with your text.

## INTRODUCTION

Let me first tell you a little about the environment where we use the LA-100. We provide technical services to the National Aeronautics and Space Administration (NASA). Our products are printed documents in the form of technical reports, working papers, and briefings.

About a year ago we installed a VAX 11/750 running VMS and MASS-11. There are 32 terminals connected to the computer, including three LA-100's. The terminals are located throughout the offices and general areas. The LA-100's are in unattended areas where all employees have easy access.

Just for reference, the workload on the LA-100's has been such that two of them have already needed repair for normal use and abuse.

Our initial experience with these printers was discouraging and frustrating. Some of the more serious problems we encountered include:

- o paper jams
- o paper miss-alignment
- o font and pitch setup changes

Clearly, something needed to be done to restore the productivity of our staff and regain confidence in the word processing functions.

## PAPER JAMS

The LA-100 has a 15 inch platen with optional tractor form feed. You can use the platen without the tractor -- but not with continuous form paper. Any miss-alignment of the paper on initial installation will eventually cause the paper to creep across the platen and crumple up on one side or the other.

For best results we found that the forms tractor performs more reliably. There are, however, problems with narrow forms. We use eight and one half inch by eleven inch paper in our LA-100's. Even with the forms tractor we experienced considerable difficulty with the paper tearing off.

There are three principle causes for this paper tearing and they all relate to increased paper tension on the left side of the platen.

First, there is a micro switch that determines out-of-paper condition. This is a troublesome device that also makes it difficult to put the paper in at all.

Second, there is a plastic band that holds the paper against the platen. The shape of this piece of plastic causes additional pressure on the paper at each end of the platen. When using narrow paper this causes the paper to drag slightly on the left side.

Finally, on some of the earlier LA-100's the power supply was installed incorrectly. The position of the power supply is such that in the incorrect position it caused additional pressure on the left side of the platen. This is now a known problem and if you suspect you may have it your service representative should be told to investigate.

One solution to the paper jamming problems we experienced was to move the paper to the center of the platen. Specifically, we placed the left perforation of the paper three inches to the right of the left edge of the platen. Now the paper stays where we want it and it feeds continuously with no evidence of the prior problems.

By solving one problem we introduced new ones. Now we must tell the printers where the paper is, and do so in a way that our 50 users are not inconvenienced. This is the exercise that prompted this paper.





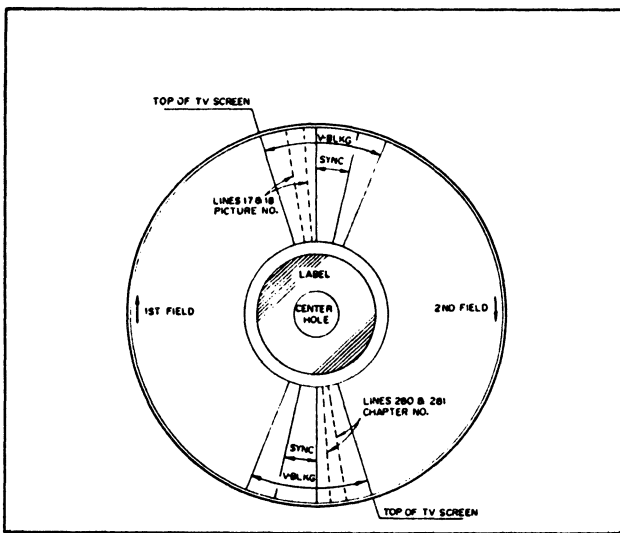


Fig. 4. Relationship of the Video Frames to the Laser Disc

The laser video disc can be mastered via two methods, CAV or constant angular velocity and CLV or constant linear velocity. The CLV method is used for video movie recordings and is not suitable for still frame information. The CAV method, as diagrammed in Fig. 4., allows for one complete video frame to be recorded per revolution of the disc.

REFERENCES

- (1) Daynes, R., *Byte*, Vol. 7, June 1982, pp48-59.
- (2) Conrac Corporation, *Raster Graphics Handbook*, Conrac Corporation, Convina, CA, 1980.
- (3) Foley, J. D. and A. Vandam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading MA, 1982.
- (4) Greenberg, D., A. Marcus, A. H. Schmidt, and V. Gorter, *The Computer Image*, Addison-Wesley, Reading MA, 1982.
- (5) Hurn, Bruce, *VideoPRO*, Vol. 3, No. 11, Dec. 1984, pp37-44.
- (6) Newman, T., *Museum News*, Vol. 59, No. 4, Jan./Feb. 1981, pp28-33.
- (7) Newman, W. and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, 1979.
- (8) Nyerges, Alexander, Lee, *Videodisc/Videotex*, Vol. 2, No. 4, Fall 1982, pp267-274.
- (9) Waite, M., *Computer Graphics Primer*, Howard W. Sons, Indianapolis, IN, 1981.
- (10) Digital Equipment Educational Services Interactive Video Information System (IVIS) pamphlet and videodisc.
- (11) Sony Overview Information Disc. Sony Corporation.

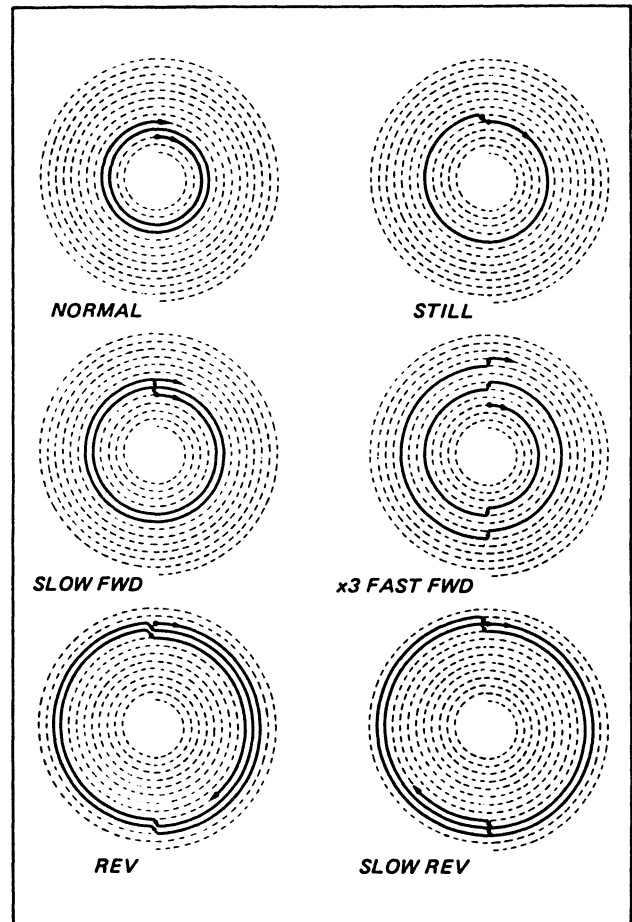


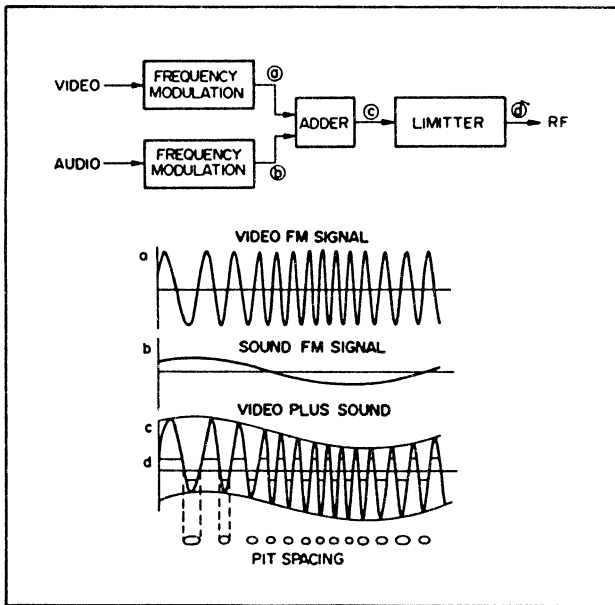
Fig. 5. Laser Beam Trace in Various Modes of Operation

The laser disc player allows for several modes of playback. The one method of interest is the still or freeze frame mode, allowing for the random retrieval and display of a single frame of visual information. In the "still" mode of operation, the player displays a single spiral track from the disc and then the beam jumps back one track and repeats the process. During each of the "trick" modes of operation, the laser beam is shifted forward or back during the vertical synchronization portion of the video frame (refer to Fig. 4.)

(picture elements) comprising the display on the color monitor.

To achieve the quality of conventional TV receivers, approximately 480 lines of 512 pixels per line would be needed (245,760 pixels per display frame). Each pixel would need to represent many values of each of the red, green and blue colors. An 8 bit byte could be assigned to each of the colors. Therefore, each TV frame could be represented by  $3 \times 245760$  bytes of digital information or approximately 0.7 megabytes for a single frame. Not too unreasonable considering today's digital disk devices.

Now consider the task of storing 54,000 color video images that a laser videodisc is capable of storing. Our digital storage requirements become immense ( $.7 \times 54,000 = 37,800$  megabytes): approximately 37.8 gigabytes.



THE LASER VIDEO DISC AND THE RECORDING/PLAYBACK PROCESS

Fig. 1. Encoding of Information onto the Disc

A composite signal is produced by electrically adding the video signal with two channels of audio. The signal is limited (clipped) as to amplitude resulting in a waveform with peaks that vary in width corresponding in signal amplitude and peaks that vary in period corresponding to signal frequency. The analogy is the width and spacing of pits burned into the photoresist of the laser disc master - the goal of the mastering process.

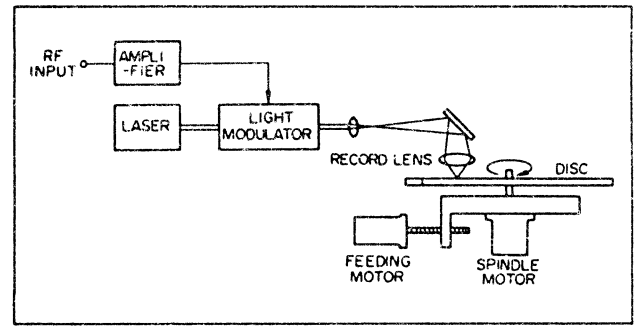


Fig. 2. The Recording Process

A finely focused laser beam is modulated by the signal as illustrated in Fig. 1. The laser produces pulses of high energy coherent light that vary in intensity and period. As the disc is rotated, the laser burns the pits into the photoresist of the mastering disc in an inward spiral pattern. The information content of the video and audio signal is transformed into physical attributes of the laser disc media. It should be noted that the process is an analog, as opposed to a digital, process since the physical pits vary continuously.

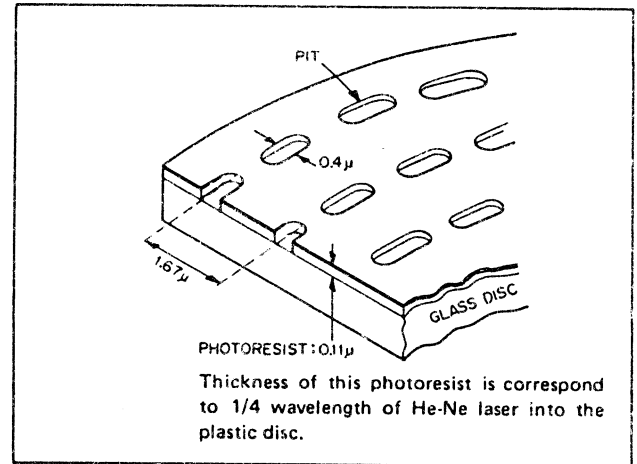


Fig. 3. Physical Dimensions of the Laser Disc Tracks

The circular tracks on the disc are spaced  $1.67 \mu$  (thousandth of a millimeter), resulting on 54,000 tracks in the space of approximately 9 centimeters (15,240 tracks per inch). It is interesting to note a comparison with magnetic disk media (hard and floppy disk) track spacing of approximately 100 tracks per inch. The overall objective is to utilize the extremely high density of optical recording media.

Just as with ARTSearch, the manual cataloging system has two major components, a limited cross indexed/master filing card system, and an incomplete photographic record system using prints or slides. This manual system, to work well, requires three efficient and accurate procedures: 1) information about an object must be easily entered into the card system and easily changed; 2) information about an object must be easily located and retrieved from the card system and the image base (photographs); 3) the image base of photographs must be as complete as possible and of high quality. For a large collection, each of these three procedures was difficult to execute with any degree of satisfaction. To summarize the procedural problems with the manual system, the textual side of the information system must be subject to constant updating, retrievals must be fast, accurate and complete and the visual records of the objects must not be subject to change. With ARTSearch, 1) and 2) are addressed using conventional computer data base programs and a not so conventional microcomputer workstation controlling the retrieval of visual images from the laser videodisc player. Step 3) requires techniques for photographing and mastering a laser videodisc. A process that results in archive quality images that prints and slides could not come close to achieving.

The gains of a computer driven information systems are realized in its use as a productive teaching and research tool. They are mainly ones of speed, accuracy and completeness of use. The performance of computer database systems is well documented and will not be discussed at length here. The complete database is stored on a mainframe computer (VAX 780) and is managed by DATATRIEVE. Subset retrievals on the microprocessor (Rainbow 100+) are managed by the KNOWLEDGE MANAGER database system. The interesting aspect of ARTSearch retrievals is the use of the laser videodisc player as the storage medium for the visual images. The initial intent in the use of laser disc players, both in the industrial and home entertainment markets, was for conventional video presentations and movies. Just as with other video recording devices, the laser player has two modes of operation, one of which allows for freeze action or single frame display. This is the capability that is used by the ARTSearch system.

Before this capability is described in more detail, one may ask: since we are using a microcomputer based system, why not use the color graphics capabilities of a micro for the presentation of the video images. The answer is that we could, however there is an economic consideration. The problem lies not with the color monitor that is needed to display a high quality video image, since microcomputer color monitors are similar in quality and use as that which would be needed with a laser player. The video signals that drive such monitors are actually very similar. The problem is

in the storage of the visual information that makes up the color video image. The amount of digital memory (computer memory, both disk and RAM) required to contain the color and gray scale information of a high quality video image is enormous. This leads into the very roots of the ARTSearch project.

Prior to the Spring of 1982, Professor Pat Mansfield was involved in a project exploring the aspects of what might be called semi-structured computer aided design (as apposed to computer aided design or CAD). The heart of the system was a micro (Z-80 with an S-100 bus) controlling a video frame grabber. This allowed conventional video images, derived from a video camera, VCR, television receiver etc., to be converted to the graphics memory of the microcomputer. The intent here is to illustrate the transition that took place from this low cost image processing system to the concepts behind ARTSearch. The video frame grabber on this micro allowed the researcher to manipulate and design images using hardware and software techniques akin to those used in manipulating full computer generated screen graphics (line elements, arcs, solid fill etc.). Because of the low cost, this sytem lacked the necessary RAM memory to reproduce high quality color images, although false color mapping was possible. Addressing the cost of a computer graphics system as related to the resolution or amount of RAM memory one has at hand, the principle is a simple one, a large amount of memory cost a large amount of money. This relationship will always be true however, each year brings the cost of computer memory down by orders of magnitude, both rotating memories (disks) and random access memory (RAM) chips. As this paper is being prepared, RAM chips are commonly being used that store 256,000 (256K) bits (as opposed to byte is a string of eight bits) of information. Today, these 256K chips cost approximately \$20.00. A bit is the simplest representation of information within a computer, it is normally thought of as having a value of 0 or 1.

In an imaging system like the one investigated by Professor Mansfield, a bit can be used to represent a point in the graphics picture. The "0" and "1" value can be used to cause the point, or picture element (pixel) to be dark or light. However, in most systems, enough memory is used so that each pixel can be assigned enough bits of memory so that levels of gray and even color can be represented. In this system, 4 bits were assigned to each pixel allowing 16 levels of gray to be represented (i.e., a 4 bit binary number allows one to count from 0 to 15). What this is all leading to is a discussion of the memory requirements needed to store a full color video image. The other aspect of the computer image affecting the amount of memory needed is resolution; that is the number of rows and columns of pixels



Professor Patricia K. Mansfield  
and  
Computer Specialist, Michael K. Mansfield  
University of Wisconsin  
1300 Linden Drive  
Madison, Wisconsin 53706  
May 1985

#### ABSTRACT

The ARTSearch system provides for selective retrievals and simultaneous display of text information from a database with any one of 54,000 full color images of the artifacts from a laser videodisc. The system has been designed to provide full informational access to a collection of artifacts from a single workstation using the DEC RAINBOW 100+. DATATRIEVE is used to manage the entire catalog, stored on the VAX, with subset retrievals managed by the KNOWLEDGE MANAGER database system. The VAX DESIGN AUTHORIZING LANGUAGE enables the overlay of text and graphics onto random-accessed frames on the disc. The disc can therefore be involved in teaching the history of the artifacts (CBI); it can provide public information on the collection (exhibits, cultural events) and it can provide archival management. The target system used to run these applications is the DEC PROFESSIONAL 350 IVIS computer system.

The ARTSearch system can be expanded to incorporate an image processing component. This component utilizes the visual database of a collection as a source of images for artistic applications. The video frame grabber allows the personal computer to select images from the laser videodisc and to manipulate the images through color mapping tables and grey scales. Graphic tools allow one to manipulate individual pixels.

ARTSearch is the name for a University of Wisconsin-Madison project that has investigated the melding of laser videodisc technologies with microcomputer workstation data base techniques. The goal is to produce an interactive retrieval system which controls a dual base of information, both visual and textual for a museum collection. The industrial laser videodisc player is the natural choice for storing the visual side of the database since it has an excellent freeze frame capability and can be controlled both manually and via microcomputer for random access. Data base programs which have been in existence for many years could be used on micros and mainframe computers for the textual portion of our museum data base. Therefore, the ARTSearch system combines retrieval of text information from the computer database with simultaneous

display of one of 54,000 color images from the laser videodisc.

To test out the concepts imbedded in the idea of ARTSearch, two needs had to be met. One, we needed information for our database, again, both textual and visual, and two, a manual system must already be in place for managing this information, a system that could be converted to the computer. Just as the ideas for ARTSearch were evolving, the Helen L. Allen Textile Collection of the University of Wisconsin-Madison campus had needs that could benefit from the ARTSearch project. This museum collection was undergoing changes indicative of the quality of its contents. The collection was growing in size and the manual cataloging system was being tested to its limits. The textiles were being used for teaching and research purposes in ever increasing amounts, and pressure for access to the collection was growing. ARTSearch and the Helen Allen Textile Museum were ideal partners to test the ideas for this project.

The textile collection was not available for casual use by researchers, students and the interested public because it is not housed in a museum gallery environment. In addition, only a sparse photographic record existed. Presently, the curatorial staff must manually retrieve objects in response to requests. Therefore, the objectives of the ARTSearch system are: 1) to provide full informational access to the Collection from a single workstation; 2) to respond to the increasing demand to use the Collection by making it more accessible, without creating additional burdens on a limited staff; 3) to keep the objects in a better state of preservation by protecting them from unnecessary handling.

The base of the collection is approximately 12,000 textiles, costumes and related objects bequested by Professor Helen Allen, assembled during her 41 year tenure at the University (1927-1968). The mission of the collection is to provide educational resources that further the understanding of human beings within their material and social environments through the study of textiles of artistic, cultural and historic significance. The textiles range from pre-Columbian and Coptic fragments to contemporary fiberworks.



```

WRITE (6,9030) ' +-----+-----+-----+-----+'
WRITE (6,9030) ' | 13 | 14 | 15 | 16 |'
WRITE (6,9030) ' +-----+-----+-----+-----+'
WRITE (6,9030) ' | 17 | 18 | 19 | 20 |'
WRITE (6,9030) ' +-----+-----+-----+-----+'
WRITE (6,9030) 'now enter window (1-20) of plot to view.'
READ (5,9050) N_WINDOW
END DO
WRITE (6,9000) ' '
WRITE (6,9000) SET_080
9000 FORMAT (A)
9030 FORMAT (' ',A)
9040 FORMAT ('+',A)
9050 FORMAT (I3)
C
RETURN
END

```

```

IF (N_YWINDOW .LT. 0 .OR. N_YWINDOW .GT. 163) N_YWINDOW = 0
N_YWINDOW = (N_YWINDOW * 23) / 164
N_ADDWINDOW = N_XWINDOW + (528 * N_YWINDOW)
ELSE
IF (N_WINDOW .EQ. 4 .OR. N_WINDOW .EQ. 8 .OR.
- N_WINDOW .EQ. 12 .OR. N_WINDOW .EQ. 16) THEN
WRITE (6,9000) ' ENTER Y PICTAL OFFSET(0-163)'
READ (5,9050) N_YWINDOW
IF (N_YWINDOW .LT. 0 .OR. N_YWINDOW .GT. 163) N_YWINDOW = 0
N_YWINDOW = (N_YWINDOW * 23) / 164
N_ADDWINDOW = 528 * N_YWINDOW
ELSE
IF (N_WINDOW .EQ.17 .OR. N_WINDOW .EQ.18 .OR.
- N_WINDOW .EQ. 19) THEN
WRITE (6,9000) ' ENTER X PICTAL OFFSET(0-263)'
READ (5,9050) N_XWINDOW
IF (N_XWINDOW .LT. 0 .OR. N_XWINDOW .GT. 263)
- N_XWINDOW = 0
N_XWINDOW = N_XWINDOW / 2
N_ADDWINDOW = N_XWINDOW
ELSE
N_ADDWINDOW = 0
END IF
END IF
END IF
WRITE (6,9000) SET_132
WRITE (6,9000) SHIFT_IN_4
WRITE (6,9000) TOP_OF_PAGE
NFIRST = N_WINDOW - ((N_WINDOW / 4) * 4)
IF (NFIRST .EQ. 0) NFIRST = 4
NFIRST = NFIRST - 1
NSECND = (N_WINDOW - 1) / 4
J = (12144 * NSECND) + (132 * NFIRST) + 1 + N_ADDWINDOW
K = J + 131
WRITE (6,9040) LINE_OUT(J:K)
DO I = 1,22
J = J + 528
K = J + 131
WRITE (6,9030) LINE_OUT(J:K)
END DO
WRITE (6,9000) SHIFT_IN_1
READ (5,9000) INPUT_LINE
WRITE (6,9000) TOP_OF_PAGE
WRITE (6,9000) ' '
WRITE (6,9000) SET_080
WRITE (6,9030) 'The graphics area is divided into windows'
WRITE (6,9030) ' you must select to view. Each window is'
WRITE (6,9030) ' 264 pictals across X 164 pictals down.'
WRITE (6,9030) 'Choosing a window other than those listed'
WRITE (6,9030) ' will cause the graph to exit.'
WRITE (6,9030) 'You will not be prompted for but can enter'
WRITE (6,9030) ' next window number after the current'
WRITE (6,9030) ' window is displayed.'
WRITE (6,9030) 'The window arrangement is as follows:'
WRITE (6,9030) '
WRITE (6,9030) ' +-----+-----+-----+-----+'
WRITE (6,9030) ' | 1 | 2 | 3 | 4 |'
WRITE (6,9030) ' +-----+-----+-----+-----+'
WRITE (6,9030) ' | 5 | 6 | 7 | 8 |'
WRITE (6,9030) ' +-----+-----+-----+-----+'
WRITE (6,9030) ' | 9 | 10 | 11 | 12 |'

```

```

C*****
C
C DISPLAY THE DIGITIZED GRAPHICS ON THE VT220
C
C*****
C SUBROUTINE VTDISP
C
C CHARACTER*1 INPUT_LINE
C CHARACTER*2 SHIFT_IN_1,SHIFT_IN_2
C CHARACTER*3 SHIFT_IN_3,SHIFT_IN_4
C CHARACTER*4 TOP_OF_PAGE
C CHARACTER*6 SET_132,SET_080
C CHARACTER*4 INTO_G3
C CHARACTER*60720 LINE_OUT
C INTEGER*4 NXCUR,NYCUR,NEXTX,NEXTY
C INTEGER*4 N_WINDOW
C REAL*4 DELTAX,DELTAY
C REAL*4 XPRIOR,YPRIOR,XFOLLOW,YFOLLOW
C
C COMMON /VTGRAPH/ SHIFT_IN_1,SHIFT_IN_2,SHIFT_IN_3,SHIFT_IN_4,
- TOP_OF_PAGE,LINE_OUT,NXCUR,NYCUR,NEXTX,NEXTY,DELTAX,DELTAY,
- SET_132,SET_080,XPRIOR,YPRIOR,XFOLLOW,YFOLLOW
C
C WRITE (6,9000) ' '
C WRITE (6,9000) SET_080
C WRITE (6,9030) 'The graphics area is divided into windows'
C WRITE (6,9030) ' you must select to view. Each window is'
C WRITE (6,9030) ' 264 pictals across X 164 pictals down.'
C WRITE (6,9030) 'Choosing a window other than those listed'
C WRITE (6,9030) ' will cause the graph to exit.'
C WRITE (6,9030) 'You will not be prompted for but can enter'
C WRITE (6,9030) ' next window number after the current'
C WRITE (6,9030) ' window is displayed.'
C WRITE (6,9030) 'The window arrangement is as follows:'
C WRITE (6,9030) '
C WRITE (6,9030) ' +-----+-----+-----+-----+'
C WRITE (6,9030) ' | 1 | 2 | 3 | 4 |'
C WRITE (6,9030) ' +-----+-----+-----+-----+'
C WRITE (6,9030) ' | 5 | 6 | 7 | 8 |'
C WRITE (6,9030) ' +-----+-----+-----+-----+'
C WRITE (6,9030) ' | 9 | 10 | 11 | 12 |'
C WRITE (6,9030) ' +-----+-----+-----+-----+'
C WRITE (6,9030) ' | 13 | 14 | 15 | 16 |'
C WRITE (6,9030) ' +-----+-----+-----+-----+'
C WRITE (6,9030) ' | 17 | 18 | 19 | 20 |'
C WRITE (6,9030) ' +-----+-----+-----+-----+'
C WRITE (6,9030) 'now enter window (1-20) of plot to view.'
C READ (5,9050) N_WINDOW
C DO WHILE (N_WINDOW .GT. 0 .AND. N_WINDOW .LT. 21)
C IF (N_WINDOW .EQ. 1 .OR. N_WINDOW .EQ. 2 .OR.
- N_WINDOW .EQ. 3 .OR. N_WINDOW .EQ. 5 .OR.
- N_WINDOW .EQ. 6 .OR. N_WINDOW .EQ. 7 .OR.
- N_WINDOW .EQ. 9 .OR. N_WINDOW .EQ. 10 .OR.
- N_WINDOW .EQ. 11 .OR. N_WINDOW .EQ. 13 .OR.
- N_WINDOW .EQ. 14 .OR. N_WINDOW .EQ. 15) THEN
C WRITE (6,9000) ' ENTER X PICTAL OFFSET(0-263)'
C READ (5,9050) N_XWINDOW
C IF (N_XWINDOW .LT. 0 .OR. N_XWINDOW .GT. 263) N_XWINDOW = 0
C N_XWINDOW = N_XWINDOW / 2
C WRITE (6,9000) ' ENTER Y PICTAL OFFSET(0-163)'
C READ (5,9050) N_YWINDOW

```

```

C*****
C
C WRITE THE BITMAP TO BE DISPLAYED LATER ON THE VT220
C
C*****
C SUBROUTINE VTMAPS
C
C CHARACTER*2 SHIFT_IN_1,SHIFT_IN_2
C CHARACTER*3 SHIFT_IN_3,SHIFT_IN_4
C CHARACTER*4 TOP_OF_PAGE
C CHARACTER*6 SET_132,SET_080
C CHARACTER*4 INTO_G3
C CHARACTER*60720 LINE_OUT
C INTEGER*4 NXCUR,NYCUR,NEXTX,NEXTY
C INTEGER*4 NCURCHR,IL
C REAL*4 DELTAX,DELTAY
C REAL*4 XPRIOR,YPRIOR,XFOLLOW,YFOLLOW
C
C DIMENSION IL(6)
C
C COMMON /VTGRAPH/ SHIFT_IN_1,SHIFT_IN_2,SHIFT_IN_3,SHIFT_IN_4,
- TOP_OF_PAGE,LINE_OUT,NXCUR,NYCUR,NEXTX,NEXTY,DELTAX,DELTAY,
- SET_132,SET_080,XPRIOR,YPRIOR,XFOLLOW,YFOLLOW
C
C IF (NXCUR .LT. 0 .OR. NXCUR .GT. 1055) RETURN
C IF (NYCUR .LT. 0 .OR. NYCUR .GT. 344) RETURN
C INDEXX = NXCUR / 2
C INDEXY = NYCUR / 3
C IOFFSETX = NXCUR - INDEXX * 2
C IOFFSETY = NYCUR - INDEXY * 3
C NUMBIT = IOFFSETX + (2 * IOFFSETY) + 1
C NUMCHR = INDEXX + (528 * INDEXY) + 1
C NCURCHR = ICHAR(LINE_OUT(NUMCHR:NUMCHR))
C IF (NCURCHR .EQ. 63) RETURN
C
C NVAL = 2
C DO I = 1,6
C IL(I) = 0
C IL(I) = NCURCHR - (NCURCHR / NVAL) * NVAL
C NCURCHR = (NCURCHR / NVAL) * NVAL
C IF (IL(I) .GT. 0) IL(I) = 1
C NVAL = NVAL * 2
C END DO
C NVAL = 2
C
C IL(NUMBIT) = 1
C NCURCHR = 64 + IL(1) + IL(2) * 2 + IL(3) * 4 + IL(4) * 8 +
- IL(5) * 16 + IL(6) * 32
C IF (NCURCHR .EQ. 127) NCURCHR = 63
C LINE_OUT(NUMCHR:NUMCHR) = CHAR(NCURCHR)
C
C RETURN
C END

```

```

C*****
C
C SETUP TO WRITE THE BITMAP TO BE DISPLAYED LATER ON THE VT220
C
C*****
C SUBROUTINE VTPLOT (XPOSIT,YPOSIT,IFUNC)
C
C CHARACTER*2 SHIFT_IN_1,SHIFT_IN_2
C CHARACTER*3 SHIFT_IN_3,SHIFT_IN_4
C CHARACTER*4 TOP_OF_PAGE
C CHARACTER*6 SET_132,SET_080
C CHARACTER*4 INTO_G3
C CHARACTER*60720 LINE_OUT
C INTEGER*4 NXCUR,NYCUR,NEXTX,NEXTY
C REAL*4 DELTAX,DELTAY
C REAL*4 XPOSIT,YPOSIT
C REAL*4 XPRIOR,YPRIOR,XFOLLOW,YFOLLOW
C INTEGER*4 IFUNC
C
C COMMON /VTGRAPH/ SHIFT_IN_1,SHIFT_IN_2,SHIFT_IN_3,SHIFT_IN_4,
C TOP_OF_PAGE,LINE_OUT,NXCUR,NYCUR,NEXTX,NEXTY,DELTAX,DELTAY,
C SET_132,SET_080,XPRIOR,YPRIOR,XFOLLOW,YFOLLOW
C
C NEXTX = INT(1055.0 * XPOSIT / 10.55)
C NEXTY = INT(344.0 * YPOSIT / 8.19)
C XFOLLOW = (XPOSIT - XPRIOR) * 100
C YFOLLOW = (YPOSIT - YPRIOR) * 100
C LIMRES = 2 * INT(SQRT(XFOLLOW*XFOLLOW + YFOLLOW*YFOLLOW) + 1.0)
C ALIMRES = REAL(LIMRES)
C DELTAX = XFOLLOW * .01 / ALIMRES
C DELTAY = YFOLLOW * .01 / ALIMRES
C
C IF (IFUNC .EQ. 2) THEN
C CALL VTMAPS
C DO I = 1,LIMRES
C NXCUR = INT(1055.0 * (XPRIOR + DELTAX * I) / 10.55)
C NYCUR = INT(344.0 * (YPRIOR + DELTAY * I) / 8.19)
C CALL VTMAPS
C END DO
C NXCUR = NEXTX
C NYCUR = NEXTY
C CALL VTMAPS
C ELSE
C NXCUR = NEXTX
C NYCUR = NEXTY
C END IF
C XPRIOR = XPOSIT
C YPRIOR = YPOSIT
C RETURN
C END

```

```

TOP_OF_PAGE(2:2) = CHAR(27)
TOP_OF_PAGE(3:4) = '[H'
C
LOAD_LINE_1(1:1) = '+'
LOAD_LINE_1(2:2) = CHAR(27)
LOAD_LINE_1(3:5) = 'P1;'
C
LOAD_LINE_2(1:4) = ';1{@'
C
LOAD_LINE_4(1:1) = CHAR(27)
LOAD_LINE_4(2:2) = '\ '
C
INTO_G3(1:1) = '+'
INTO_G3(2:2) = CHAR(27)
INTO_G3(3:4) = '+{@'
C
CURSOR_OFF(1:1) = '+'
CURSOR_OFF(2:2) = CHAR(27)
CURSOR_OFF(3:7) = '[?251'
C
CURSOR_ON(1:1) = '+'
CURSOR_ON(2:2) = CHAR(27)
CURSOR_ON(3:7) = '[?25h'
C
DO I = 1,60720
 LINE_OUT(I:I) = '@'
END DO
C
SET_132(1:1) = '+'
SET_132(2:2) = CHAR(27)
SET_132(3:6) = '[?3h'
C
SET_080(1:1) = '+'
SET_080(2:2) = CHAR(27)
SET_080(3:6) = '[?31'
C
WRITE (6,9000) CURSOR_OFF
DO I = 1,9
 WRITE (6,9010) LOAD_LINE_1,I,LOAD_LINE_2,BIT_MAP(I),LOAD_LINE_4
END DO
DO I = 10,94
 WRITE (6,9020) LOAD_LINE_1,I,LOAD_LINE_2,BIT_MAP(I),LOAD_LINE_4
END DO
WRITE (6,9000) INTO_G3
WRITE (6,9000) CURSOR_ON
C
9000 FORMAT (A)
9010 FORMAT (A5,I1,A4,A17,A2)
9020 FORMAT (A5,I2,A4,A17,A2)
C
RETURN
END

```



```

BIT_MAP(51) = 'FFFFFFFF/NNNN????'
BIT_MAP(52) = 'www????/NNNN????'
BIT_MAP(53) = '----????/NNNN????'
BIT_MAP(54) = 'wwwFFFF/NNNN????'
BIT_MAP(55) = '----FFFF/NNNN????'
BIT_MAP(56) = '????www/NNNN????'
BIT_MAP(57) = 'FFFwww/NNNN????'
BIT_MAP(58) = '????----/NNNN????'
BIT_MAP(59) = 'FFF----/NNNN????'
BIT_MAP(60) = 'wwwwww/NNNN????'
BIT_MAP(61) = '----www/NNNN????'
BIT_MAP(62) = 'www----/NNNN????'
BIT_MAP(63) = '-----/NNNN????'
BIT_MAP(64) = '??????/????NNNN'
BIT_MAP(65) = 'FFF??/????NNNN'
BIT_MAP(66) = '????FFF/????NNNN'
BIT_MAP(67) = 'FFFFFF/????NNNN'
BIT_MAP(68) = 'www??/????NNNN'
BIT_MAP(69) = '----??/????NNNN'
BIT_MAP(70) = 'wwwFFF/????NNNN'
BIT_MAP(71) = '----FFF/????NNNN'
BIT_MAP(72) = '????www/????NNNN'
BIT_MAP(73) = 'FFFwww/????NNNN'
BIT_MAP(74) = '????----/????NNNN'
BIT_MAP(75) = 'FFF----/????NNNN'
BIT_MAP(76) = 'wwwwww/????NNNN'
BIT_MAP(77) = '----www/????NNNN'
BIT_MAP(78) = 'www----/????NNNN'
BIT_MAP(79) = '-----/????NNNN'
BIT_MAP(80) = '??????/????NNNN'
BIT_MAP(81) = 'FFF??/????NNNN'
BIT_MAP(82) = '????FFF/NNNNNNNN'
BIT_MAP(83) = 'FFFFFF/NNNNNNNN'
BIT_MAP(84) = 'www??/????NNNN'
BIT_MAP(85) = '----??/????NNNN'
BIT_MAP(86) = 'wwwFFF/NNNNNNNN'
BIT_MAP(87) = '----FFF/NNNNNNNN'
BIT_MAP(88) = '????www/NNNNNNNN'
BIT_MAP(89) = 'FFFwww/NNNNNNNN'
BIT_MAP(90) = '????----/NNNNNNNN'
BIT_MAP(91) = 'FFF----/NNNNNNNN'
BIT_MAP(92) = 'wwwwww/NNNNNNNN'
BIT_MAP(93) = '----www/NNNNNNNN'
BIT_MAP(94) = 'www----/NNNNNNNN'

```

C

```

SHIFT_IN_1(1:1) = '+'
SHIFT_IN_1(2:2) = CHAR(15)

```

C

```

SHIFT_IN_2(1:1) = '+'
SHIFT_IN_2(2:2) = CHAR(14)

```

C

```

SHIFT_IN_3(1:1) = '+'
SHIFT_IN_3(2:2) = CHAR(27)
SHIFT_IN_3(3:3) = 'n'

```

C

```

SHIFT_IN_4(1:1) = '+'
SHIFT_IN_4(2:2) = CHAR(27)
SHIFT_IN_4(3:3) = 'o'

```

C

```

TOP_OF_PAGE(1:1) = '+'

```

```

C*****
C
C INITIALIZE VT220 FOR GRAPHICS
C
C*****
C SUBROUTINE VTINIT
C
C CHARACTER*2 SHIFT_IN_1,SHIFT_IN_2
C CHARACTER*3 SHIFT_IN_3,SHIFT_IN_4
C CHARACTER*4 TOP_OF_PAGE
C CHARACTER*6 SET_132,SET_080
C CHARACTER*17 BIT_MAP
C CHARACTER*5 LOAD_LINE_1
C CHARACTER*4 LOAD_LINE_2
C CHARACTER*2 LOAD_LINE_4
C CHARACTER*4 INTO_G3
C CHARACTER*60720 LINE_OUT
C INTEGER*4 NXCUR, NYCUR, NEXTX, NEXTY
C REAL*4 DELTAX, DELTAY
C REAL*4 XPRIOR, YPRIOR, XFOLLOW, YFOLLOW
C CHARACTER*7 CURSOR_OFF, CURSOR_ON
C
C DIMENSION BIT_MAP(94)
C
C COMMON /VTGRAPH/ SHIFT_IN_1,SHIFT_IN_2,SHIFT_IN_3,SHIFT_IN_4,
- TOP_OF_PAGE,LINE_OUT,NXCUR,NYCUR,NEXTX,NEXTY,DELTAX,DELTAY,
- SET_132,SET_080,XPRIOR,YPRIOR,XFOLLOW,YFOLLOW
C
C DO I = 1,16
C BIT_MAP(I) = '????????/????????'
C END DO
C BIT_MAP(17) = '??{CCC??/????????'
C BIT_MAP(18) = '??CCC{??/????????'
C BIT_MAP(19) = '??OgCA??/????@A??'
C BIT_MAP(20) = '??ACgO??/??A@????'
C BIT_MAP(21) = '??o?????/??BAAA??'
C BIT_MAP(22) = '?????o??/??AAAB??'
C DO I = 23,30
C BIT_MAP(I) = '????????/????????'
C END DO
C BIT_MAP(31) = '-----/NNNNNNNN'
C BIT_MAP(32) = '????????/????????'
C BIT_MAP(33) = 'FFFF????/????????'
C BIT_MAP(34) = '????FFFF/????????'
C BIT_MAP(35) = 'FFFFFFFF/????????'
C BIT_MAP(36) = 'www?????/????????'
C BIT_MAP(37) = '----????/????????'
C BIT_MAP(38) = 'wwwwFFFF/????????'
C BIT_MAP(39) = '----FFFF/????????'
C BIT_MAP(40) = '????wwww/????????'
C BIT_MAP(41) = 'FFFfwwww/????????'
C BIT_MAP(42) = '????----/????????'
C BIT_MAP(43) = 'FFFF----/????????'
C BIT_MAP(44) = 'wwwwwwwwww/????????'
C BIT_MAP(45) = '----wwww/????????'
C BIT_MAP(46) = 'wwww----/????????'
C BIT_MAP(47) = '-----/????????'
C BIT_MAP(48) = '????????/NNNN????'
C BIT_MAP(49) = 'FFFF????/NNNN????'
C BIT_MAP(50) = '????FFFF/NNNN????'

```

Figure 2

| sixel |   | 3 | 4 | 5 | 6 | 7 | 8 |
|-------|---|---|---|---|---|---|---|
| 1     | 2 |   |   |   |   |   |   |
| 0     | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1     | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1     | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1     | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1     | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1     | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

| sixel |    | 11 | 12 | 13 | 14 | 15 | 16 |
|-------|----|----|----|----|----|----|----|
| 9     | 10 |    |    |    |    |    |    |
| 1     | 0  | 0  | 0  | 0  | 0  | 1  | 0  |
| 1     | 0  | 0  | 0  | 0  | 0  | 1  | 0  |
| 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

| NUMBER OF<br>SUBFIELDS | DENSITY<br>LEVELS | CHARACTERS<br>REQUIRED |
|------------------------|-------------------|------------------------|
| 1                      | 94                | 94                     |
| 2                      | 9                 | 81                     |
| 3                      | 4                 | 64                     |
| 4                      | 3                 | 81                     |
| 5                      | 2                 | 32                     |
| 6                      | 2                 | 64                     |

used to define the character. The method of defining a character consists of a control string to introduce the soft set then up to seven positional parameters to define the character to be loaded. Next the designation of the soft set is sent. Last, the sixel values converted to ASCII characters are sent separated into the two halves by a "/" followed by the end string code that indicates the end of the loaded character.

```

<DCS> is the device control string
 <ESC>P
PARM-1 is the font buffer 0 or 1
PARM-2 is the starting character
PARM-3 selects the characters to erase
 (0-all,1-reloaded,2-allsets)
PARM-4 selects the character matrix
 size (0,2,3,4)
PARM-5 selects the device width
 attribute
PARM-6 select text or full font
"{" indicates the end of the the
 parameter list
"@ " is the name of the character set
"}OOOOO}?/B?????B?" are the sixels
<ST> is the string terminator <ESC>\

<ESC>P1;1;1{@}OOOOO}?/B?????B?;<ESC>\

```

A table of bit patterns is provided to allow quick lookup of sixel patterns. This table is provided with sixel values prior to their values being added to 111111 binary.

| SIXEL  | CHARACTER | SIXEL  | CHARACTER |
|--------|-----------|--------|-----------|
| 000000 | ?         | 100000 | `         |
| 000001 | @         | 100001 | ~         |
| 000010 | A         | 100010 | a         |
| 000011 | B         | 100011 | b         |
| 000100 | C         | 100100 | c         |
| 000101 | D         | 100101 | d         |
| 000110 | E         | 100110 | e         |
| 000111 | F         | 100111 | f         |
| 001000 | G         | 101000 | g         |
| 001001 | H         | 101001 | h         |
| 001010 | I         | 101010 | i         |
| 001011 | J         | 101011 | j         |
| 001100 | K         | 101100 | k         |
| 001101 | L         | 101101 | l         |
| 001110 | M         | 101110 | m         |
| 001111 | N         | 101111 | n         |
| 010000 | O         | 110000 | o         |
| 010001 | P         | 110001 | p         |
| 010010 | Q         | 110010 | q         |
| 010011 | R         | 110011 | r         |
| 010100 | S         | 110100 | s         |
| 010101 | T         | 110101 | t         |
| 010110 | U         | 110110 | u         |
| 010111 | V         | 110111 | v         |
| 011000 | W         | 111000 | w         |
| 011001 | X         | 111001 | x         |
| 011010 | Y         | 111010 | y         |
| 011011 | Z         | 111011 | z         |
| 011100 |           | 111100 | {         |
| 011101 | \         | 111101 |           |
| 011110 | ]         | 111110 | }         |
| 011111 | ^         | 111111 | ~         |

## VII

In order to use a character set for

graphic purposes in an application, subroutines must be provided to handle the loading of the character set, rasterization, and finally display. Included at the end of this paper is the source code used to create a library of FORTRAN subroutines.

The first routine is called VTINIT, it take no arguments and readies the tube for graphics. This routine initializes the internal bitmap and loads the character set. Please note that the terminal on which this software is being run must be set up for 132 column mode and as a VT200 with seven bit codes.

The second routine called VTDISP is used to display the plotted image on the VT220. It breaks up the image into 20 subfields which can then be called up separately.

The third routine is VTPLOT it requires input as to X position and Y position and if the line is to be drawn from the last position. X values can vary from 0 to 10.55 and Y values can vary from 0 to 8.19. This routine calls VTMAPS, the fourth routine, in order to rasterize the plot of each line.

These routines while rudimentary will provide all the plotting primitives needed to create a large plot library. In order to see plots in the shortest amount of time I used the DECUS library software package SPEEDS to convert DATATRIEVE plot files in ReGIS to vector files. I then used the vector files as input to a display routine which called the library of plotting routines. The speed of rasterization becomes much slower as the length of the vectors increases.

Figure 1

| column | 1     | 2     | 3     | 4     | 5     | 6     | 7    | 8    |
|--------|-------|-------|-------|-------|-------|-------|------|------|
| row    | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 1      |       |       |       |       |       |       |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 2      | **    |       |       |       |       | **    |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 3      | **    |       |       |       |       | **    |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 4      | **    |       |       |       |       | **    |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 5      | ** ** | ** ** | ** ** | ** ** | ** ** | ** ** |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 6      | **    |       |       |       |       | **    |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 7      | **    |       |       |       |       | **    |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 8      | **    |       |       |       |       | **    |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 9      |       |       |       |       |       |       |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |
| 10     |       |       |       |       |       |       |      |      |
|        | +--+  | +--+  | +--+  | +--+  | +--+  | +--+  | +--+ | +--+ |

```

Set G0-G3 Character Set
(- G0 B - ASCII
) - G1 < - DEC supplemental
* - G2 0 - DEC special
+ - G3 @ - User defined

```

In order for any of these sets to be used it must be shifted into the GL or GR space. To do this we use the shift in escape sequences. These sequences are: (LS0,LS1,LS2,LS3,LS1R,LS3R,SS2,SS3). Those sequence names are mnemonics for shift operations from G0 through G3 into GL and GR. "LS0" stands for, "Locking shift of G0 into GL." All of these shift sequences and their expanded escape codes are presented below.

```

LS0 = <SI>
Shift in and lock G0 in GL
LS1 = <SO>
Shift in and lock G1 in GL
LS2 = <ESC> n
Shift in and lock G2 in GL
LS3 = <ESC> o
Shift in and lock G3 in GL
LS1R = <ESC> ~
Shift in and lock G1 in GR
LS2R = <ESC> }
Shift in and lock G2 in GR
LS3R = <ESC> |
Shift in and lock G3 in GR
SS2 = <SS2> or <ESC> N
Shift in for next character
 G2 in GL
SS3 = <SS3> or <ESC> O
Shift in for next character
 G3 in GL

```

Each character set in G0 through G3 contains ninetyfour characters which can be mapped into the eight bit code tables GL and GR in positions 21 through 7E hex. In seven bit operation only the characters mapped into GL can be displayed as the eighth bit is zeroed preventing the upper half of the character tables to be accessed. Before any characters can be displayed they must be selected and mapped. Selecting a character set was covered earlier and mapping is performed by the shift commands. In the following example all four character sets will be selected and then the ascii and special graphics set will be mapped into GL and GR.

```

<ESC>(B ... Select ASCII as G0
<ESC>)O ... Select Special graphics
 as G1
<ESC>*@ ... Select loadable as G2
<ESC>+< ... Select DEC supplemental
 as G3
<SI> Map G0 to GL
<ESC>~ Map G1 to GR

```

As stated earlier, the intent to use the the terminal as a low resolution graphics device will require the use of the loadable character set. The method of designating a set as downline loadable and then loading that set is vital. In order

to load a character set it must first be designated as a soft character set. The set so designated can be loaded and then selected and mapped into GL or GR space just as one of the hard character sets. The escape sequence "<ESC>(@" loads the unregistered soft character set "@" into G1. "@" will be the character we will use to designate the down line loadable character set.

The actual creation of the down loadable character set involves the setting of sixel values to be passed to the terminal in a load character sequence. The matrix in which the character is built is eight pixels wide and ten pixels high. The ten rows are numbered from top to bottom of the matrix and the columns are numbered left to right c.f. figure 1. These rows and columns in the example are used to build a letter "H". The sixel patterns themselves are made up of each column divided into two parts. The first eight sixels consist of columns one through eight over rows one through six. The last eight sixels consist of columns one through eight over rows seven through ten. A pattern of turned on bits is generated for each sixel c.f. figure 2. These values are:

```

SIXEL 01 = 111110
SIXEL 02 = 010000
SIXEL 03 = 010000
SIXEL 04 = 010000
SIXEL 05 = 010000
SIXEL 06 = 010000
SIXEL 07 = 111110
SIXEL 08 = 000000

```

```

SIXEL 09 = 0011
SIXEL 10 = 0000
SIXEL 11 = 0000
SIXEL 12 = 0000
SIXEL 13 = 0000
SIXEL 14 = 0000
SIXEL 15 = 0011
SIXEL 16 = 0000

```

In order to prepare the sixels for use in the character loading scheme their values must have added to them 111111 binary producing the following values:

```

SIXEL 01 = 1111101 or 7D hex
SIXEL 02 = 1001111 or 4F hex
SIXEL 03 = 1001111 or 4F hex
SIXEL 04 = 1001111 or 4F hex
SIXEL 05 = 1001111 or 4F hex
SIXEL 06 = 1001111 or 4F hex
SIXEL 07 = 1111101 or 7D hex
SIXEL 08 = 0111111 or 3F hex

```

```

SIXEL 09 = 1000010 or 42 hex
SIXEL 10 = 0111111 or 3F hex
SIXEL 11 = 0111111 or 3F hex
SIXEL 12 = 0111111 or 3F hex
SIXEL 13 = 0111111 or 3F hex
SIXEL 14 = 0111111 or 3F hex
SIXEL 15 = 1000010 or 42 hex
SIXEL 16 = 0111111 or 3F hex

```

The final sixel values are the ones

can however shrink the verticle dimension by splitting the characters into two with a top and bottom half. We must pay for the refinement in aspect ratio in allowing all levels of density to be available to us in either half of the cell simultaneously. This means for each level of density in the top of a cell the level of density in the bottom of the cell can be any of the level of density. Mathmatically the levels of density raised to the power of the power of the number of cells is the number of characters required to represent all combinations. This relationship is shown in table one. Note from the table that in order to arrive with the aspect ratio of 1:1, the most number of levels, and the highest density of cells per inch we would choose to represent nine levels of density in two cells in sixtyfour cells per square inch.

#### IV

While density is a concern in some plotting applications, most applications use vectors to define an image. We will build on what we learned in the previous section to bring vectors to the VT220. As stated earlier the use of densities to present an image requires a decision on the number of cells into which to divide a character and the number of levels of density which are required for the application.

In the display of vectors the number of densities is reduced to two either a particular pixel is on or a particular pixel is off. We must then decide on the the number of cells we will break a character into, in order to have all combinations of on and off pixels. We will begin by making some decisions on acceptable cell configurations.

Since we have an eight by ten array of pixels we would like to split it into a regular pattern. A regular pattern will have the same number of sub cells in each row and the same number of sub cells in each column. The only combination that works is to split the character into six sub cells. Each sub cell should be approximately the same size. We do this by dividing the eighty pixels into two columns four pixels wide and three rows which are three, three, and four pixels wide. The aspect ratio of the sub cells varies between 2:3 for the first two rows and 1:2 for the last two rows. If we place the terminal in 132 column mode the aspect ratios become 1:2 and 3:8. The sub cells are about nine times as large as the pixels on a VT240 and therefore the resolution in both the horizontal and verticle directions is about one third that of the VT240. This is acceptable for a graphics display. Now we will design layout of the characters to permit us to handle rasterization of an image.

#### V

We will use the layout of the sub cells in a regular pattern to help us decide which characters are assigned which

sub cell pattern. If we look at the six sub cells as bits then we could use the patterns of six bits to represent which cells are turned on. For example a bit string of six zeros would indicate that all subfields are off.

```
000000 all sub cells off.
000001 upper left sub cell on
000010 upper right sub cell on
000100 middle left sub cell on
001000 middle right sub cell on.
010000 lower left sub cell on.
100000 lower right sub cell on.
```

If we append bits 01 to the front of these strings the bit patterns can then be related back to the ASCII character set with the following results.

```
01000000 = '@'
01000001 = 'A'
01000010 = 'B'
01000100 = 'D'
01001000 = 'H'
01010000 = 'P'
01100000 = ''
```

We can then designate any combination of turned on bits by anding the bit patterns of the selected bits together. The result is some character between '@' and <DEL>. If we substitute '?' for the delete character we have used sixtyfour of nintyfour available characters to define all the necessary bit patterns in our two by three array. Now we will cover the steps necessary to load a character set into the VT220.

#### VI

The first step in preparing a VT220 for a downline loaded character set is to designate the downline loadable character set as one of the four available character sets G0 through G3 that the VT220 can load into its GL or GR space. The GL space is the left half of the displayable character set available to the terminal and it corresponds to values 0 through 127 received by the terminal. The GR space is the right half of the displayable character set and it corresponds to values 128 through 255 as received by the terminal. Character sets from G0, G1, G2, or G3 are shifted into the GL or GR area to be used by the terminal. In seven bit operation only the GL area has any meaning as the GR space is unadressable due to its corresponding values above 127.

In the example that follows we see the escape sequences sent to the terminal to designate various character sets into the G0 through G3 space. Under each character is an explanation of its function

|          |       |              |
|----------|-------|--------------|
| <ESC>    | (     | B            |
| <ESC>    | )     | O            |
| <ESC>    | *     | <            |
| <ESC>    | +     | @            |
| Escape   | Set   | Character    |
| to start | G0-G3 | Set selected |
| sequence |       |              |

LOW COST TERMINAL OPTIONS  
FOR DEC EQUIPMENT USERS

Charles S. Janik  
Bell Helicopter TEXTRON  
Fort Worth, Texas

ABSTRACT

The VT200 series terminals represent a unique opportunity for providing graphic capabilities in otherwise non-graphic environment.

I

The single most difficult problem in bringing graphics applications to life is the availability of a suitable development devices on which to demonstrate ideas regarding the final product. The cost of producing graphics on borrowed equipment is related to the time a programmer must spend in learning the device protocols and kinks. If the device used to produce graphics is simple enough ; and the routines used to drive the device are basic enough then, the task of producing a graphics application becomes the act of designing the application rather than learning the equipment.

II

The DEC VT220 series of terminals provides a ready answer to the simulation of most graphic devices with its user loadable character sets. The ability to downline load a character set to a terminal allows a user to create up to ninetyfour different characters beyond the usual available on the device. Use of patterns to represent parts of a graphic image is already available in the linedrawing character set. The purpose of using the loadable character set is to provide more flexibility in the generation of images.

The secret to an effective graphic display is resolution, by which we mean the density of pixels used to define an image. The DEC VT240/241 terminals have a horizontal resolution of onehundred dots per inch and a verticle resolution of fifty dots per inch. These devices are considered to be low resolution graphics terminals, and yet they still produce acceptable graphics for the average application. The question confronting any implementer of graphics is therefore, "How fine does the resolution need to be before the image is identifiable and therefore usable."

With the use of the downline loadable character set, a resolution one third as good as the low resolution of the VT240 series terminals can be obtained. In many

first tries at an implementation this level of resolution is acceptable. First, however, let us look at a graphic application that relies on the verticle and horizontal relationship of a character to decide the number and types of characters in the user defined character set.

III

One way to present a graphic image on a text device is to use the density of the characters to represent the density of the image for a particular cell. This is seen most clearly in the newspaper photograph effect obtained on printers with overstriking to produce density pictures. The use of characters to define density is not limited to printers and several packages are available which produce density plots on the standard display tube through the use of characters representing various levels of density. In the most common mode with two levels of density (either on or off) an "@" character is used to represent the presence of data and the " " the absence. As other levels of density are added between these two the need for a key to tell the difference becomes apparent. With the VT220's loadable character set true densities can be created for each desired level to the maximum of the number of characters available. Each character has available to it eight columns by ten rows of pixels which can be turned on or off. It would seem likely that to cover the density range of characters we need only use eighty of the ninetyfour characters available. Even if we do this and we present a usable density plot the ratio between the horizontal and verticle lengths of the character cell is 1:2. The cell that we are representing is usually not a rectangle but a square and the aspect ratio on the cells is 1:1. To represent the cell accurately on the tube requires that we either widen the horizontal direction or shrink the verticle direction. If we widen the horizontal direction our problem becomes one of unacceptable cell representation with sixteen cells per square inch. We

```

onhook(chn_ptr)

CHANNEL_PTR chn_ptr;
{
 UCOUNT j;
 Enter(OHHOOK);
 Variable(chn_ptr,%4x);

 (2) Comment("Set the line status to on hook.\n",NULL);
 chn_ptr->line_state = ON_HOOK;

 (2) Comment("Fill the entire memory with silence\n",NULL);
 for (j = 0; j < MAX_WINDOWS; j++)
 {
 (2) Comment("Fill window #%d with silence.\n",j);
 (3) Select_window(chn_ptr->chn_num,j);
 (3) Fill(chn_ptr->wndw_base,0x0000,WINDOW_SIZE,SILENCE);
 }
 (2) Comment("Cmd channel on hook, disable cpm\n",NULL);
 (3) Wr_chn_cmd(chn_ptr,(ON_HOOK_CMD + CPM_DISABLE));

 (1) Exit(OHHOOK);
 return;
}

/* DEC/CMS REPLACEMENT HISTORY, Element ONHOOK.C */
/* *1 20-MAY-1985 11:16:18 SCHORNAK "PPR 7" */
/* DEC/CMS REPLACEMENT HISTORY, Element ONHOOK.C */

```

Figure 7  
Code Fragment Written to Run in Simulation

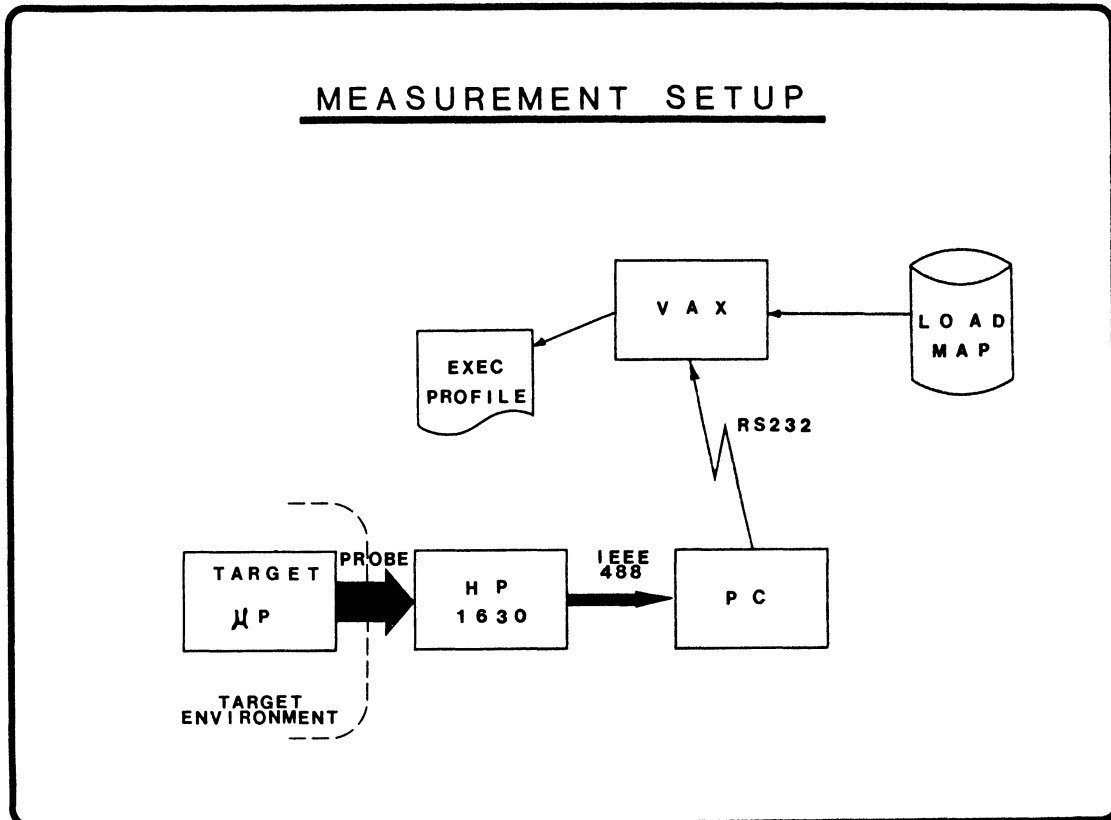


Figure 8  
Measurement Setup for Project "A"



As I mentioned before, the "C" coding is done by filling out a blank form. In this way, the header comments then become machine readable, if we so desire. The Entry and Exit macros already built in and a number of other standard coding conventions are supported. The documentation form is used the same way. We bring in a standard format for documentation and fill in the blanks. We use runoff for all documentation.

Batchable Tools - All of the tools I've talked to you about, except for the PPS Create, run in batch. The commands are written in DCL and use DTR. As a result they are terribly slow when run interactively. Therefore we tend to run them in batch. We are able to get a lot more interactive use of the VAX by using the batch queue to handle these and most other length non-interactive tasks in batch.

Coding Standard And Naming Conventions - We use coding standard and naming conventions. Enforcement is not yet automatic. I would like to have the CMS CREATE ELEMENT do a coding standards check and reject incorrectly formatted modules. However, the code reviewer we currently employ does a pretty good job of enforcing the standard.

Streamlined Documentation - We have streamlined software documentation. We don't have program logic manuals. We make three documents, a design document, an operators manual or a programmers manual, depending upon the program and a functional specification, that is used for an RFP. The rest of the documentation is in the code. I haven't met a maintenance programmer yet that does anything but go right to the code the minute he thinks he has a problem. So, any other documentation is probably not worthwhile. We concentrate on documentation of the code in the code, and the glue that holds modules together is in the design document. We enforce the maintenance of the design document. Our shop does not have government or military requirements, so we can get away this scheme.

## TWO CASES

I'd like to present the results from two projects that have used these methods. Please bear in mind, that we've been using these methods now for about 2 1/2 year, and the first project predates the current company.

### Case "A"

Table 1 summarizes the first project.

TARGET: 68000, STANDALONE

TYPE: POLLED COMMUNICATIONS PROCESSOR

SIZE: 30,000 BYTES C, 196 BYTES ASSEMBLY  
186 BYTES VENDOR LIBRARY

EFFORT: 15 CALENDER MONTHS  
3.5 MEN, 36 MAN-MONTHS

TOOLS: SA, PPS, LSE, SIMULATION,  
TUNING BY MEASUREMENT

### METRICS:

- 289 PPRS, ~ 20 FOUND IN TARGET
- MOST TARGET DEBUG FOR CUSTOM HARDWARE.
- SPEC FOR 10,000 TRANSACTIONS/SEC.
  - BEFORE TUNING 5000/SEC
  - 9300/SEC AFTER 1 MANWEEK TUNING
  - NO ASSEMBLY USED FOR PERFORMANCE

Table 1  
Summary of Case A

The target was a Motorola 68000 without operating system. It is a data communication processor that ran inside a data communications node. The program was 30,000 bytes of "C" with only 196 bytes of Assembly code. We don't use any vendor-supplied IO library. It took about three man-years to develop.

We used the structured analysis, the problem reporting system, language-sensitive editor. We used validation in simulation. We also used discrete event simulation to model the connection protocol. We were able to model data networks of thousands of nodes that would never be built and see how the protocol would do when pushed to the edge. Those networks of that size will not be built for a number of years and I hate being called back to do maintenance.

Here are some of the metrics from that project. found. We had about 289 problem reports. We found about 20 of those in the target environment. All the rest were results of simulation runs and the code review. Most were fixed before we had operational hardware. Most of the problems we found in the target environment dealt with errors in the custom hardware or the network environment. The documentation on the rest of the network was not sufficient to write a complete specification, so we found a some problems when we got into the network. We had built a simulator for the rest of the network to test in simulation. However that simulator was based on the same out-of-date specifications, and therefore did not help with that sort of problem.

We met our performance goals through measurement. The product was specified to run 10,000 characters a second. Whoever thought up 10,000 characters a second had

no basis for that--just kind of grabbed it out of the air and said, "Gee, I think we can do that." You know, here I am, a poor programmer, stuck trying to make 10,000 characters a second. So, I wrote it for maintainability and without any tuning of the product hit 5,000 characters a second.

To give you an idea of the kinds of things it was doing, it was handling multiple data streams of up to 45 possible transaction types and piping them on to the Ethernet, encoding them in three layers of protocol and decoding them on the other end. So it was doing a lot of work.

Figure 8 displays the measurement setup used on this project to tune performance. We set the target microprocessor up in the target environment with an HP1630 logic analyzer connected to it. The logic analyzer was controlled by a PC over an IEEE 488 bus. The PC told the HP1630 to collect 1,000 instruction fetches and send the samples back to the PC. The PC tabulated them and ordered the HP1630 to do it again. This process was repeated until 2,000,000 samples had been collected. It took about two hours to do that, but everybody went to lunch. When we came back it was done.

The totaled samples were uplinked to the VAX. On the VAX we wrote a program to bounce the samples against the load map. What the process did was to count how many times each location of memory was fetched for an instruction. The end result was an execution profile which told us the percentage of time each separate subroutine actually executed in the real environment. We could then sort that with a standard Vax sort utility, and we had a sorted list of the ten top pigs on the system.

These ten modules were then examined in detail and algorithms redesigned to improve efficiency. However the single most productive technique was to change subroutines to inline "C" macros. We reached 9300 characters a second with only one manweek of code rework. No Assembler code was used in the operation, only modifications to the "C" algorithms.

We almost doubled the performance. We could have made 10,000 characters a second. We did the measurements for another set of changes. The changes this time would have had to be a little more drastic to the protocols and would have adversely affected maintainability. We decided the 700 characters weren't worth it, so we just stopped.

#### Case "B"

Table 2 summarizes the results for the second project.

TARGET: IBM PC WITH CUSTOM BOARD

TYPE: INTERRUPT DRIVEN, SPOOLER

SIZE: 4034 BYTES C, 326 BYTES ASSEMBLY  
6,800 BYTES VENDOR LIBRARY

EFFORT: 2 CALENDAR MONTHS,  
2.5 MAN MONTHS

TOOLS: SA, LSE, SIMULATION, PPS

#### METRICS:

- NO APPLICATION ERRORS FOUND IN TARGET ENVIRONMENT
- TEN ERRORS IN ASSEMBLY INTERFACE CODE
- PERFORMANCE BUDGET MET WITHOUT TUNING

Table 2  
Summary of Case B

This is the product that we produced for Comdex in May, 1985. You can see it's a much smaller problem, with only 4000 bytes of code. It had more assembly, because it fielded three interrupts and it took a lot of bytes of assembly to get the interrupts taken care of. It took 2 1/2 man months in two calendar months. That doesn't include the hardware, that's just the software component of it.

We used all of the techniques that I've mentioned before, particularly language-sensitive editor, testing in simulation and the problem reporting system. Of these techniques, testing in simulation provided a very big surprise. The code was written to execute in simulation under the control of a very simplistic test driver. The code was not moved to the PC until all problems were resolved in the simulation.

Once the code was moved to the target environment, it just worked. No changes were made in the simulation tested code, NONE, ZERO. All of the problems were involved in getting the assembly language code to work. We met our performance budget without any difficulty.

#### FUTURE PLANS

What do we plan to do with our environment as we go along? If we keep selling product, or start selling product, one thing we're going to do is use this for everything we do. This paper describes a methodology for developing products for a microprocessor target operating environment. The VAX, though not a microprocessor, is certainly a target environment. In fact, the day after I wrote the design specifications for the project system the system manager came in and said, "Gee, could you change these

three things so I could use it for everything? "I've got ten projects I'd like to build." Sure enough, the minute Whitesmith's compiler came in, it was placed in a standard project structure. All of their source code went into the CMS library, their objects went into an object file, their EXEs went into the public, and a PPR system was setup for it.

We want to integrate MMS. In the six months since DECUS last, we have developed all of the tools described here and our first product. The tools are designed for MMS, we just didn't have time to put MMS in.

As soon as somebody finds a way to buy DEC Test Manager we'll do so. One of the really big pluses for doing testing in simulation is that it will work with the Test Manager.

We want a full language-sensitive editor. I've made my comment on buying a language-sensitive editor over making one. We tried making one, and we're a about one-tenth of where we need to be. We'll wait for DEC to sell us one.

We're looking for structured analysis tools. There has got to be a better way than spending \$25,000 and buying an graphics terminal and a ton of software. I am still looking for a better way.

Finally, the whole concept will evolve. That is why there is a Problem Reporting System setup for the project system itself. So we can collect and track problems and suggestions for the environment.

**LARGE SYSTEM SIG**



# TOPS-20 Q & A

Betsy Ramsey  
American Mathematical Society  
Providence, Rhode Island

## Abstract

Large Systems software engineers answered questions from members of the audience on the TOPS-20 operating system and related utilities.

## Introduction

As an experiment, this session was preceded by a "novice" section in which users with less technical questions were encouraged to come forward. No users took immediate advantage of this offer, so the "Novice Q&A" session will probably be omitted in Anaheim.

The questions and answers presented at "TOPS-20 Novice Q&A" and "TOPS-20 Q&A" are paraphrased in this paper.

Present from Digital Large Systems Software Engineering were David Braithwaite, Kevin Paetzold, Dave Lomatire, Marty Palmieri and Mark Pratt. Other engineers were present via a teleconference link to Marlboro.

## Questions & Answers

**Q.** We have experienced a problem where the load suddenly spikes from 3.0 to 15.0, lines freeze and the ACJ times out.

**A.** The answer is not obvious from this data. It sounds like it could be an open line problem for a bank of terminal lines. In that case, we would expect it to effect the same lines each time. You can check WATCH data for high BOND figures. If that isn't the case, be sure that the ACJ itself isn't causing the problem.

**Q.** If all that fails, where else can I look?

**A.** Crash the system and get a dump while the lines are hung.

**Q.** What is the best way to do that?

**A.** At PARSER level on the console, type JUMP 71. That will simulate a KPALVH bught.

**Q.** In the new version of DUMPER that was distributed with V6.0, RESTORE/TAPE-INFORMATION no longer seems to be the default. Will this be fixed?

**A.** Yes, it has been QAR'd, and the fix is a one-line patch.

**Q.** Will it be possible for users to add their own network support to MS? E.g., will MS sources be distributed?

**A.** We're not sure what the policy will be on this. We will get back to you. An attempt was made to write the MS network support in a modular fashion, so hopefully it will be easy for users to add their own code.

**Q.** Is it possible to convert a DECsystem-1091 with 1.25 MW memory to TOPS-20 without a big performance loss? The maximum user load is 60 to 70 users with 20 operator jobs. About

half of those are 1022 users, and the other half are general student users.

**A.** You should talk to Software House about differences in 1022 performance. File structure and memory management in TOPS-10 and TOPS-20 are significantly different. It could be that if your files are large, you will get better performance under TOPS-20. For the most part, the things that TOPS-10 does well (like BASIC) will run faster there than under TOPS-20. TOPS-20 performance is very dependent on the amount of memory available, so look at those TOPS-10 statistics that will relate to things like balance set size, swapping rate, working set size.

**Q.** We have experienced a problem with a DUMPER full save where every other tape in the set was not written. Alternate tape drives (TU45, TU78) were used.

**A.** This problem hasn't been seen on either type of drive previously. It probably isn't DUMPER. If DUMPER is spinning the tape, it's writing something.

**Q.** (another user) Have seen this problem on a TU78. Symptom is that the tape seems to be moving too fast.

**Q.** (still another user) Have seen this problem under TOPS-10 as well. It usually culminates in a PULSAR crash.

**A.** We will investigate this through our hardware people.

**Q.** I have a possible answer to the user with the load spike problem. LPTSPL can cause load spikes when it tries to print a file with unprintable characters or when it is spooling to a fast output device (such as magtape).

**Q.** Does LPTSPL set itself up to be a system job?

**A.** Yes, but this can be changed when you build Galaxy. It is a GALGEN parameter.

**Q.** DUMPER restores of a structure never go like they're supposed to. Files-only directories suddenly become not files-only, retention counts aren't preserved, accounts are messed up. DEC should try the process themselves to see if their documented procedure actually works.

**A.** There is a known problem with DUMPER where if directory groups are deleted from a superior directory after a save has been done, DUMPER will not be able to create the subdirectory on a restore because the group number is missing from the superior. The DUMPER developers are working on this.

**Q.** It isn't a DUMPER problem, it's a CRDIR problem. There are about half a dozen CRDIR/GTJFN/etc. bugs that combine to create these DUMPER restore problems.

**A.** We're working on it.

**Q.** We use modem control with a statistical MUX on a DEC-2020 so that links will be broken if either the 2020 or the MUX goes down. The problem is that if the 2020 never sees ring indicate, it won't bring up DTR, so it can't drop it to break the link.

**A.** Don't know what to say.

**Q.** (another user) The answer is that there are two different versions of Bell 103 modem operation. With the newer version (which I know is used with VAX running VMS), you don't need ring indicate to raise DTR. All that is needed is carrier detect.

**Q.** This is a CFS question. I am running the MSCP server to share my RPxx disks. System A crashes, then bughlts again claiming that system B has its PS: locked for exclusive access. How can this be?

**A.** System B can mount system A's PS: for exclusive access while system A is down because system B has no one to do CFS voting with. This will prevent system A from restarting.

**Q.** When V6.0 was distributed to us, the size of BOOT and MTBOOT looked bad on the floppies, and sure enough, these files did not contain RP20 support. What happened?

**A.** You caught us in mid-process. Up through V5.1, RP20 support was treated as an unbundled product. RP20 support was provided on a separate tape. The monitor includes RP20 support, but a different bootstrap is needed to load the RP20 microcode. This is what was supplied on the separate tape. Be sure to follow the installation guide, which has specific instructions for RP20 users.

**Q.** We've noticed that Control-C no longer does an XON. Why?

**A.** That's the way it was always intended to work, so we fixed it. It won't be changed back.

**Q.** Can I use the DUMPER CREATE command between structures?

**A.** No, you should restore files to the same structure they were saved from. If you are worried about the create process, run DLUSER as part of your save process.

**Q.** This is a TCP/IP question. I've encountered a problem where Control-O close to the end of terminal output causes TELNET to hang.

**A.** There are two known TELNET bugs: windows get small at times, and there are problem with sink stuffing. Don't have the fixes out yet.

**A.** (another user) The system does flush the output buffers on Control-O. The code doesn't work.

**Q.** Sometimes hardwired VT102s go nuts. Smooth scroll was on.

**A.** Turn off smooth scroll.

**A.** (another user) Don't let anyone use smooth scroll, especially with EMACS. It doesn't work.

**Q.** We use Control-C trapping to prevent users from aborting LOGIN.CMD prematurely. Will we be able to do this under V6.1?

**A.** V6.1 will allow system-wide startup .CMD files which users can't edit (since they are in SYSTEM:). You could place your Control-C trap there.

**Q.** Problems with DUMPER on TU72s. We don't clean tapes frequently and we use an old version of DUMPER.

**A.** Not enough information, but it's probably not DUMPER's fault.

**Q.** We've experienced problems with Interchange mode in DUMPER. DUMPER is patched through Autopatch Tape 8.

**A.** There have been problems for some time with Interchange mode. Please submit SPRs on any problems encountered. The new version of DUMPER now in field test should fix some of the problems.

**Q.** When we went to restore three versions of an .EXE file, the first two of which were identical, only two were restored. Why?

**A.** When DUMPER supersedes older versions (which is its default action), the older file on disk will be deleted.

**Q.** We have a situation where, ever since Autopatch Tape 9 of V5.4, our accounting data has shown mysterious OPERATOR jobs with account OPERATOR (which is invalid at our site) logged out while detached.

**A.** You could be seeing an autologout of not-logged-in jobs. Note that with Autopatch Tape 9 patches installed, not-logged-in jobs are no longer enabled.

**Q.** We have experienced hung jobs while using KERMIT.

**A.** Hung jobs occur fairly frequently under TOPS-20. Use SYS-DPY to look at the jobs scheduler state to determine the problem. Sometimes UNATTACHing the job will clear up the problem. Often hung jobs will be accompanied by FLKTIM bugchks. If this happens frequently, you can crash the system, get a dump, and SPR the problem.

**Q.** Will we be able to add custom terminal types to EDT-20?

**A.** It's not hard to add new terminal types, but since EDT is written in Bliss, you would need a Bliss compiler to do it. We will look into other solutions to this problem.

**Q.** We would like to dual-port a TU78 between a KL and a VAX. Can we?

**A.** Yes, you can do it the same way you would between two KLs. It's better to use the drive in either A or B mode rather than A/B mode, however, or problems will occur when a system that has just rebooted tries to rewind the tape.

**Q.** We would like a special restore mode in DUMPER in which files are always superseded.

**A.** We disagree with that. It is better to start with the most recent incremental save and work your way back to the full save using SUPERSEDE OLDER.

**Q.** The problem was that DLUSER made MAIL.TXT files, so DUMPER didn't restore them.

**A.** (another user) Have to say \*.\* on RESTORE command.

**A.** DUMPER almost always does the correct thing by default unless you tell it otherwise. Be careful when using the RESTORE command. It has different wildcard defaults depending on whether you did recognition input while issuing the command.

**Q.** When I SAVE an image and GET it back, pages that contained all zeroes are not there.

**A.** This is a feature.

**Q.** And it's fine under most circumstances. But I want to trap illegal memory references, and this gets in my way. Why aren't these all-zero pages saved?

**A.** Because if they were, they would all be private pages which get swapped out, and the swapping space would fill up quickly. The illegal memory reference trap was designed only for PA1050's use.

**Q.** I would like an option to the START command to create these pages.

**A.** The trap was implemented only for PA1050. It was not intended to be used by general user programs.

**A.** (another user) You can make your interrupt handler check for the page address and allow it.

**A.** (still another user) FAIL used to have the same problem, and it was fixed in the manner just described.

**Q.** Batch jobs submitted by certain users can't run without being NEXT'd from OPR.

**A.** SPR it.

**Q.** I need to be able to raise DTR at will.

**A.** There is some support in the front end to do this on command from the KL. There is an unsupported patch to MTOPR to add this functionality.

**Q.** We are running V4.0 on a DEC-2020 with V4.1 SM files. Some Cobol executables don't run.

**A.** Might be mismatched LIBOL or COBLIB.

**Q.** I've noticed problems executing VT100 escape sequences on a VT52.

**A.** Yes, \$[ does terrible things to a VT52.

**Q.** We use SPSS in batch. When one of our users put in a Control-C to return to monitor level, the job looped.

**A.** It's probably that BATCON is getting stuck trying to force the job to monitor level (via the SETJB jsys).

**Q.** It went away when we took out the Control-C.

**Q.** What about SPEAR under V6.1?

**A.** The new SPEAR (which has been in field test forever) will be released with V6.1 of TOPS-20. The ANALYZE module will no longer exist.

**Q.** We have a problem with INFO where sometimes the time-stamp gets put in the place of the PID.

**A.** Please SPR this problem. It is interesting.

**Q.** Is it possible to disable line editing in the TEXTI jsys?

**A.** Yes, in V6.1.

**Q.** We've had strange problems on our RP20s that go away when we push the Attention button. For example, people using that drive would suddenly have their jobs hang. It turned out that BATCON was attempting to write a .LOG file and started sucking up CPU. It stopped when we pushed the Attention button.

**A.** There have been lots of fixes made to the RP20 code in V6.1. Jobs hang because that's the way TOPS-20 works. In particular, when a job queues an IORB, it will wait until that IORB is satisfied—it can't abort. This will cause problems with shared HSC disks. We are working on ways to fix this. Perhaps some sort of "DISMOUNT/DAMMIT" command that will abort all the waiting IORBs. If you do that, though, then you run into problems when the system gets an interrupt and there's no IORB waiting. Anyway, we're working on it.

**Q.** I'm running a V5.1 system and we get lots of TTYSTP bug-infs.

**A.** These are handled completely differently under V6.1.

**Q.** The terminal in the buginf message really does have a job on it.

**A.** That shouldn't happen. Also, you should never get more than one TTYSTP per line.

**Q.** This is a performance question. I have a DEC-2060 with 1.25 MW, 95 jobs and a load of 35. We are installing the MCA25. Would it be better for us to use LAT terminal servers in place of hardwired terminals?

**A.** We can talk to the NIA-20 faster than we can to the front end. The NI incurs less CPU overhead because it is on the C bus (it is a DMA device). DTE devices incur more overhead. Your bottleneck is probably either in the DTE or RSX itself.

**Q.** We have a directory containing 2000 files all with the same file name but different file types. It takes us three to four minutes to do a VDIRECTORY with the NO FILES subcommand.

**A.** The monitor stores the first five characters of a filename in the directory symbol table as a shortcut to finding the FDB. Since all the file names are the same, you are not taking advantage of this feature.

**Q.** When I install CFS, I want to change the name of PS: without doing a rebuild. Can't I just change the home block?

**A.** Yes, that's no problem. Just be sure to reboot the system afterwards.

**Q.** The new autobaud detect code in RSX-20F does it too readily.



**A. RSX-20F will autobaud only on Control-C or carriage return.**

**Q. No, it autobauds if the character looks like a Control-C or carriage returns at one of the baud rates. The solution would be to wait for two successive character matches.**

---

**Q. Will the LAT protocol be available?**

**A. This is the wrong session to ask that question—you should have gone to the LAT protocol session. The protocol is not public domain at present.**

---

**Q. Ever since we applied Autopatch Tape 8 to TOPS-20 V5.1, there have been two session entries made in the accounting .BIN file at logout. The second entry is only a few seconds after the first. Why are these being made?**

**A. Accounting was always meant to work that way. It is related to the way the EXEC handles accounting at login time, which results in two entries at login and logout.**

---

Jack Stevens  
The Gillette Company  
Boston, Massachusetts

ABSTRACT

A member of DEC's LSM Technical Support group gave an overview of the program development environment and features of VMS, tailored to users of TOPS-10 and TOPS-20.

AGENDA

- A. Presentation  
Kathy Rosenbluh (Digital Equipment Corporation)
- B. Questions and Answers

Kathy Rosenbluh, of DEC's Large Systems Marketing Technical Support group, described the basic tools available to VMS program developers and the program development cycle as it applies to VMS. The slides used in the talk are included below.

The basic schedulable entity on VMS is a process. One is created for you when you log in. Images run in them. Each process has one 32-bit address space which is divided into four parts, only one of which concerns the average applications programmer.

A process can execute images ("run programs"), execute DCL commands (the regular VMS user interface commands), execute procedures (equivalent to TOPS command, MIC, or batch files), start detached processes; or spawn subprocesses, which can run concurrently or return control to the parent process after completion.

Program development starts with source files, which, if they are in any of the VMS native-mode languages (Fortran, Cobol, Basic, PL/1, RPG, Pascal, Macro, Bliss-32) can call one another, as well as system services, via a standard calling sequence. The standard calling sequence defines three ways of passing arguments (by value, reference, or descriptor), and all these languages support all the methods.

There are no generic compile or save commands. Compilers and the linker are invoked explicitly.

The /NOOPTIMIZE switch should be included in compilations until one is confident that a program runs correctly.

There is no explicit library program, but there is a LIBRARY command which runs a program. The /REPLACE switch is used to add entries to libraries (there is no /ADD switch).

Like most linkers, the VMS linker has a large assortment of switches, most of which are not normally used. A user can set up a logical definition to cause the linker to search an arbitrary library as a default.

Shareable images (not to be confused with shared images) are nonexecutable. They are brought into memory when the calling image is executed.

Programs or parts of programs which require them can be given privileges, so that users don't have to be given extra privileges. This is done with the INSTALL command.

The INSTALL command is run at system startup. It tells the system to keep headers of certain images in memory, to reduce the overhead of starting them up.

Shared images (as opposed to Shareable images) keep only one copy in memory for all users.

The debugger can be invoked at run time, but it is not as useful that way. VMS version 4 provides a window mode for the debugger, which supports up to 12 simultaneous windows. The source code display shows the source code as it is being executed. The debugger also allows the definition of VT100 keypad keys to perform sequences of commands. Windows can be set up to display lists of program modules and to show calls to user subroutines and user or system services.

Besides using the debugger, other commands can check up on programs. SHOW PROCESS /CONTINUOUS shows how much I/O is being performed, what image is being run, how much of various quotas are being used, etc. CTRL/T does not show as much information as TOPS systems show. The set program name system service can be useful here, in that a program can change the name it displays as it moves from stage to stage. If necessary, one can run a program in a subprocess, set to dump on error. That dump can be examined with the System Dump Analyzer.

Of the numerous editors available for the VAX, at least two versions of EMACS can be obtained from third parties. TPU is DEC's newest editor, a programmable one that will be distributed (with VMS version 4.2) with two user interfaces: one that imitates EDT and one that is slightly different. TPU can be programmed to present

whatever interface one wishes, with not much more effort, say, than it takes to create EMACS libraries.

System routines can be called directly by many languages, without requiring assembler subroutines. The three different kinds of system routines vary in the kinds of arguments they can accept, the default values they provide, and, most significantly, their scope. The system services provide information about, and affect, wider system areas; the run-time library routines tend to provide information about, and affect, files, other images, etc. The utility routines involve things like the callable editor and the print symbiont.

Run-time library routines provide, among other services, easy ways to use RMS in one's program. CLI (command language interface) parsing routines provide services similar to the TOPS-20 COMMAND JSYS. File conversion services are equivalent to the ANALYZE/RMS command. The EDT editor can be called from a program.

Many of the methods for inter-process communication work only by user definition among cooperating processes. Global sections are, essentially, sharing memory. Since lock management names don't have to refer to actual resources, they can be used to pass data between processes. Kernel mode AST's can be used by one process (with sufficient privileges) to force another to execute routines which one has defined.

AST's (asynchronous system traps) are equivalent to programmable software interrupts.

File system organizations and access modes are standard across all languages.

\* \* \* \*

## A P R O C E S S

Process = Context + Executable Image

- o Has one 32-bit physical address space
- o Has 4 30-bit virtual address spaces
- o Contains current image in P0
- o Contains stacks, I/O database, quota and privilege information, logical name tables, PSL, etc. in P1
- o contains system space, shared by all processes in S0

## A P R O C E S S C A N

- o Execute images
- o Execute DCL commands
- o Execute procedures
- o Spawn another process

## P R O G R A M D E V E L O P M E N T

- o Native-mode VMS Languages:  
FORTRAN, COBOL, BASIC, PL/1,  
RPG, Pascal, MACRO, BLISS-32  
Can all call one another
- o No explicit compile command or save command  
\$ FORTRAN FILE1.FOR, FILE2.FOR, . . .  
\$ COBOL FILE1.COB, FILE2.COB, . . .  
\$ LINK FILE1,FILE2,FILE3,FILE4 . . .

## C O M P I L E R S

- o Create object modules
- o Source code can have multiple program units  
-AND- object file can have multiple object modules
- o /DEBUG adds symbols, entry points, line number info
- o /CHECK for out-of-bound subscripts, arithmetic overflows and underflows
- o /NOOPTIMIZE
- o /LIST (can use in conjunction with /NOOBJECT) line numbers, variable datatypes and addresses

## O B J E C T L I B R A R I E S

- o created with \$ LIBRARY/CREATE libname
- o add entries with  
\$ LIBRARY/REPLACE libname objectmodule  
(you can delete the object file after placing the object module in a library)
- o entries can be extracted and deleted  
(Extracting an entry creates an object file from it. If you delete a lot of modules, do \$ LIBRARY/COMPRESS to reclaim file space)

o Libraries contain compiled object modules

## SHAREABLE IMAGES

o -and- other objects (not compiled):

- command language descriptions
- error descriptions
- symbol definitions
- system-defined procedures

## LINKER

o Invoked with the \$ LINK . . . command

o Gets object modules from object files and/or library files

o Creates executable and shareable images

o /DEBUG appends symbol, line # info to image

- causes image to run under debugger by default
- override default at runtime with \$ RUN/NODEBUG (can still enter debugger after CTRL/Y)

o /TRACEBACK dumps image/process state after error

o /MAP/FULL provides virtual memory map, global symbols, cross reference, module synopses, etc.

o Shareable images are nonexecutable

- saves disk space
- executable images link to them without physically including them
- linkage is set up when image is activated

- use transfer vector macro to save having to relink executing image when shareable image changes

- use CLUSTER in options file to bind macro and image

o Use /GSMATCH in options file to indicate whether executable image must relink when shareable changes

o Allow global symbol to be referenced outside by including /UNIVERSAL in options file (must relink executables if shareable is changed)

o Create shareable images with \$ LIBRARY /CREATE /SHAREABLE imagename

- default file type is .OLB

o Advantages: save disk space, maintainability

Disadvantage: image execution is slower

## GLOBAL SYMBOLS, LIBRARIES

o Linker searches

- explicitly named modules and libraries
- system default libraries
- user default libraries

o User default libraries: (where MYLIB is an object library)  
\$ DEFINE LNK\$LIBRARY dev:[dir]MYLIB

## PRIVILEGED PROGRAMS

o Some programs execute privileged system services or obtain access and resources through enabled privileges.

o Instead of giving all users the privileges install image with privileges

\$ INSTALL

INSTALL| CREATE dev:[dir]image /PRIV=priv

## SHARED IMAGES

o Shared image: only one copy in memory

o Use Install utility to make image shared

## DEBUGGER

o Can be invoked at compile, link, or execution time

o If invoked only after execution, won't have access to symbol table

- o In window mode, shows 3 default displays: Debugger output, Source code, Register contents
- o Can save a snapshot of a display
- o Can define other displays
- o Has keypad mode
- o Has HELP and SPAWN commands

BREAKPOINTS, ETC.

- o Can set breakpoints at routine start, at exception break, at any location, on a type of instruction . . .
- o Can set tracepoints at same places, to just display execution of interesting instruction and continue
- o Can activate breakpoint /AFTER n iterations
- o Can conditionally execute list of commands at break
- o Symbols can be used with patch utility

OTHER INFORMATION

- o SHOW MODULES
- o SHOW REGISTERS
- o SHOW CALLS
- o SYMBOLS

- SET SCOPE to define program region to use in interpreting symbols
- make symbol uniquely identifiable with pathname prefix

module routine block section line symbol

CHECKING UP ON PROGRAMS

- o SHOW PROCESS /CONTINUOUS /ID=xxx
- o CTRL/T
- o RUN /PROCESS image /DUMP
  - examine dump with SDA
- o RUN /PROCESS image /ERROR
- o relink image with /DEBUG and/or /TRACEBACK

- o SOS -- line oriented editor, unsupported
- o EDT -- line mode
  - screen mode with keypad commands
  - screen mode with typed-in commands
- o EMACS-32 -- available from third party
- o TECO/TV -- Integration tools tape, unsupported
- o SED -- Integration tools tape, unsupported
- o TPU -- Programmable editor

SYSTEM ROUTINES

- o Languages which can call:
  - MACRO, BASIC, BLISS-32, C, COBOL(-74), CORAL, DIBOL, FORTRAN, Pascal, PL/1
- o Arguments are passed by
  - value, reference, or descriptor
- o Condition value always returned
- o Kinds of system routines:
  - system services
  - run time library routines
  - utility routines

SYSTEM SERVICES

Functions:

- o Security -- check protections, ACL's, identifiers, disk erase
- o Event flag services
- o AST's -- set and deliver
- o Logical names -- create, delete, translate
- o I/O -- channels, QIO, device, volume mailbox, breakthrough, message to job controller, operator, etc.
- o Process control -- creation, state, priority, privileges
- o Timer and time conversion
- o Condition handling setup
- o Memory management -- working set, global section, lock page, swap mode, stack limits, map section

I N T R A - P R O C E S S  
C O M M U N I C A T I O N

- o Lock management -- enqueue/dequeue,  
get lock info  
Does deadlock detection; chooses victim process and denies the lock it's waiting for. Program knows lock was granted by event flag setting, by AST delivery (routine having been set up for execution upon AST receipt), or by polling the lock status block. Same as for QIO request.

R U N T I M E L I B R A R Y

- o Same calling and return standards as system routines
- o Use RMS for file I/O
- o Execute in same access mode as caller
- o Major subsets
  - mathematics
  - resource allocation
  - condition handling
  - screen management
  - image/process handling

O T H E R S Y S T E M R O U T I N E S

- o CLI parsing
- o RMS services
- o File definition language routines
- o Sort/merge routines
- o File conversion services
- o Data compression/expansion
- o EDT access
- o Librarian routines
- o Print symbiont, job controller interface

I N T E R - P R O C E S S  
C O M M U N I C A T I O N

- o Common event flags (fast, but only bits)
- o Logical name tables (limited data amounts)
- o Mailboxes (limited data amounts)
- o Global sections (fastest)
- o Lock management (fast, but only bytes)
- o Shared files (slowest, but unlimited data)
- o DECnet task-to-task
- o Kernel mode AST

(Between different images executed by same process)

- o Local event flags
- o Per-process common blocks
- o AST's
- o Symbol table

F I L E S Y S T E M

- o File organizations
  - sequential
  - relative
  - indexed
- o File access modes
  - sequential (works with all organizations)
  - random access by key value (for indexed only)
  - random by relative record number (sequential and relative)
  - random by record file address (works with all)
  - block I/O
- o Record formats
  - fixed length (all organizations)
  - variable length (all organizations)
  - variable with fixed-length control (sequential and relative)
  - stream (terminator delimited -- disk sequential only)



## TOPS-20 System Directions

Don DenTandt  
The Trane Company  
LaCrosse, Wisconsin

### ABSTRACT

David Braithwaite of Digital Equipment Corporation presented an update of currently supported versions of TOPS-20. Facilities and support of each version were outlined. Guidelines for implementation of new operating system versions were given.

### PRODUCT DESCRIPTION

There are presently five versions of TOPS-20 in various stages of support and development. Each version has been released for a specific application and contains special features:

- 4.1 - TOPS for the 2020 and Model A
- 5.1 - The present regularly supported TOPS
- 5.4 - For NIA and TCP/IP sites
- 6.0 - CI and HSC disk support
- 6.1 - In field trial, supports DECNET Phase IV, CFS, LAT, NIA-20

### STATUS AND PLANS

4.1 Original release of this version for 2020's and Model A's was in the Spring of 1982. Included is a DECNET Phase II implementation. No additional facilities are planned, but DEC is consulting with third parties to possibly get DECNET Phase IV implemented on 2020's. Autopatch support will continue.

5.1 This version was also released in the Spring of 1982. The last autopatch tape, number 11 is planned for early fall this year. Version 5.1 will be superceded by the final release of 6.1. Maintenance will continue for six months after 6.1 release.

5.4 Originally released in September 1984, this special TCP/IP release has been maintained over the ARPANET. It has very limited availability and according to Braithwaite is "only for the brave." Support is direct from engineering. This release will be superceded by 6.1.

6.0 Another engineering supported release of TOPS-20, version 6.0 was first released in December of 1984. The first update was sent out in April of this year. The last of the

quarterly updates will come in September. There is no SDC support, and engineering has tried to keep distribution limited. Engineering reports that occasionally a site experiments and finds that this version does indeed contain no support for the CI and HSC, just as advertised. Version 6.0 will be superceded by 6.1.

#### 6.1 Field Trial

The field trial will end in July and limited support will continue through August. Version 6.1 will be released to SDC early in September which should put it on site in November. DEC Common File System (CFS) is supported. The initial maximum configuration supported with be two cpu's with 3 HSC's with facilities to access front end disks on dual ported drives.

Configurations larger than officially supported are presently working at customer sites, but DEC is hesitant about immediately supporting larger systems for a number of technical reasons. Larger configuration support will come as part of DEC's commitment to continue to support Large Systems.

Updates to DUMPER, MS and RP20 will be included in the final release of 6.1.

### 6.1 RELIABILITY

Field test for version 6.1 has been the largest ever run with 26 sites, including 18 customer sites involved. There are at least 42 cpu's with potential of up to 60 using this version. Because of this and other factors, the final release of 6.1 will be the best tested, most stable TOPS-20 ever. Braithwaite reports it as being the most thoroughly tested and stable TOPS ever in field test.

### PERFORMANCE

For all of the added features and facilities, 6.1 will cost little in performance. Some sites have even reported an increase in



performance. Performance comparisons are to the latest autopatched version of 5.1. The advertised degradation in going to version 6.1 from 5.1 is 8 percent.

Changes in hardware configuration possible because of new operating system features may allow a reduction in floor space.

File synchronization across CFS is presently being done with OPEN/CLOSE sequences by some third parties. This is quite costly. DEC is working with this vendors to find a better solution.

#### IMPLEMENTATION RECOMMENDATIONS

Because of the many new facilities which closely involve system administration and operations, intelligent choices must be made before implementing new features. For example, choices must be made about which network connection to use, how to implement CFS, the use of LAT's and the effect of larger memory (4 MW now accesible) on system performance. Password encryption is available by structure and will probably require a change of procedures for lost passwords. This is not something difficult, just something changed.

Clustering may require changes to structure names because no two structures in a system can have the same name. Operations and system administration are the most affected areas with version 6.1. Users will notice very little difference in system operation. EXEC enhancements will be the most obvious change to users, but only to those who use them.

Orderly implementation is a must:  
First, be sure you can fall back to a previous usable implementation.  
Second, be sure that operations is well trained.

A logical plan of attack is to first replace 5.1 with 6.1 without implementing any of the new features. This is a good falling back place for later implementations.

Next, be sure that procedures involving the network and shared drives are in place and well understood and usable by operations staff before production implementation of these features.

In the "fall back" vein, be able to split systems before a full commitment to shared disks. Implementing CFS with mostly exclusive disks will allow operations to come up to speed without giving too much to users. Begin without the MCSP server.

#### CFS IMPLEMENTATION

The Common File System (CFS) can cut down on pack changing in multi-cpu environments. Bringing CFS up slowly is important.

1. Begin with all exclusive disks. This will allow check of the MOUNT operations facilities.
2. Allow limited disk sharing and experiment. Note performance changes and how your procedures work.
3. Share mountable dual ported RP06's next.
4. General sharing of HSC50 and RP07 disks should follow this.
5. The MCSP server is a heavy resource user and should wait until this point. Again, be sure to be able to fall back if the gains are less than the losses.
6. LAT and other configuration changes will follow.

#### THE FUTURE

The SPR backlog has been going down and should continue to decrease.

We are presently two years into DEC's promise of 5 year development and 10 year support on DECSYSTEM 10/20's. The planned major development work is nearing completion.

During the next 3 to 4 years, efforts will be spent on finishing commitments, in particular to supporting larger CFS configurations. Continued work on the "integrated environment" is also high priority. The continuing concerns of performance and maintainability, especially in the area of decreased complexity, will also be addressed in DEC's continuing efforts. Present plans call for a final development release in the Spring of 1988. As priorities have not yet been officially established, DEC is open to suggestions and comments from users.

#### CONCLUSION

DEC's commitment to continue 10/20 development is producing a very usable product that should meet the growing needs of users for some time. DEC has obviously committed many engineering resources to working with customers and is bringing many features that have been asked for into the next major release of TOPS-20.

Questions and answers were postponed until the end of the following session, as many of the questions would probably be answered there.

TOPS-10 NOVICE Q & A  
TOPS-10 Q & A

Jack Stevens  
The Gillette Company  
Boston, Massachusetts

ABSTRACT

Members of DEC's TOPS-10 development groups answered questions from the audience. The questions and answers are summarized here.

AGENDA

A. Questions and Answers

Carl Appelhof, Bill Davenport, Joseph Dziedzic, Warren Sander; Dawn Banks, Raun Boardman, Jim Flemming, Bob Houk, Barb Huinzenga, Don Mastrovito, Larry Sendlosky, Nick Tamburri, Kimo Yap (Digital Equipment Corporation)

The TOPS-10 Novice Q & A and the TOPS-10 Q & A sessions are presented here in a combined summary, reflecting the actual composition of the audience and the questions.

Three members of DEC's High Performance Systems/Clusters Engineering -- Carl Appelhof, Bill Davenport, and Joseph Dziedzic -- plus Warren Sanders of Large Systems Marketing Technical Support were available to answer questions from the audience. Dawn Banks, Raun Boardman, Jim Flemming, Bob Houk, Barb Huinzenga, Don Mastrovito, Larry Sendlosky, Nick Tamburri, and Kimo Yap also answered questions via telephone hookup from Marlboro.

A user who is running RSX-20F version 15-06 under 7.01A finds that users on a terminal switch see a fragment of the welcome message when they autobaud. This is fixed in 7.03 by having INITIA not put anything into a line's output buffer until it is dialed into. A PCO could be generated for 7.02 that might work under 7.01A.

In an SMP system, an imbalance in the indicated number of TTY characters transferred on the policy CPU versus the other CPU(s) is not necessarily a problem, as all network TTY traffic is handled by the policy CPU (at clock level for the DN20/DN87-based network I/O. 800 characters per second TTY output is a relatively insignificant number for a KL).

With Galaxy v.4.1, KJOB's in batch streams can sometimes make jobs hang in TO state in LOGOUT.

Eight-bit terminal support in 7.03 will allow transmitting and receiving of full 8-bit characters. If the terminal is opened in 7-bit mode, the input and output will be translated to and from 7-bit, to avoid breaking older programs. To use all the characters, the terminal will have to support 8-bit characters, and the program will have to open the terminal in I/O mode 4.

Having an additional model number field in the terminal type characteristics for user-defined terminals is being considered, but will not be done in 7.03.

A problem with RSX-20F sometimes losing its KL Config file and coming up in KLI does seem to be a bug, but a directory should be taken of the front-end area to determine if the file is lost or just inaccessible.

MF20 and MG20 can (probably) be intermixed.

Just reading a badly fragmented file opened in update mode can cause monitor detected file checksum errors.

To get rid of the "File errors exist" message when logging in (after cleaning up the file errors), log in with the /QUOTA switch.

The "DECsystem-10 not running" message that is sent to all terminals when TGHA runs is caused by the monitor, not TGHA.

There is no standard 10 utility that will write a ANSI-labeled tape that the VAX will read easily (other than the less-than-robust TENVAX). This is because the 10 only supports volume labels, not the file labels that the VAX uses. Support for label processing, as TOPS-10 offers, is not the same as support for labeled tapes, as there is no guarantee that what is actually written on the tape agrees with what the label says. There is a document that deals with reading and writing VAX tapes (albeit for TOPS-20) on the Integration Tools Tape.

On dual-ported RP20's, paths to the drives get lost randomly. Trying to reestablish the path by detaching and reattaching the drive seems to corrupt data and crash programs. Because of the frequency of reformatting, replacing HDA's, and microprocessor restart errors, a hardware problem

was suggested (though the questioner's Field Service office didn't agree).

A suggestion was made to change the hardware configuration dialogue of MONGEN to separate actual hardware-related items from the non-hardware-related items (such as ersatz device definitions). This goes against the grain of current development, which is to reduce the number of configuration files. A further suggestion was made to remove these types of items from MONGEN altogether, to be read, perhaps, at ONCE-only time, to avoid having to rebuild the monitor as frequently. Anything like this cannot be done for 7.03.

System-wide PATHological device names were requested. They are being considered, but will not appear in 7.03.

Will an RH20/DX20/TX02 tape controller combination give better performance than a DX10/TX02 combination? Not without the 200 ips tape drive (which DEC won't sell, anyway).

In hacking TOPS-10 to change TTY output from even parity to space parity, changes have to be made not only to SCNSER, but also to RSX-20F (xon, xoff, and DECsystem not running message) and TTDINT (DECsystem-10 continued message).

If one includes all the new hardware, software, and DECnet functionality of the 7.03 monitor, it becomes about 130 pages larger than 7.02.

Backup in 7.03 supports labeled tape reel switching with indications in the listings.

Backup in 7.03 will not support larger tape block sizes for 6250 bpi tapes.

[At this point, a short recess was declared until the beginning of the nominal start of the TOPS-10 Q & A session. An occasional giggle could be heard over the telephone hookup.]

MDA in 7.01A can cause all tapes to hang in event wait (for itself). This was fixed in an autopatch tape (and is a two-line PCO to TAPUO).

PULSAR regularly dying with a CCD stopcode when a volume switch is done to an uninitialized tape (or one initialized at the wrong density) was fixed with a couple of PCO's.

The requester of larger block sizes for BACKUP running at 6250 bpi was thanked for his suggestion and assured that it was being considered for a future release (not 7.03).

There will be no new tape utilities on the 7.03 CUSP tape (such as one to read VAX labeled tapes). This is something that might be requested of LSM on an Integration Tools Tape. It would take a great deal of work to put it into PULSAR. The 130-page increase in size of 7.03 does not necessarily include such things as CI buffers, LAT buffers, DECnet buffers, etc.

The RUN and GETSEG UO's still trash the accumulators in software channel 0. This bug has been around long enough to develop a sentimental following. Also, no one wants to break programs that depend on having their accumulators trashed.

Fact file accounting is completely removed in 7.03. The only vestige is a program to convert Usage files to Fact format.

If one does a LOOKUP or ENTER on a file, using an extended block, then tries to RENAME the file, RENAME also expects an extended block. (Bug: SPR).

There is no chance that anything like the TOPS-20 PMAP UO will be implemented in TOPS-10.

PULSAR code is brittle; fixing problems is difficult. No date can be given when it will live happily ever after (as opposed to fixing individual bugs).

TOPS-10 Monitor Directions

Susan M. Lamaestra  
Johnson & Johnson  
Raritan, NJ 08869

ABSTRACT

William Davenport of Digital Equipment Corporation, who is a Principle Software Engineer involved with the development of the TOPS-10 monitor product discussed features of version 7.03 and future directions.

AGENDA

- A. Slide and lecture presentation of 7.03 monitor features.
- B. Questions and answers.

MAJOR HARDWARE ENHANCEMENTS -

- Support for CI20 and HSC50 (RA60 and RA81 disks)
- Support for NIA20 (ETHERNET)
- IAT support via DECSA (32lines), DECSERVER 100 (8lines), IAT-11
- Support for mixed MCA-25 and non MCA-25 KL's in a SMP environment
- Software distribution package available with one Q# for mixed 1090 and 1091 sites.

MAJOR SOFTWARE ENHANCEMENTS -

- DECNET Phase IV
- CTERM - (layered on DECNET for support of heterogeneous terminals.)
- User mode extended addressing
- Alternate context; Push/Pop
- Security Features -
  - Password encryption, longer passwords and usernames (up to 39 characters), password expiration, LOGIN support for validation failures, SET PASSWORD command allows password change at monitor, MONGEN parameter disallows use of commands when not logged in (SYSTAT, Queue, etc.)

Galaxy Enhancements -

- CATALOG utility (replaces STRLST)
- Event queue scheduling
- OPR commands -
  - Set core max, Set log max, OPSER commands (KSYS, etc.), NCP support for all DECNET IV, ICP support for IAT terminals
  - Multi threaded FAL
  - Disk quota support (QUOTA.SYS)

GENERAL ENHANCEMENTS -

- NI supports any ETHERNET protocol; Full DECNET Phase IV requires NI.
- Installing either CI or NI in KL will use all four top RH20 slots. (NI uses 4-5, CI uses 6-7)
- SAVE.UUO support for multi section MACRO programs
- /USE switch for RUN, GET, MERGE commands; forces relocation to different sections.
- Commands may be defined auto pushable.
- HELP will no longer destroy core image.
- User defined commands via DECLAR program.
- New REACT - (GLXLIB based); SYS:ACTDAE.SYS replaces ACCT.SYS and AUXACC.SYS
- Paging enhancements
- Monitor support for 8 bit ASCII terminals

NOT SUPPORTED -

- HSC50 based tapes, DX10, CI with DECNET IV.

Product due for release February/March, 1986.



# Reading Foreign Tapes on a DEC-20

Betsy Ramsey  
American Mathematical Society  
Providence, Rhode Island

## Abstract

Reading and writing non-DUMPER tapes is difficult on the DEC-20, primarily because little documentation exists on the methods to use. Berkley Shands of Emerson Electric, author of several tape utility programs, offered advice on reading and writing non-DUMPER tapes. Members of the audience asked questions and presented their own ideas and solutions to the problem.

## Reading Foreign Tapes

It is difficult for users to read non-DUMPER tapes from other sites because little or no documentation exists on how to do it.

When a foreign tape is received at your site, the first thing to do is determine whether or not it is labeled. The easiest way to do this is to mount it on a drive that has Automatic Volume Recognition (AVR) enabled. MOUNTR will inform you via OPR whether the tape is labeled and, if it is, the type of label and volume identifier. Even if the sender tells you the tape is a standard ANSI-labeled tape, often you mount the tape and find it is unlabeled. It is easier not to take anyone's word for it.

If the tape is ANSI- or EBCDIC-labeled, you can use the EXEC DIRECTORY command to get a list of files on the tape, and the COPY command to restore the files to disk. COPY will translate EBCDIC to ASCII if necessary. For text files, use the BYTE 8 and ASCII subcommands to COPY. For files you want to restore in 8-bit format, use just the BYTE 8 subcommand. Be sure to set the tape record length appropriately (4096 is the maximum for ANSI standard tapes)—the default 512 is usually too short.

If the tape is not labeled, you can still try the COPY command, although you will not be able to get a DIRECTORY. Try it both with and without the BYTE 8 subcommand before you give up on COPY.

A number of user-written utilities exist for looking at the tape and determining its format. Among these are TOPS-10 DIRECT and Nelson Beebe's TLOOK.

If you are unable to restore the files with the COPY command, you can try other utilities. RFIL11 from the Integration Tools Clearinghouse does a good job of reading ASCII files from ANSI-labeled tapes. To use it and other programs that expect to process the tape labels, you will either have to assign the tape drive directly or mount the tape with the /LABEL:BYPASS switch. The ancient CHANGE program does run on TOPS-20 with PA1050 but it is slow.

## Writing Foreign Tapes

The first problem in sending out a non-DUMPER tape is determining what format the remote site can read. When in doubt, send an ANSI-labeled tape with fixed format records of length 80 with no blocking factor. If the tape is going to an IBM machine,

use fixed records of length 80 and a block length of 800 (that is, blocking factor 10). If the tape is going to a DEC machine, use variable length records with a maximum record length of no more than 4096.

You can create an ANSI-labeled tape with variable length records as follows. Use OPR to initialize a ANSI-labeled tape. Mount the tape using the volid specified at initialization. Change 7-bit ASCII files to 8-bit before putting dumping them to tape. The DEC-supplied utility RSXFMT will do this nicely. Set the tape record length to 4096 or less. Use COPY with its BYTE 8 subcommand to put the files on the tape. When you send the tape out, be sure to include a directory listing and information that the tape contains variable length records with a maximum record size of whatever.

COPY cannot create fixed length records. For this, the best utility is TAPE11 from the Integration Tools Clearinghouse. It can create fixed or variable format tapes, with or without labels. It can handle both ASCII and EBCDIC data. It will accept either 7-bit files or 8-bit files, depending on the format you select.

Utilities like TAPE11 expect to do their own label processing, so you must assign the tape drive directly, mount the tape with /LABEL:BYPASS or start with an unlabeled tape.

If you simply need to make a copy of a tape, utilities such as MTCOPY, TAPCOP and DUMCPY exist. A couple of these have appeared on past TOPS-20 SIG tapes.

## Information from Q&A Period

If you don't have any luck interpreting a tape, COPY its first file to disk and use CHANGE, FILDDT or other utilities to examine it there. It's too slow to keep rewinding the tape.

Bob Ham of MA-Com/Linkabit reports that they have had good success reading tapes using a simple Cobol template program.

Rather than use DUMPER Interchange mode to create TOPS-10 BACKUP tapes, it is better to obtain a copy of BACKUP itself. With a three-line DDT patch, it will run under TOPS-20.

This paper was produced by the T<sub>E</sub>X typesetting system, and printed on a low-resolution laser printer.



## TOPS-20 Utility Closet

Steve Attaya  
Wiener Enterprises  
New Orleans, Louisiana

### Abstract

The SIG tape copy process was reviewed. TOPS-20 users identified programs that they will submit to the SIG tape. TOPS-20 users indicated programs they would like to see on future SIG tapes.

Before the presentation of utilities began, Betsy Ramsey, member of the Large Systems SIG Steering Committee, covered some administrative matters:

- The Fall 1984 SIG tape was distributed through the National LUG Organization. The one attendee who had requested the tape from his LUG had not received it, however.
- The Spring 1985 tape will also be distributed through the NLO. Users should contact their LUG to obtain a copy of the tape when it is available (after early- or mid-August 1985).
- Steve Attaya would be the new TOPS-20 Tape Copy Coordinator.
- All submissions to the SIG tape must be accompanied by a completed Tape Copy Release form filled out by the author or their organization.
- *No proprietary programs, modified or otherwise, can be included on the SIG tape. Therefore, if you wish to submit modifications to proprietary software, note that only comparison files will be accepted: send SOUPR .COR files or REDIT .RED files if possible, otherwise FILCOM/SRCCOM output. Be sure to indicate the version number and write date of the distributed version of the program.*

Steve Attaya then covered the new way in which material would be identified for inclusion on the SIG tape. In the past, the material that appeared on the tape was only that which someone voluntarily submitted. In the future there would be a concentration on users identifying material they would like to have included and then trying to obtain that material. It was indicated that this was an experiment and that it was dependent both on users stating what it was they would like to have and also on finding someone to submit those programs. To this end, Steve stated that users should identify where they had seen the software requested if they had seen or heard of it in use.

The following is a list of programs/items that were requested by attendees:

- BitNet support for TOPS-20
- "C" compiler for TOPS-20
- Current version of MM mail system
- WATCH data reduction and reporting system
- Directory mover/creator
- Kermit for Visual Tech 1050
- Menu system for TOPS-20
- PROLOG compiler
- System M Microfiche device driver
- Directories of old TOPS-20 SIG tapes
- Modifications to LOGOUT to simulate terminal server break character(s)

Steve Attaya noted that the following software had been identified earlier, and that he would try to get it on the SIG tape:

- EMACS Libraries (Nelson Beebe, University of Utah)
- EMACS Libraries (Rob Austein, MIT)
- Current version of EMACS (MIT)
- Current version of PASCAL, SAIL compilers; an "ancient" version of SCRIBE (Chuck Hedrick, Rutgers University)
- Current KERMIT distribution from Columbia (via Reed Powell of Digital)
- PC-DOS KERMIT that emulates a VT100 (Tad Marshall, Banker's Trust)

The following is a list of attendees that said they would make submissions to the Spring 1985 TOPS-20 SIG tape:

- Mike Joy (First Church of Christ, Scientist, Boston, MA)
  - Operations and production management tools
- Osman Ahmad (Association of American Railroads, Chicago, IL)
  - EMACS libraries/modifications
- Clive Dawson (MCC, Austin, TX)
  - Galaxy interface to Imagen laser printer
  - Text of "Alice's PDP-10" (also to be included in *AT LARGE*, the Large Systems SIG newsletter)
- Betsy Ramsey (American Mathematical Society, Providence, RI)
  - LPTSPL modifications for spooling to terminal lines
  - T<sub>E</sub>X & METAFONT typesetting system (if Donald Knuth, its author, would submit it)
  - Updated WPSIM word processor program (if its author Douglas Bigelow would submit it)
- Ronny Appleby (Woodward Communications, Dubuque, IA)
  - LPTSPL modifications
- Bob McQueen (Stevens Institute, Hoboken, NJ)
  - A public domain version of "C"
- Richard Messinger (Eastman Kodak, Rochester, NY)
  - LPTSPL modifications for spooling to terminal lines
  - SETTERM/TERMTYPE utilities to modify terminal parameters, FORTRAN/MACRO utility to set up terminal
  - JKILLR utility to log out idle jobs
  - Minor modifications to NFT
- Reed Kelly (Shearson Lehman Bros, NY, NY)
  - LPTSPL modifications for writing EBCDIC data
- Robert Jones (University of DC, Washington, DC):
  - Modifications to ACCT20 accounting program
- Charles Lewis (Electro Scientific, Portland, OR)
  - /DESTINATION implementation for TOPS-20



- Ray Cadmus (Computerized Business Systems, Moberly, MO)
  - Has some integration tools that will be on the Integration Tools Clearinghouse tape, not the SIG tape.

At the end of the session a note was made that Digital is going to implement "barebones" TTY support in LPTSPL for 6.1. It was recommended that the SIG tape for FALL 1985 have as a goal a number of LPTSPL modifications submittals.

The closing date for submitting utilities to the Spring 1985 SIG tape will be July 15, 1985. Further information will appear in the next edition of *AT LARGE*, the Large Systems SIG newsletter.

MG-10/MH-10 MEMORY UPGRADE AND  
MULTI-PORT INTERNAL MOS MEMORY

Donald A Kassebaum  
University of Texas at Austin Computation Center  
Austin, TX 78712

ABSTRACT

The need for upgrading existing LCG (MG/MH) memories been satisfied at this site by replacing core memory modules with DEM type MOS memories. Utilizing the same concepts, a multi-port internal memory subsystem for SMP has been designed.

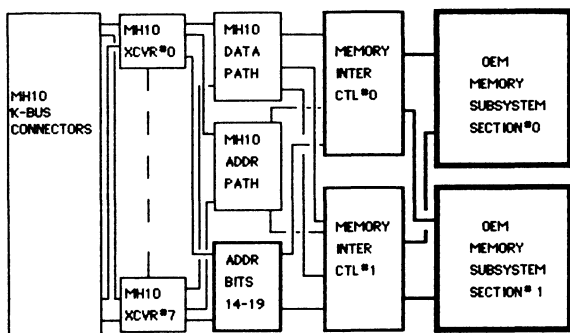
AGENDA

- A. Introduction by Chairman.
- B. Presentation by Ed Jordan of Goodyear Atomic Corporation.

MG/MH10 MOS Upgrade

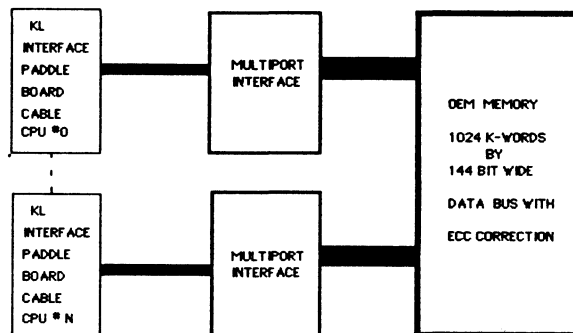
- 1) Each memory controller may be upgraded to maximum of 2M words.
- 2) Each memory subsystem may be upgraded to 4M words.
- 3) Two way interleaving within the memory subsystem and two-four way interleaving is supported between two or more boxes.
- 4) Completely diagnostic transparent.

MULTI-PORT MOS  
BLOCK DIAGRAM



| MG/MH10 MOS<br>PERFORMANCE COMPARISON |                 |               |                 |
|---------------------------------------|-----------------|---------------|-----------------|
|                                       | MH10            | OEM #1        | OEM #2          |
| READ ACCESS (NS)                      | 735             | 580           | 350             |
| READ CYCLE (NS)                       | 1170            | 580           | 350             |
| WRITE CYCLE                           | 1040            | 670           | 400             |
| READ-MODIFY-WRITE CYCLE               | 1730            | 970           | 730             |
| POWER REQUIREMENTS                    | 10 AMPS (256 K) | 8 AMPS (4 MW) | 8.7 AMPS (4 MW) |
| POWER DISSIPATION                     | 2400 WATTS      | 1500 WATTS    | 1800 WATTS      |

MG/MH10 UPGRADE  
BLOCK DIAGRAM



Multiport Interface Functions:

- 1) 4 word multiplexing producing 144 data bits to OEM memory
- 2) Uncorrectable ECC bad parity generation.
- 3) Overlapping or Pipelining multi-cpu memory requests by not starting OEM memory cycle until last word from KL-10 is in transit.
- 4) Future address mapping registers for memory capacity up to 32 megawords.
- 5) Full DMA-20 emulation to maintain diagnostic compatibility.

MULTI PORT INTERNAL MEMORY  
PERFORMANCE COMPARISON

|                        | MF20     | OEM #1 |
|------------------------|----------|--------|
| READ CYCLE (NS)        |          |        |
| 1 WORD                 | 730      | 430*   |
| 2 WORD                 | 930      | 430    |
| 3 WORD                 | 1130     | 430    |
| 4 WORD                 | 1330     | 430    |
| READ ACCESS (NS)       |          |        |
| 1 WORD                 | 800      | 450    |
| 2 WORD                 | 1000     | 650    |
| 3 WORD                 | 1200     | 850    |
| 4 WORD                 | 1400     | 1050   |
| WRITE CYCLE (NS)       |          |        |
| 1 WORD                 | 860-1130 | 450    |
| 2 WORD                 |          | 450    |
| 3 WORD                 |          | 450    |
| 4 WORD                 | 1260     | 450    |
| READ-MODIFY-WRITE (NS) | 1000     | 670    |
| POWER CONSUMPTION      | 3700 W   | 1100 W |

\*REDUCED CYCLE TIMES DUE TO OVERLAPPING WORD TRANSFERS.



Peter B. Galvin  
University of Texas at Austin Computation Center  
Austin, TX 78712

ABSTRACT

Representatives of Version 6.1 field test sites discussed their experiences of converting from V5.X to V6.1 and their impressions of TOPS-20 V6.1.

AGENDA

A. Introduction by Chairman.

B. Presentations by Richard Jannick of Abbott Laboratories, Robert Ham of M/A-COM LINKABIT, and Sean Welsh of Stanford University's LOTS computing facility.

C. Questions and answers.

Richard Jannick:

Abbott labs maintains 1 DEC-2060 and 2 DEC-2065s, with many RP06s and RP07s. They have an HSC50 and 3 RA81s clustered between all 3 systems. They also have 3 DN20s.

All of the systems are allowed access to all of the disk structures. This required rebuilding the PS structures to avoid name conflicts. In addition, all login directories are duplicated on the 3 systems, allowing users to login to all of the systems in the same way. After login, users are moved off to a shared structure to work.

The CFS system has enabled the users to copy data directly from any structure on the any of the systems, making the NFT program unnecessary. It is also now possible for all users to continue work even when one of the 3 systems is down.

One problem of the shared disk environment is that some files may be accessed by multiple users, and, unless they are careful, the users could destroy the work of others.

Other points include:

- CTERM works well between 20s, allowing multiple connects between systems.
- 9600 autobaud detect works as advertised.
- New login security works well, limiting the number of wrong passwords that can be entered before the system either ignores the user or drops the phone connection.

Along with the clustering of the DEC-20s came the need for operator retraining. Operators could no longer shut down a system with impunity: a parser "halt" is needed at the end of a system shutdown to signal other systems on the cluster. Without the "halt", all systems on the network would hang.

Hardware problems with disk drives on the cluster tend to hang the entire cluster.

The only program incompatibility comes in program which have job-number based tables built into them. Programs of this type must be rewritten to take into account the new global job numbers associated with clustering DEC-20s.

The performance of an individual system has changed between V5.1 and 6.1 of TOPS-20. Some benchmarks are:

- Number crunchers gained 15-20%
- I/O bound programs gained 15%
- 1022 applications lost 40%! (possibly because of multiple file opens and closes) (this is for batch jobs only).
- Load averages seem about even.

Bob Ham:

M/A-COM LINKABIT has five 2065s and 25 VAXen networked with DECNET and Ethernet. Only 3 systems were involved in the field test, and those 3 systems have no CFS, only the NI hardware.

The conversion from V5.1 to V6.1 was "painful" for several reasons. Password encryption was troublesome since M/A-COM already was using another password encryption scheme, and incompatibilities existed.

The first time V6.1 is booted, it installs PPN support on the system disk. The process is not robust, however, and one time resulted in a trashed PS: structure. The cause was a bad directory, so running CHECKD before the conversion would have avoided the problem.

Version 6.1 itself is very nice, with a few exceptions:

- VT52 and VT100s no longer pause after every 24 lines by default, which is disconcerting to users who expect them to do so.
- Spoolers which have been modified to spool to daisy-wheel printers at 1200 baud will no longer run at that speed. The front-end is slower (for some reason) to acknowledge XOFFs than it used to be, so the printer's buffers tend to get overrun.
- The monitor is fairly reliable now

In general, the NI interface is very good, and easily replaces the DN20. It's much more reliable, cheaper to maintain, and simpler to use.

CTERM between VAX VMS and TOPS-20 does not work for most programs. Both multi-line TEXTI jsys calls and control-character passing fail. (This is apparently a problem with VMS).

The monitor can no longer be patched with FILDDT.

In terms of performance, little change has been noticed by most users, although some improvement has been seen in COBOL batch jobs. Also, MAILER and MMAILR are slowed by the fact that they loop through the job tables, which have grown from 128 jobs to 512. MMAILR was at one point 1.75 hours behind in delivering local mail.

The new DUMPER is still slightly incompatible with the old, making it difficult to read V6.1 tapes under V5.1.

Sean Welsh:

It should be noted that installing either the CI, or the NI, or both, takes up 4 RH20 slots.

LOTS only uses the CIs, not NIs, but has ethernet support on the DEC-20s via another device.

Adding a CFS and V6.1 to a system does decrease system performance, by 20% at LOTS. There is a noticeable improvement in the performance when the CFS is turned off and the systems are essentially unclustered.

In general, both V6.1 and the CFS are reliable. However, if a system in the CFS hangs, the other systems will tend to hang too.

Using the MSCP service degrades system performance, and causes the clustered systems to be more dependent upon each other (so one system crashing will cause the others to crash).

When converting, make sure password encryption is not turned on until V6.1 stabilizes, since it is impossible to go back to V5.1 with encrypted passwords.

LOTS also noted the PPN conversion problem, and it is suggested that a catastrophe tape be made before this conversion is attempted.

Questions and Answers:

Performance of 6.1 has been improving with each distribution, with the worst performance being seen in V6.0.

MANAGING A LARGE MULTI-SYSTEM  
SITE -- A CASE STUDY

Michael D. Joy, Manager  
Data Processing Division  
First Church of Christ, Scientist  
Christian Science Center  
Boston, Massachusetts 02115

ABSTRACT

Michael Joy presented a case study of how his organization manages a large business with multiple DECsystem-10s. The techniques used enable them to actually reduce operations staff during a period of rising production demands. A major reason for the increase in productivity was the automation of functions, such as scheduling, job set-up and shift reports. These tools are written in COBOL and are available on the Spring 1985 DECUS library tape.

Topics covered during Mike's presentation were: his site environment, site scheduling standards, data entry standards, production improvements (by using computer generated runbooks, schedules, job set-ups and reporting on scheduled jobs), and other hints for improving the workplace. Mike had samples of the runbook standards, an actual runbook, an Operators Handbook and Online Data Entry instructions.

Appended to this report is a copy of the slides that Mike used during his presentation. These slides are referenced during the following description of the presentation.

Site Environment

- . 1 KL1090 with 2 MEG of memory.
- . 1 KL1077 with .5 MEG of memory.
- . 90-100 jobs on weeknights.
- . 100-150 jobs on weekends.
- . 45 applications.
- . Heavy printing.
- . Dozens of custom forms.
- . Very high quality is required.

Standards

- . All jobstreams have an ID. See example on page 1 of appendix.
- . Scheduling cutoff for batch submissions is 2 PM for initiation at 4:30 PM.
- . Online timesharing is cutoff at 4:15 PM.
- . Cutoff for submission of jobs to the data entry staff is 4 PM.

Data Entry Evolution

- . Data entry was initially card input.
- . Next came a disk to tape system.
- . There was a decision made to go to online entry. They now have 3 data entry personnel. They wrote their own online entry system in MACRO-10 (to be submitted to the DECUS library).
- . Pages 2 and 3 of the appendix show formats. Four formats are possible:
  - : X = all printable characters
  - : N = numbers only
  - : A = alpha only
  - : D = date
- The examples on page 2 illustrate the flexibility of the formats for data entry. You are able to define field, beginning column, field length, and prompts for each field.
- . Online entry features:
  - : Moved easily from key to tape system.
  - : Simple for novices to learn.
  - : Flexible.
  - : Requires a minimum of system resources

## Production Improvements

. In this part of the presentation, Mike reviewed functions that were automated to improve productivity.  
. Although this is a DECsystem-10 site, the ideas could be used at a VAX or DEC-20 site. Since the programs are in COBOL, they could be easily converted.

### Runbooks

These work so well that the operator is able to handle 90% of the problems that arise. Pages 4 through 7 of the appendix show items in the runbook that are required for each jobstream.

. Page 4 shows the table of contents for the jobstream. Items required by the jobstream are marked with an X.

. Page 5 is a sample of the transmittal sheet for a jobstream. Note that it is signed by the client.

. Page 6 is the resource sheet on which you show items, such as tapes, disks, ppn.

. Page 7 is the dispatching sheet that shows what is to be done with all output reports and tapes.

### Schedules

Page 8 of the attachment is a copy of the batch schedule. You start with a skeleton schedule with empty job slots. As jobs are scheduled, they are placed in a job slot. Each slot shows the job number, jobstream ID and priority. When the job is done, it is checked off in the last column. Page 9 of the appendix shows notes that can be entered by the scheduler.

### Job Setup

Page 10 of the appendix shows the job setup sheet with the tape labels. These are automatically produced by the computer. The transmittal checklist for the schedule on page 11 of the appendix is also automatically produced. This gives the scheduler a quick check off capability. Page 12 of the appendix is the automated check list for the data entry personnel so that they can ensure that data is ready for the scheduled runs.

### Reporting on scheduled jobs

Page 13 of the appendix is the automatically generated distribution so that you can check off creation, handling, and delivery of reports.

Page 14 of the appendix shows that the computer also can automatically generate labels to go on packages that are mailed. Page 15 of the appendix is a Job Analysis Report that can be generated by the system if information is entered concerning specific jobstreams. You may want to do this as a reminder in the future for particularly long or troublesome jobs.

## Reporting generated by the system

### Shift Report (page 16 of appendix)

. These are problems and comments on system status or specific jobs.

. These are generated by the operator on the computer during the course of the shift. Miscellaneous comments can also be entered.

. It is printed and distributed to all managers.

. A copy is also sent to the field service engineer.

### Job Performance Memo (page 17 of appendix)

. This can be completed by anyone including operators, programmers, analysts, or managers.

. It indicated any problems experienced with jobs.

. Turnaround in responding to these will be from one day to a week, depending on the problem.

. The manager must review these very carefully to be sure there really are problems. Resolving them without upsetting people takes a lot of tact and diplomacy.

### Problem Reporting Form (pp. 18 and 19)

. This can be completed by clients of the computer center.

. As with performance memos, they should be handled quickly and tactfully.

### Memorandum Format (page 20 of appendix)

. This is used to respond to the problems reported by clients.

. Words must be chosen carefully.

### AMAR Reports (page 21 of appendix)

. This is one of the reports generated by the DECsystem-10 performance monitor.

. This has been a product available from Digital for a set price. It was indicated that it will soon be available on the DECUS library tape.

. This product is extremely helpful in monitoring the DECsystem-10's performance and pinpointing areas that are causing performance problems.

## Documents to Review

Mike had a copy of several documents that his Operations Staff uses. He was willing to send copies to interested people. The documents were:

- : Runbook Standards
- : Sample Runbook
- : Operators Handbook (see page 22 of appendix for table of contents)
- : Online Data Entry Instructions

## Miscellaneous Improvements

Mike closed with a discussion of several other items that have helped them to improve productivity. These were:

- . Better communications among people. You have to work at this constantly.
- . Training for operators. Their Handbook for Operators helps here.
- . Promotion Review Board.
- . Automated Tape Library system.
- . Intra system queing facility between their computer systems.



JOBSTREAM ID

SSSF##

SSS = SYSTEM  
PAY = PAYROLL  
PER = PERSONNEL  
COA = CHART OF ACCOUNTS

F = FREQUENCY  
D = Daily  
W = Weekly  
M = Monthly  
Q = Quarterly  
S = Semi-annually  
Y = Yearly  
X = As required  
U = Unload

## = A unique two digit number

Example: PAYW21 Weekly Payroll edit

JOB:APSD01  
 SELECT  
 101:APS101  
 201:APS201  
 301:APS301

JOB:APS101 REC:80

```

FORMAT:99 NEXT:1 TITLE:"BATCH HEADER"
 FIELD:N COL:1 LEN:4 PROMPT:"YR.,MO."
 FIELD:N COL:5 LEN:4 PROMPT:"FILE NO."
 FIELD:N COL:14 LEN:4 PROMPT:"FILE NO."
 FIELD:N COL:18 LEN:12 PROMPT:"BATCH TOTAL"
 FIELD:N COL:30 LEN:5 PROMPT:"DOCUMENT COUNT"
 FIELD:N COL:35 LEN:10 PROMPT:"ACCT & SUB-ACCT HASH"
 FIELD:A COL:45 LEN:1 PROMPT:"O OR C"
 FIELD:A COL:46 LEN:1 PROMPT:"A,R,OR J"
 FIELD:X COL:47 LEN:10 PROMPT:"DISTRIBUTION"
 FIELD:N COL:69 LEN:2 PROMPT:"ACT.NUMBER"
 FIELD:N COL:71 LEN:2 VAL:"01"
 FIELD:A COL:75 LEN:4 VAL:"CONT"
 FIELD:N COL:79 LEN:2 PROMPT:"BATCH NUMBER"
FORMAT:1 NEXT:2 TITLE:"11 CARD"
 FIELD:N COL:1 LEN:4 DUP:REC
 FIELD:N COL:5 LEN:4 PROMPT:"FILE NO."
 FIELD:N COL:9 LEN:1 VAL:"8"
 FIELD:N COL:10 LEN:2 VAL:"11"
 FIELD:X COL:12 LEN:10 PROMPT:"INTERNAL NO."
 FIELD:N COL:22 LEN:6 PROMPT:"REQUEST DATE"
 FIELD:N COL:28 LEN:10 PROMPT:"GROSS AMOUNT"
 FIELD:N COL:53 LEN:6 PROMPT:"DUE DATE"
 FIELD:N COL:79 LEN:2 DUP:REC
FORMAT:2 NEXT:3 TITLE:"12 CARD"
 FIELD:N COL:1 LEN:4 DUP:REC
 FIELD:N COL:5 LEN:4 DUP:REC
 FIELD:N COL:9 LEN:1 VAL:"8"
 FIELD:N COL:10 LEN:4 VAL:"1200"
 FIELD:N COL:14 LEN:2 PROMPT:"ACT"
 FIELD:N COL:79 LEN:2 DUP:REC
FORMAT:3 NEXT:3 TITLE:"13 CARD"
 FIELD:N COL:1 LEN:4 DUP:REC
 FIELD:N COL:5 LEN:4 DUP:REC
 FIELD:N COL:9 LEN:1 VAL:"8"
 FIELD:N COL:10 LEN:2 VAL:"13"
 FIELD:N COL:12 LEN:2 DUP:REC INC
 FIELD:N COL:14 LEN:2 DUP:REC
 FIELD:N COL:16 LEN:4 PROMPT:"FUND"
 FIELD:N COL:20 LEN:4 PROMPT:"ACCOUNT"
 FIELD:X COL:24 LEN:4 PROMPT:"SUB"
 FIELD:X COL:30 LEN:29 PROMPT:"DESCRIPTION"
 FIELD:X COL:59 LEN:10 PROMPT:"DEBIT AMOUNT"
 FIELD:X COL:69 LEN:10 PROMPT:"CREDIT AMOUNT"
 FIELD:N COL:79 LEN:2 DUP:REC

```

0749J

JOB:ADSX01 REC:80

FORMAT:0

FIELD:X COL:1 LEN:5 VAL: "ADS1A"

FIELD:N COL:16 LEN:1 VAL:"1"

FORMAT:1

FIELD:X COL:1 LEN:5 DUP:AUX

FIELD:X COL:6 LEN:10 PROMPT:"ACCOUNT NO."

FIELD:N COL:16 LEN:1 DUP:AUX

FIELD:X COL:17 LEN:39 PROMPT:"FIELD #, FIELD DATA"

FIELD:D COL:75 LEN:6 PROMPT:"DATE"

X = All printable characters

N = #s only

A = Alpha only

D = Date

-----

JOB:ADSX11 REC:80

FORMAT:1

FIELD:X COL:1 LEN:80 PROMPT:"ENTER ALL 80 CHARACTERS  
ON 1 LINE"

JOBSTREAM ID SYSX99

DATE mm-dd-yy

1.5 RESOURCES

|              |                           |
|--------------|---------------------------|
| PPN          | <u>      ,14      </u>    |
| MTA          | <u>                  </u> |
| DISK         | <u>                  </u> |
| ISQ          | <input type="checkbox"/>  |
| DIRECT PRINT | <input type="checkbox"/>  |

1.6 INPUT TAPES

| TAPE ID | FROM |
|---------|------|
|         |      |

1.7 OUTPUT TAPES

| TAPE ID      | # OF REELS | RETENTION | DESCRIPTION     |
|--------------|------------|-----------|-----------------|
| SYSUNLSYSX99 | 1          |           | RECOVERY UNLOAD |

JOBSTREAM ID SYSX99

DATE mm-dd-yy

3.3 DISPATCHING SHEET

Page 1 of 1

| DISK: <u>OP4A</u> |                      |        | T | D | B | S | T | <input type="checkbox"/> SEE PAGE 3.2<br><input type="checkbox"/> SPECIAL OUTPUT<br><input type="checkbox"/> PROCESSING INSTRUCTIONS |           |
|-------------------|----------------------|--------|---|---|---|---|---|--------------------------------------------------------------------------------------------------------------------------------------|-----------|
| PPN: <u>,14</u>   |                      |        | O | E | U | L | I |                                                                                                                                      |           |
|                   |                      |        | A | C | R | I | M |                                                                                                                                      |           |
|                   |                      |        | L | L | T | S |   |                                                                                                                                      |           |
|                   |                      |        | C | A |   |   | P |                                                                                                                                      |           |
|                   |                      |        | O | T |   |   | R |                                                                                                                                      |           |
|                   |                      |        | P | E |   |   | I |                                                                                                                                      |           |
|                   |                      |        | I |   |   |   | N |                                                                                                                                      |           |
|                   |                      |        | E |   |   |   | T |                                                                                                                                      |           |
|                   |                      |        | S |   |   |   | E |                                                                                                                                      |           |
| REPORT ID         | FILE NAME OR TAPE ID | FORM # |   |   |   |   |   | TITLE                                                                                                                                | MAIL STOP |
|                   |                      |        |   |   |   |   |   |                                                                                                                                      |           |
|                   |                      |        |   |   |   |   |   |                                                                                                                                      |           |
|                   |                      |        |   |   |   |   |   |                                                                                                                                      |           |
|                   |                      |        |   |   |   |   |   |                                                                                                                                      |           |
|                   |                      |        |   |   |   |   |   |                                                                                                                                      |           |

```

***** CSFS ***** CSFS ***** CSFS ***** NAV 09 ***** TMC ***** TMC ***** SYSTEM *****
001+CALLS/A + + 025+ADS01+501+ + 049+LJCU10+001+ + 073+UNLU93+CPM+ + 097+UNLU93+CPM+ + 121+DU001+9PM+ + J
002+SUSL71+ \ + 026+ADS01+025+ + 050+ 074+ 098+HMD20+ \ + + 122+MRU95+6PM+ + J
003+FUC131+ \ + 027+ 051+ 075+MSL01+073+ + 103+ 126+091J95+ X + + 123+MRU95+5PM+ + J
004+TRFL52+ \ + + 028+HMDU01+S26+ + 052+ 076+ 100+HMLA80+ \ + + 124+U77U95+ X + + J
005+ 029+ 053+ 077+ 101+ 125+SPU95+ X + + J
006+SUSL02+004+ + 030+ 054+PPSA24+001+ + 078+PAYA07+ SA+ + 102+MNTU20+C73+ + 126+091J95+ X + + J
007+ 031+ 055+PPSA25+ \ + + 079+PAYA03+ SA+ + 103+ 127+PSYU95+ X + + J
008+TRFL01+003+ + 032+ 056+PPSU01+ \ + + 080+PAYD80+073+ + 104+ 128+OMLU93+7AM+ + J
009+TRFL02+ \ + + 033+ 057+(C.1.+ \ + + 081+TUA430+ \ + + 105+ 129+MRU92+2AM+ + J
010+TRFL03+ \ + + 034+ 058+PSM03+ \ + + 082+ 106+ 130+KUSU01+1230+ + J
011+ 035+ 059+PAS01+ \ + + 083+MSA10+S73+ + 107+ 131+KUSU02+ \ + + J
012+MAGL01+004+ + 036+ 060+ 084+MSA20+ \ + + 108+IASU10+073+ + 132+L77M01+1AM+ + J
013+INVL01+ \ + + 037+ 061+ 085+MSA57+ \ + + 109+APSU01+S73+ + 133+OMLU97+12A+ + J
014+MAGL02+ \ + + 038+TRH001+014+ + 062+MARU00+ SA+ + 086+ 110+APS405+ X + + 134+LOGU01+1AM+ + J
015+FLC102+ \ + + 039+PUCLO3+016+ + 063+MARU01+ \ + + 087+MSA10+073+ + 111+ 135+TUSU01+ X + + J
016+MAGL03+ \ + + 040+ 064+ 088+MSA11+ \ + + 112+ 136+DKU09+230+ + J
017+PAGL04+ \ + + 041+ 065+ 089+CASA06+ \ + + 113+ 137+FICHE+ + AM+ + J
018+ 042+ 090+ 090+CASA20+ \ + + 114+ 138+L77M01+ X + + J
019+ 043+REF402+ E + + 067+ 091+CASA71+ \ + + 115+ 139+ 140+STADU01+ X + + J
020+STAL01+015+ + 044+TRFU96+ N + + 068+ 092+CASA78+ \ + + 116+ 141+FOCU01+ X + + J
021+STAL02+ \ + + 045+PSVU96+ T + + 069+ 093+ 117+077U07+136+ + 142+MAGU04+ X + + J
022+REFAC3+ \ + + 046+CRPU96+044+ + 070+RBAJ8+001+ + 094+ 118+091R07+ 45+ + 143+FOCDO2+ X + + J
023+MATAC1+004+ + 047+FUC198+045+ + 071+PBUA51+ \ + + 095+ 119+CUSU01+1230+ + 144+LUSU01+8AM+ + J
024+MAT403+ \ + + 048+S77R03+043+ + 072+PBUA51+ \ + + 096+ 120+CUSU02+ \ + + J
***** SCHEDULE NOTES *****
+001J FOCUS - 4 PRT LABS, 2 SET SAAS, 4 COPIES FACING SLIPS... 000J
+001J PLS 2 SETS OF THE PINK LABELS. 000J
+02J AUS SLCTS - 25,90. 000J
+03J 044/049J SLCTS - 10,17,22,24,38,55/26,43,49,59,64,72. 000J
+10J PLEASE HAVE ALL THE AFS IN DATA ENTRY AREA BY 7:30 A.M. 000J
+11J SLCTS - 44,49,117. 129J MRU92 - PLEASE USE SF #1. 000J

```

SPECIAL NOTE REPORT FOR MAY 09  
 -----

- ) 014) < >MAGC02: USE SE #1 UNLESS YOU RECEIVE A T R A N S M I T T A L .
  - ) 025) < >ALSA01: USES CRC3, & TRF, AND RUNS AFTER UNLU97.
  - ) 026) < >ACS#01: RUN AFTER AUSA01, CASA20, BUT BEFORE POID01.
  - ) 028) < >PCIG01: PUI MUST RUN AFTER ANY ACS#01 OR ADS#02 RUN.
  - ) 043) < >REF#02: \*\*\*\*\* RUN S77#03 \*\*\*\*\* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  - ) 049) < >PLCC10: MUST RUN BEFORE ANY OTHER LJC JOBS.
  - ) 056) < >PPSD01: RUNS EARLY ON THURS, AND PRINT REPORTS BEFORE ANY PPS#07 IS RUN.
  - ) 075) < >MSEL01: WHENEVER MEL#40 IS SCHEDULED MBS M U S T R U N \* B E F O R E \* I T . ! ! !
  - ) 078) < >PAY#07: IF PAY#21 IS SCHEDULED, RUN IT AFTER PAY#07.
  - ) 079) < >PAY#03: SPECIFY RUN EITHER FOR AN EDIT OR ELSE FOR AN UPDATE.
  - ) 080) < >PAY#08: THIS JOB GATHERS ALL THE LJC PAY ONLINE STUFF SO IT RUNS AFTER 001.
  - ) 083) < >RES#10: NON-CRITICAL JOB - HOLD TIL A.M. IF IT BOMBS - RUNS ON 0F4A.
  - ) 084) < >RES#20: \*\*\* IF GASM20 IS RUNNING THE SAME NIGHT THIS MUST RUN \*AFTER\* IT. \*\*\*
  - ) 085) < >RES#57: \*\*\* MUST RUN AFTER GASM20 FINAL, AND RES#20 IF SCHEDULED.
  - ) 087) < >MSB#10: RUN MSB#11 DIRECTLY AFTER THIS, AND RUN BOTH AFTER ALL CAS JOBS.
  - ) 089) < >CASA#06: RUN AFTER MSB#11 BUT BEFORE ANY OTHER CAS JOB.
  - ) 090) < >CASA#20: RUN AFTER CAS#06 WHENEVER SCHEDULED, ALWAYS RUN AFTER CASA20.
  - ) 091) < >CASA#71: RUN AFTER CASA#66.
  - ) 098) < >MEL#20: PLZ BE SURE TO GIVE OPERATIONS A TRANSMITTAL
  - ) 099) < >MEL#55: DUB PENNEMON OR LINDA COLLINS MUST SIGN THE TRANSMITTALS.
  - ) 106) < >IASD#10: ALWAYS RUN BEFORE ANY AFS#01 RUN.
  - ) 109) < >AFS#01: RUN AFTER CRT#10, & IASD#10, EXCEPT IF THEY BOMB.
- \*\*\* END OF REPORT \*\*\*





## TRANSMITTAL CHECK LIST FOR TONIGHTS SCHEDULE

REFXC3 < > - CHECK IF RECEIVED  
MKTXC1 < > - CHECK IF RECEIVED  
MKTXC3 < > - CHECK IF RECEIVED  
REFXC2 < > - CHECK IF RECEIVED  
PPSX24 < > - CHECK IF RECEIVED  
PPSX25 < > - CHECK IF RECEIVED  
PPSD01 < > - CHECK IF RECEIVED  
PPSW03 < > - CHECK IF RECEIVED  
PASW01 < > - CHECK IF RECEIVED  
PBUX38 < > - CHECK IF RECEIVED  
PBUX51 < > - CHECK IF RECEIVED  
PBUX51 < > - CHECK IF RECEIVED  
MBSD01 < > - CHECK IF RECEIVED  
PAYXC3 < > - CHECK IF RECEIVED  
TDAX30 < > - CHECK IF RECEIVED  
RESX10 < > - CHECK IF RECEIVED  
RESX20 < > - CHECK IF RECEIVED  
RESM57 < > - CHECK IF RECEIVED  
MSBX10 < > - CHECK IF RECEIVED  
CASX06 < > - CHECK IF RECEIVED  
CASX20 < > - CHECK IF RECEIVED  
CASX78 < > - CHECK IF RECEIVED  
MBLW55 < > - CHECK IF RECEIVED  
MBLX60 < > - CHECK IF RECEIVED  
APSX05 < > - CHECK IF RECEIVED

25 TRANSMITTALS COUNTED FOR THIS SCHEDULE.

THE FOLLOWING COE JOBS ARE NEEDED FOR  
1CNIGHTS SCHEDULE:

ADSX01 < > CHECK IF FILE READY TO RELEASE  
POID01 < > CHECK IF FILE READY TO RELEASE  
MARD00 < > CHECK IF FILE READY TO RELEASE  
PAYX07 < > CHECK IF FILE READY TO RELEASE  
PAYX03 < > CHECK IF FILE READY TO RELEASE  
RESX10 < > CHECK IF FILE READY TO RELEASE  
APSD01 < > CHECK IF FILE READY TO RELEASE

---- END OF REPORT -----

05/08/1985

PBU

| MAG | 116.14 | # | RM # | JOB | BURSTING_ROOM | DELIVER_TO | REPORT_NAME |
|-----|--------|---|------|-----|---------------|------------|-------------|
|-----|--------|---|------|-----|---------------|------------|-------------|

|    |            |    |   |      |        |     |                       |            |
|----|------------|----|---|------|--------|-----|-----------------------|------------|
| -- | PBUR50.X38 | -- | 2 | 6722 | PBUX38 | DEC | P300 CSPS BUDGET DIV. | PBUR50.X38 |
|----|------------|----|---|------|--------|-----|-----------------------|------------|

\*\*\*\*\*LAST REVISED: 03/18/

|    |            |    |   |      |        |          |                           |         |
|----|------------|----|---|------|--------|----------|---------------------------|---------|
| -- | PBUX53.EXP | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIVISION | EXPENSE |
|    |            |    |   |      | ***    | INCLUDES | 81,82,83,84               |         |

|    |            |    |   |      |        |          |                       |        |
|----|------------|----|---|------|--------|----------|-----------------------|--------|
| -- | PBUX53.INC | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIV. | INCOME |
|    |            |    |   |      | ***    | INCLUDES | 81,82,83,84           |        |

|    |            |    |   |      |        |          |                           |         |
|----|------------|----|---|------|--------|----------|---------------------------|---------|
| -- | PBUX5E.ACT | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIVISION | EXPENSE |
|    |            |    |   |      | ***    | INCLUDES | REPORTS 91,92,93,94       |         |

|    |            |    |   |      |        |          |                           |                |
|----|------------|----|---|------|--------|----------|---------------------------|----------------|
| -- | PBUX5E.CST | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIVISION | EXPENSE REPORT |
|    |            |    |   |      | ***    | INCLUDES | 51,52,53,54               |                |

|    |            |    |   |      |        |          |                           |                |
|----|------------|----|---|------|--------|----------|---------------------------|----------------|
| -- | PBUX5E.DIV | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIVISION | EXPENSE REPORT |
|    |            |    |   |      | ***    | INCLUDES | 61,62,63,64               |                |

|    |            |    |   |      |        |          |                           |                |
|----|------------|----|---|------|--------|----------|---------------------------|----------------|
| -- | PBUX5E.DPT | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIVISION | EXPENSE REPORT |
|    |            |    |   |      | ***    | INCLUDES | 71,72,73,74               |                |

|    |            |    |   |      |        |          |                                   |  |
|----|------------|----|---|------|--------|----------|-----------------------------------|--|
| -- | PBUX5E.GRP | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIVISION         |  |
|    |            |    |   |      | ***    | INCLUDES | PBUR01, PBUR02, PBUR03 AND PBUR04 |  |

|    |            |    |   |      |        |          |                       |        |
|----|------------|----|---|------|--------|----------|-----------------------|--------|
| -- | PBUX5I.ACT | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIV. | INCOME |
|    |            |    |   |      | ***    | INCLUDES | 91,92,93,94           |        |

|    |            |    |   |      |        |          |                           |        |
|----|------------|----|---|------|--------|----------|---------------------------|--------|
| -- | PBUX5I.CST | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIVISION | INCOME |
|    |            |    |   |      | ***    | INCLUDES | 51,52,53,54               |        |

|    |            |    |   |      |        |          |                       |        |
|----|------------|----|---|------|--------|----------|-----------------------|--------|
| -- | PBUX5I.DIV | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIV. | INCOME |
|    |            |    |   |      | ***    | INCLUDES | 61,62,63,64           |        |

|    |            |    |   |      |        |          |                       |        |
|----|------------|----|---|------|--------|----------|-----------------------|--------|
| -- | PBUX5I.DPT | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIV. | INCOME |
|    |            |    |   |      | ***    | INCLUDES | 71,72,73,74           |        |

|    |            |    |   |      |        |          |                                   |  |
|----|------------|----|---|------|--------|----------|-----------------------------------|--|
| -- | PBUX5I.GRP | -- | 2 | 6722 | PBUX51 | DEC      | P300 CSPS BUDGET DIVISION         |  |
|    |            |    |   |      | ***    | INCLUDES | PBUR01, PBUR02, PBUR03 AND PBUR04 |  |

\*\*\*\*\*LAST REVISED: 03/09/

**PRODUCTION CONTROL**  
PPP 333 0000 4 4  
P P 3 0 0 4 4  
PPP 333 0 0 4444  
P 3 0 0 4  
P 333 0000 4  
**FOC & MAGDOO**  
**FROM I.S. DATA PROCESSING DIV.**

**SEND TO:**  
**CONTROLLERS**  
  
P300  
  
**FOC & MAGDOO**  
**FROM I.S. DATA PROCESSING DIV.**

THE FIRST CHURCH OF CHRIST, SCIENTIST  
CHRISTIAN SCIENCE CENTER  
BOSTON, MASSACHUSETTS, U.S.A. 02115  
**SEND TO:**  
  
**CSPS BUDGET DIVISION**  
  
P300  
  
PBUX38  
  
**FROM I.S. DATA PROCESSING DIV.**

THE FIRST CHURCH OF CHRIST, SCIENTIST  
CHRISTIAN SCIENCE CENTER  
BOSTON, MASSACHUSETTS, U.S.A. 02115  
**SEND TO:**  
  
**CSPS BUDGET DIVISION**  
  
P300  
  
PBUX51  
  
**FROM I.S. DATA PROCESSING DIV.**

## JOB ANALYSIS REPORT

| JOB NAME     | TOTAL RUN TIME     | DATE TAKEN<br>OR AVERAGE | DATE NOTE<br>ENTERED | NOTES                                                                                                |
|--------------|--------------------|--------------------------|----------------------|------------------------------------------------------------------------------------------------------|
| ***HARD01*** | 3 HOURS 15 MINUTES | 4-30-85                  | 5-01-85              | THIS TIME WAS TAKEN FOR HARD01 MONTH END AGAINST A FAIRLY HEAVY JOBSTREAM LOAD - POI 505 TRF         |
| ***TRFD01*** | 5 HO 15 05 MINUTES | AVERAGE                  | 4-22-85              | THIS IS AN AVERAGE RUN TIME FOR APRIL.<br>TIMES VARY FROM 4.5 TO 5.75 HOURS - CREATING 4 LABEL TAPES |

## M.I.S. COMPUTER OPERATIONS DIVISION

## SHIFT REPORT

DATE: 05-08-85

SHIFT: FIRST

OPERATOR ATTENDANCE: PRESENT: RAY, ED, RUSS, WES, VALERIE O/T

ABSENT: NONE

| JOB-NAME | TERMINATION | JOB-SET-UP | HARD-SOFT-WARE | PROBLEM DESCRIPTION                                                                                                                                                                                                        |
|----------|-------------|------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RESA20   | REW/HOLD    |            |                | BOMBED IN STEP 20... "NO RECORDS ON FILE" MESSAGE DURING THE RES20 PROGRAM... CALLED MEL METS... HELD TILL THE CLIENT LOOKS AT THE RES10 REPORT TOMORROW...                                                                |
| MNT#02   | COR/NORM    |            |                | BOMBED IN STEP 990... THERE WASN'T ANY MNTF03.BLK FILE IN THE MBR3:114,14J AREA... CALLED MEL METS... RENAMED THE MNTF03, MNTF10 AND MNTF16 IDA'S AND IDJ'S TO BLA & BLX PER MEL... RESTARTED FROM ST990 TO NORMAL TERM... |
| PPSA24   | COR/NORM    |            |                | BOMBED IN STEP 20... UNLOAD TAPE PARITY ERROR... RESTARTED TO NORMAL TERM...                                                                                                                                               |

## CRASHES:

| REASON | CRASH-NAME | SAVE-TAPE | COMMENTS OR DESCRIPTION                                                |
|--------|------------|-----------|------------------------------------------------------------------------|
| OTHER  | NONE       | NONE      | 1077 CRASHED AT 1716... EVE STOPCODE... THE SYSTEM CONTINUED ITSELF... |

## HARDWARE DEVICE PROBLEMS:

D#87 1077 FRONT END WENT DOWN AND THEN CONTINUED ITSELF AT 1705..

## COMMENTS:

PASK70 TOTALS CHECK...  
 \*\*\*\*\*  
 FICHE STAM11 HAS BEEN RESUBMITTED FOR THE MASTERS ONLY AS THE CLIENT ONLY RECEIVED THE COPIES...  
 \*\*\*\*\*  
 3700, 37104, 0722, 07251 FORMS REJECTED...  
 \*\*\*\*\*  
 CONFIDENTIAL TRANS RUN COMPLETE...  
 \*\*\*\*\*  
 BOB KCHNEK FROM MATERIAL SERVICES PICKED UP THE BOX

Information Systems - Operations

Computer Systems Development

Page 17 of 2  
Request No.: \_\_\_\_\_

Date Desired \_\_\_\_\_

SOFTWARE  
PRODUCTION JOB PERFORMANCE MEMO

To: \_\_\_\_\_ Approved \_\_\_\_\_

From: \_\_\_\_\_ Date \_\_\_\_\_

PPN: \_\_\_\_\_ Run Date: \_\_\_\_\_ Run Time: \_\_\_\_\_

Control File: \_\_\_\_\_ Program: \_\_\_\_\_

Problem Encountered

Comments and Questions

Attachments

Listing

Log

RM 67.142B

Data Processing

SEND TO: Manager, Information Systems A-31

DATE: \_\_\_\_\_

CLIENT: \_\_\_\_\_

REPORTED BY: \_\_\_\_\_

SYSTEM: \_\_\_\_\_

REPORTS INVOLVED: \_\_\_\_\_

CONDITION: [Please check box(es) that apply]

Late Delivery

Scheduled Delivery Date: \_\_\_\_\_ Time: \_\_\_\_\_

Actual Delivery Date: \_\_\_\_\_ Time: \_\_\_\_\_

Incomplete Delivery

What was missing: \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

Inaccuracy

Explanation: \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

Other

Explanation: \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

FURTHER INFORMATION:

- Was advance warning of the condition given? \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

- Was there anything unusual about this run?

Volume of Processing: \_\_\_\_\_

Complexity: \_\_\_\_\_

Other: \_\_\_\_\_

- Was input delivered to Information Systems on schedule? \_\_\_\_\_

If not, date/time due: \_\_\_\_\_

date/time delivered to Information Systems: \_\_\_\_\_

- How would you rate the effect of this condition on your operations? [Please check box(es) that apply]

- Significant...Caused us major problems
- Caused problems but we could adjust
- An inconvenience but not serious
- Little or no problem caused
- The "better-than-expected" performance helped us.



MEMORANDUM

DATE \_\_\_\_\_

TO \_\_\_\_\_  
\_\_\_\_\_

FROM Information Systems

RE: Your Data Processing Quality Control Report

Thank you for your report. It helps us to continually monitor our performance and to make improvements when they are needed. The following is provided for your information.

DATE OF REPORT: \_\_\_\_\_ DEPARTMENT CONTROL No: \_\_\_\_\_

CONDITION REPORTED: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

WHAT CAUSED IT: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

WHAT CAN BE DONE: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

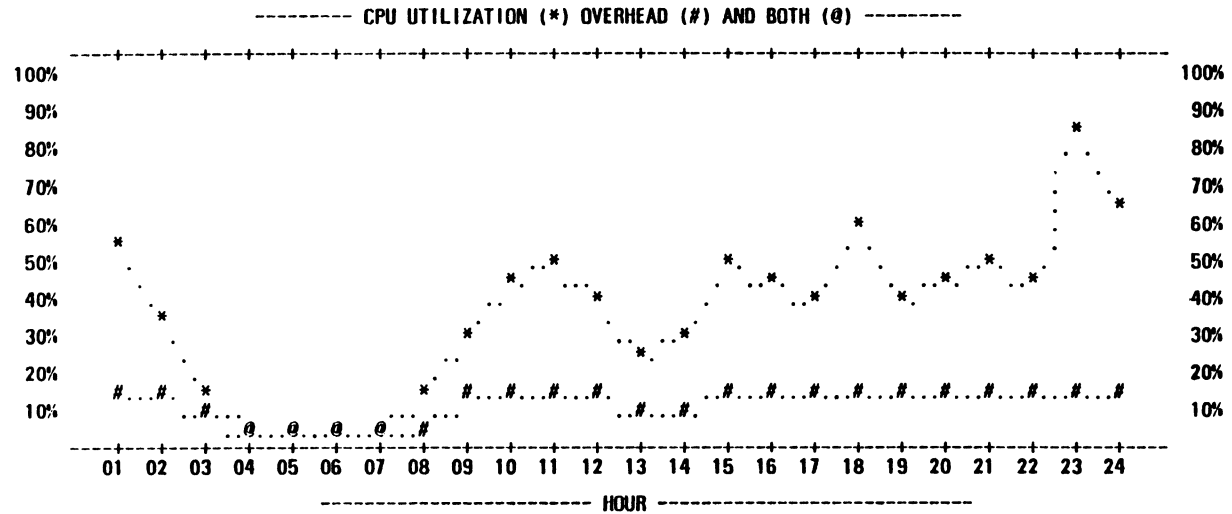
COMMENTS: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Information Systems

DATE: 06-JUL-84 (FRIDAY)

- AMAR -  
DAILY SYSTEM UTILIZATION SUMMARY REPORT

TFCCS - 1091  
SYSTEM: FCS2 PRIME TIME: 0900 - 1200, 1300 - 1600



----- SUMMARY OF KEY UTILIZATION ITEMS -----

| -----AVERAGE-----    | %AMAR TI SMP UTIL | % CPU UTIL       | %AMAR TI SMP OVHD | % OVHD TIME | %AMAR TI SMP LOST | % LOST TIME | AVG SCHED RSP TIME | ACT SWAP RATIO | SWAPPING BLKS/SEC | USER DSK BLKS/SEC | NO. OF HOURS KEY ITEMS OVER LIMITS |
|----------------------|-------------------|------------------|-------------------|-------------|-------------------|-------------|--------------------|----------------|-------------------|-------------------|------------------------------------|
| ---PRIME TIME---     | 42                |                  | 13                | 13          | 6                 | 6           | 6                  | .2             | 90                | 26                | 4                                  |
| ---NON-PRIME TIME--- | 33                |                  | 9                 | 9           | 0                 | 0           | 2                  | .1             | 4                 | 86                | 4                                  |
| -----AVERAGE-----    | USER UUDS/SEC     | # JOBS LOGGED IN | # LINES IN USE    |             |                   |             |                    |                |                   |                   | NO. OF HOURS KEY ITEMS OVER LIMITS |
| ---PRIME TIME---     | 162               | 58               | 43                |             |                   |             |                    |                |                   |                   |                                    |
| ---NON-PRIME TIME--- | 110               | 23               | 9                 |             |                   |             |                    |                |                   |                   |                                    |

+ = MORE THAN YESTERDAY      - = LESS THAN YESTERDAY

----- CONTINUED NEXT PAGE -----

## Operator's Manual--Table of Contents

### Preface

1. General Concepts
  2. Operating System
  3. Timesharing
  4. Communication Hardware
  5. 1077 Hardware
  6. 1091 Hardware
  7. System Startup
  8. Terminal Usage
  9. GALAXY Batch System
  10. System Programs
  11. Utility Programs
  12. Monitor Commands
  13. Software Concepts
  14. Environment
    - 14.1 Air Conditioning
    - 14.2 Humidity
    - 14.3 Air Plenum Guage
    - 14.4 Voltage Regulation
    - 14.5 Cleanliness
    - 14.6 Physical Security
  15. Tape Library System
  16. The SLC-II
- Appendix A ONCE Dialog
- Appendix B 1091 Console Front-end Programs
- Glossary
- Index

Session Chair: Barbara Bersack

Session Speaker: Susan Porada  
Manager, Marlboro Software Publications  
Large Systems Group  
Digital Equipment Corporation

ABSTRACT

The Manager of Software Publications presented the status of documentation for TOPS-10 / TOPS-20 operating systems and layered products and described the Integration Documentation.

There will be continued support for TOPS-10 and TOPS-20 Documentation Products as well as Integration and Conversion Documents and customer feedback is extremely helpful. The TOPS-10 and TOPS-20 Documentation Products fall into four categories: Operating Systems, Layered Unbundled Products, Corporate Communication Products, and Software Notebook Sets. The Integration and Conversion documents also cover four areas: Language Compatibility and Transportability, Networks and Data Management, Operating Systems and Utilities, and System Management and Operations.

TOPS-10/TOPS-20 Documentation

The following is a list of TOPS-10/TOPS-20 revised or new documentation, presented by area, and a list of the work planned for the future.

\* TOPS-10 / TOPS-20 Operating Systems

1. Documentation has been updated for the following products: TOPS-10 DDT V42A, DIL V2, Traffic-20 V4, and LINK-20 V6.
2. EDT-20 Version 1. Documentation includes EDT Quick Reference Guide, EDT-20 Primer, and the EDT-20 Reference Manual.

\* TOPS-10 / TOPS-20 Layered Products

1. DBMS-20 Version 6.1. Documentation includes the Documentation Directory, User's Guide and Utilities Manual, DML and DDL Language Manuals, DBCS Error Message Manual, Installation Guide, three reference guides and poster. The documentation kit is QT008-GZ.
2. Datatrieve-20 Version 1. Documentation includes Introduction to Datatrieve-20, User's Guide, Reference Manual, Guide to Using Graphics, and Guide to Writing Reports. The documentation kit is QT371-GZ.
3. Fortran-10/20 Version 10. Documentation includes the Language Manual, Pocket Guide, Installation Guide, and Compatibility Manual. This is scheduled to be released this summer.

\* Corporate Communication Products

1. PSI-10 Version 1. Documentation includes an Installation Guide, a System Manager's Guide, and a User's Guide. The Documentation Kit is QH228-GZ.
2. DECnet/SNA TOPS-20 Version 1.

\* Software Notebooks

1. Update Number 93 of the TOPS-10 Software Notebooks is shipping now. It includes DDT V42A and DIL Version 2.
2. Update Number 23 of the TOPS-20 Software Notebooks is shipping now. It has documentation for EDT-20 V1 and DIL V2.

\* Future--work in Progress

1. TOPS-10 V7.03 and TOPS-20 V6.1
2. DECnet V4.
3. DIU V1, RMS-20 V3.
4. TOPS-10 Software Notebook Update 94 will contain PSI-10 and FORTRAN documentation.
5. TOPS-20 Update 24 will contain SNA-20 and FORTAN documentation.

Integration Documentation

In addition to the TOPS-10 and TOPS-20 documentation work, attention has been focused on Integration Documentation for multisystem sites. The following is the list of documentation already available in each of the 4 key areas as well as work planned for the future.

\* Language Compatibility/Transportability

1. FORTRAN
2. APL
3. PASCAL
4. COBOL-20
5. Language Fundamentals

- \* Networks, Communications, Data Base Management
  1. DECnet
  2. DIL
  3. FTS
  4. NFT
- \* System Management and Operations
  1. SPEAR
- \* Operating Systems and Utilities
  1. Commands Comparison (10/20)
  2. Monitor Calls Comparison (10/20)
  3. Digital Dictionary
- \* Future-In Progress
  1. COBOL-10/VAX COBOL
  2. TOPS-10/VMS Commands
  3. TOPS-20/VMS Commands
  4. TOPS-10/20/VMS Utilities
  5. Cross-System Text Editors
  6. TOPS-10/20/VMS Operators
  7. Networking User's Guide.

Integration Documentation Kits, which contain all Integration documents, are available. They are being distributed to TOPS-10 and TOPS-20 sites. The order number for TOPS-10 is QH308-GZ. It includes Language Fundamentals, DIL, TECO Pocket Guide, Networking Pocket Guide, FORTAN Compatibility and Network Compatibility. The TOPS-20 order number is QT650-GZ. That kit includes EDT Quick Guide, FTS-20, Language Fundamentals, DIL, Networking Pocket Guide, Language Compatibility, and Network Compatibility.

## TOPS/VMS Performance Comparisons

Session Chair: Barbara Bersack

Session Speakers: Dr. Thomas P. Blinn  
Principal Technical Support Specialist  
Large Systems Marketing  
Digital Equipment Corporation

Mr. Dan Kazzaz  
Independent Consultant

### ABSTRACT

ABSTRACT Representatives from DEC's Large System Marketing reported on the results of a variety of benchmark tests between the VAX (780, 785, 8600) and the KL (2065). Dan Kazzaz made some general introductory remarks; Dr. Blinn then reviewed the results of the tests which he ran.

For the past six months, Dan Kazzaz has worked with LSM to help them develop benchmark information. Mr. Kazzaz emphasized that the VAX and KL are different machines. TOPS supports paged I/O and uses dynamic balancing; VMS uses block I/O and manually selectable working set size. By developing benchmarks which rely on certain features, results can vary between machines. A SORT on the 8600 was twice as fast as the 2065 SORT. It is not clear if the difference is due to the different algorithms used or to the machines. Mr. Kazzaz indicated that most applications will run faster on the 8600 than the 2065; however he concluded that choosing a system is based on preference almost as much as on performance.

Dr. Blinn had done more extensive benchmarks. All comparisons were done using the same source programs. The configurations were:

KL:

- \* 2065 with 2 MW of memory
- \* TOPS-20 Operating System, V5.4 (Field Image)

VAX:

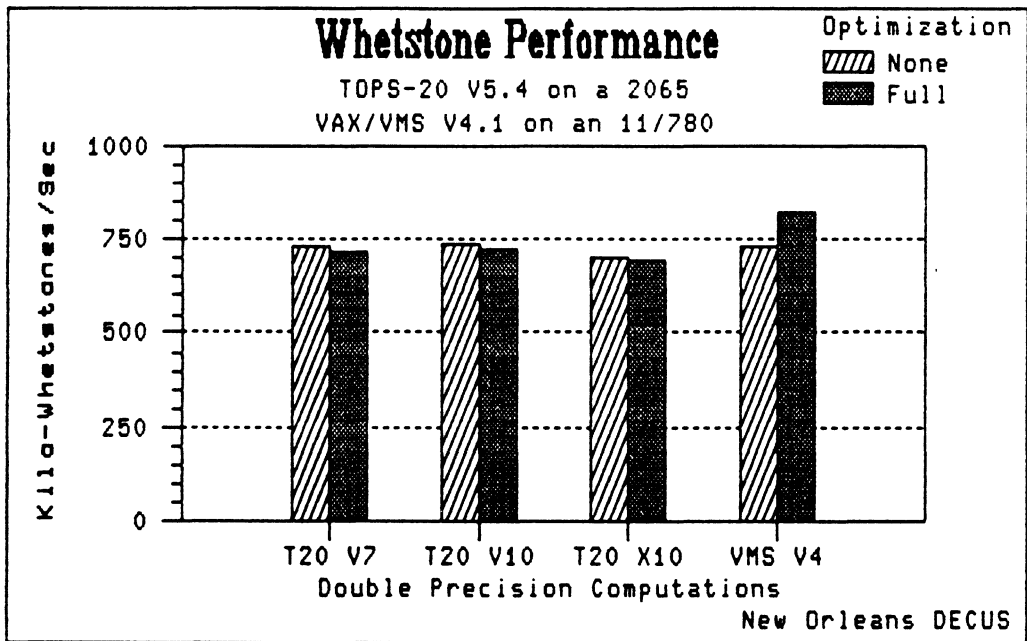
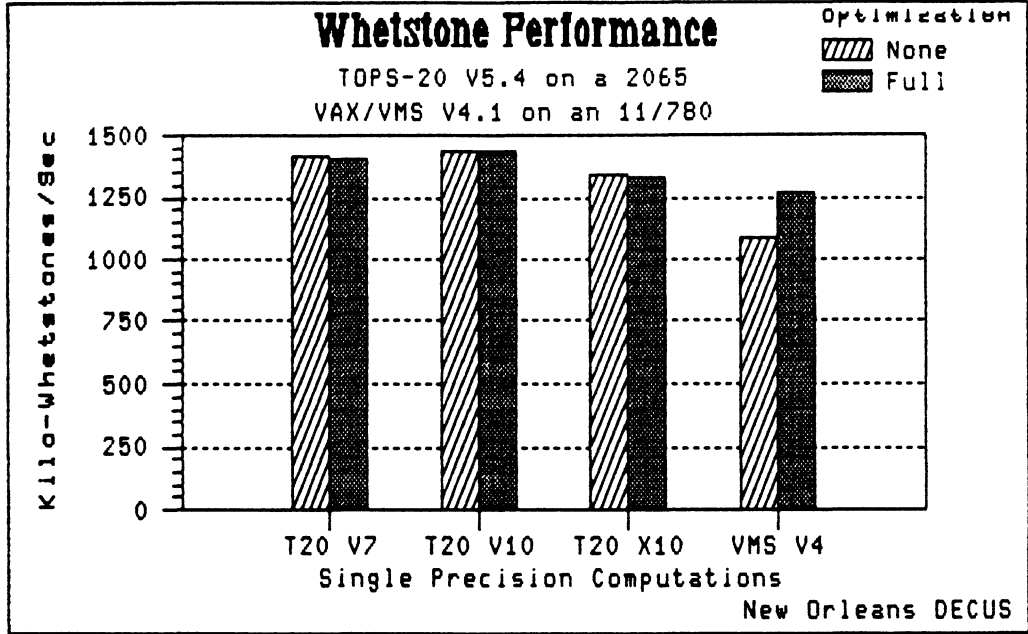
- \* 780/785/8600 with FPA and 32 MB of memory \*
- \* VAX VMS Operating System V4.1 (Field Image)

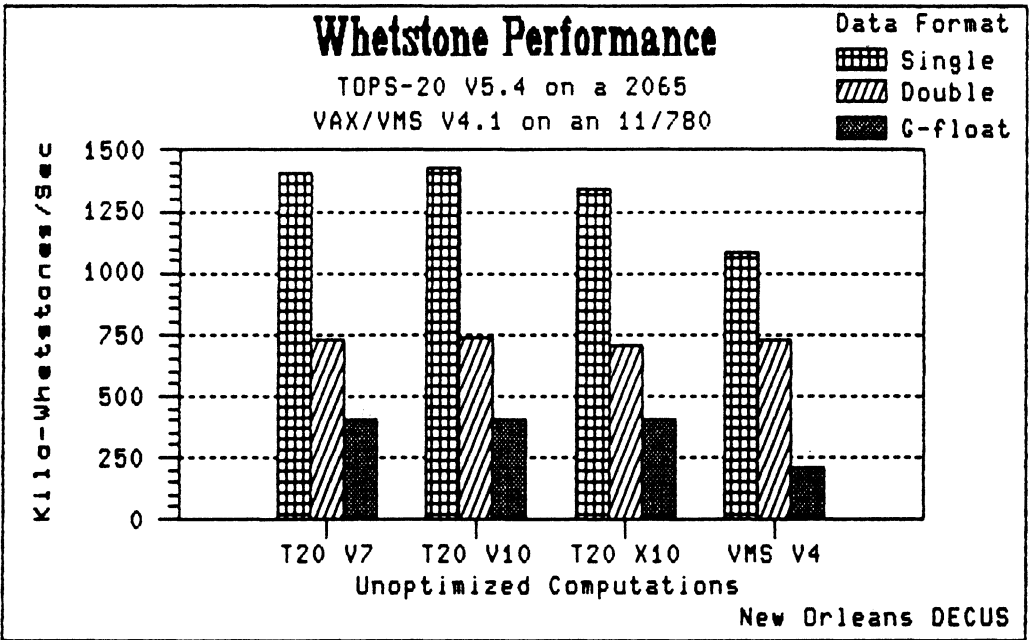
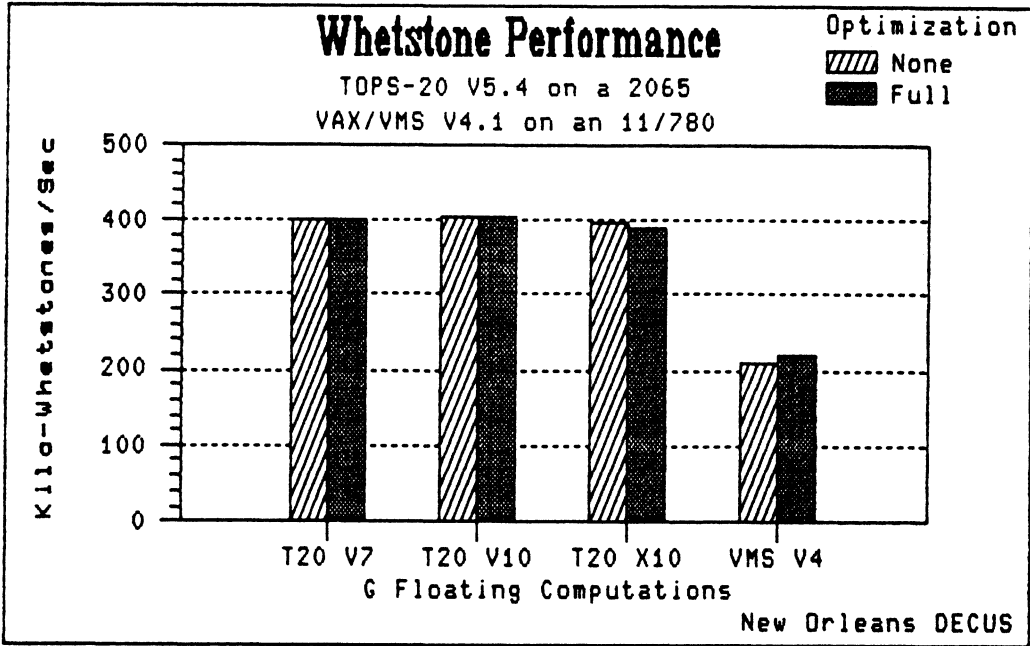
Most of Dr. Blinn's tests were done on the 780 and the 2065. He then incorporated other's results for the 785 and 8600.

The first set of tests were based primarily on FORTRAN. Tests included the Whetstone benchmark (Single Precision, Double Precision, and Gfloat). FORTRAN V10 and V7 were tested with and without optimization. All tests showed the 8600 results exceeding the 2065 by a factor of between 2 (considered the average) to as much as 13 in at least one case. However, it is important to take into consideration the difference between the two compilers since the VMS FORTRAN compiler optimizes code better than its TOPS-20 counterpart. The 2065 results were generally about the same as those for the 780, with some times better and some, worse.

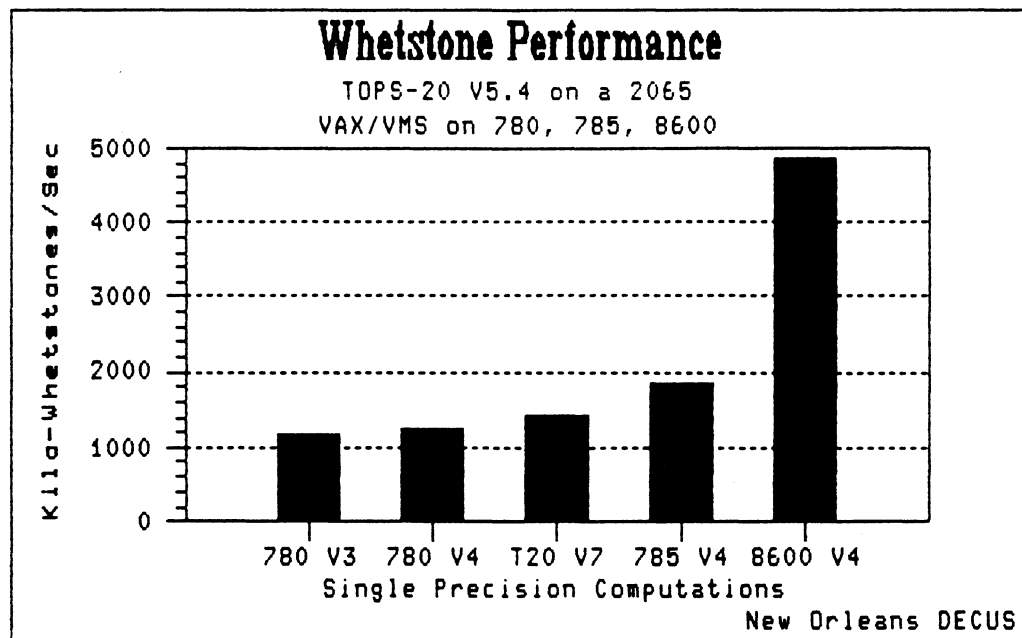
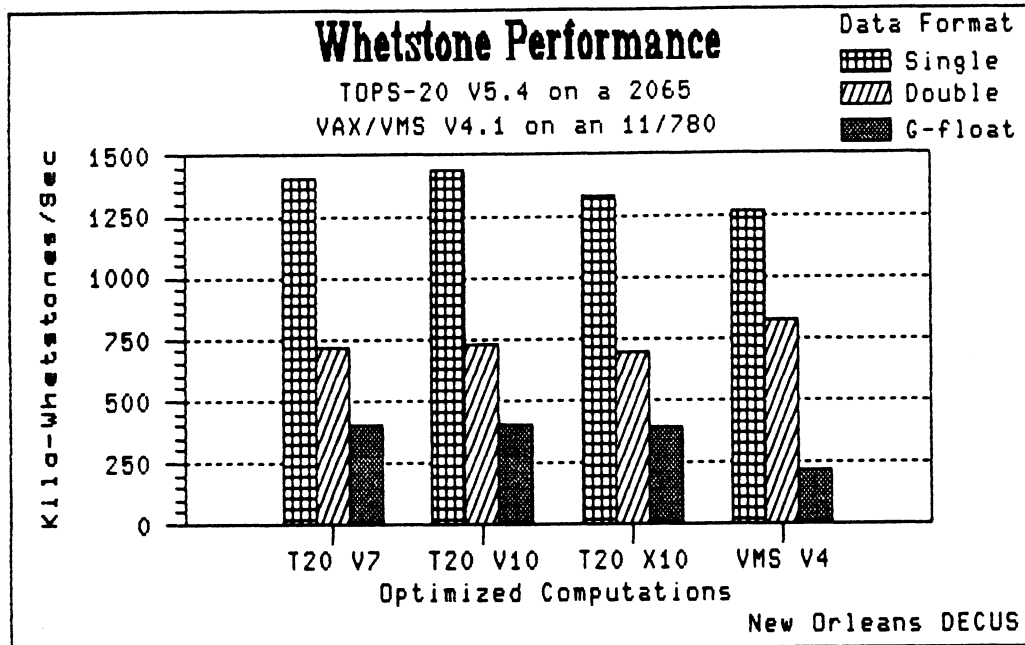
Some COBOL benchmarks were run between TOPS-20 V13 and VMS COBOL V3.1. IO was tested using the IO Devil benchmark. ECS Anker was the basis of the multi user benchmark. This benchmark on the KL produced acceptable response times which ranged from 1/4 second to 1 1/2 seconds at 90 users. The 8600's response curve was flat out to 90 users; the maximum response time was 1 second and occurred at 200 users.

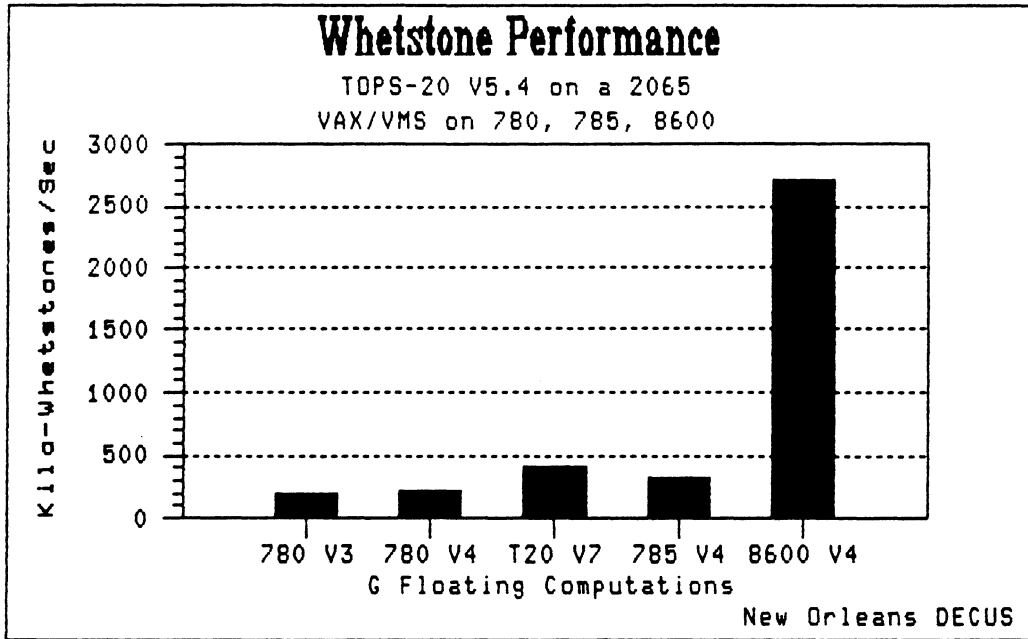
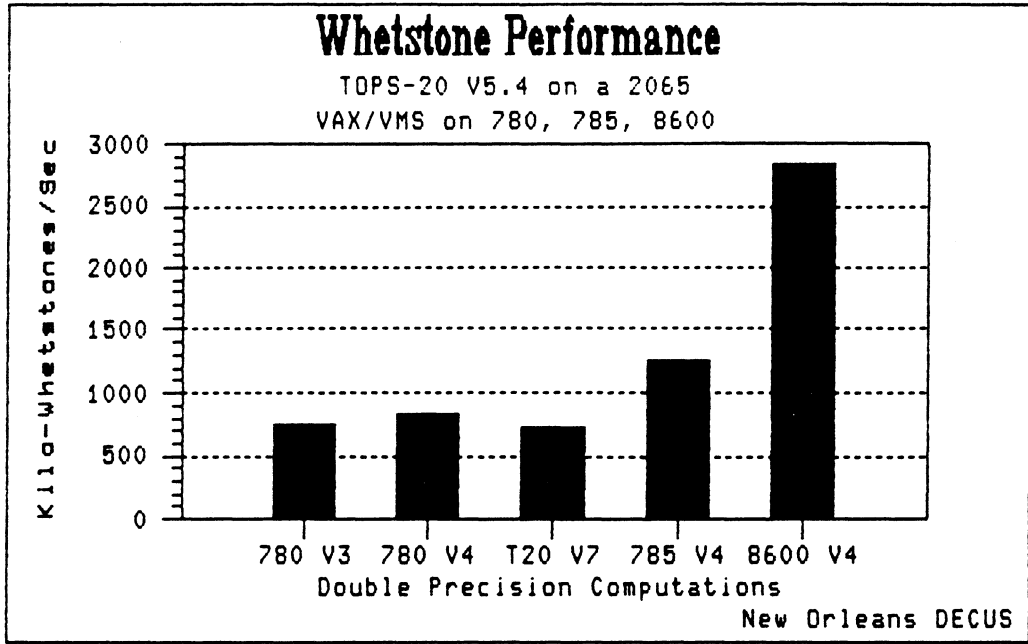
In summary, there is no simple answer to how one quantifies the difference between an 8600 and a KL. Tuning is much more complex and much more important under VMS than under TOPS. There is still a need for better multi-user benchmarks and Dr. Blinn invited users and third party vendors to submit their results at future symposia and to consider making arrangements to run their benchmarks at DEC's Benchmarking Center.

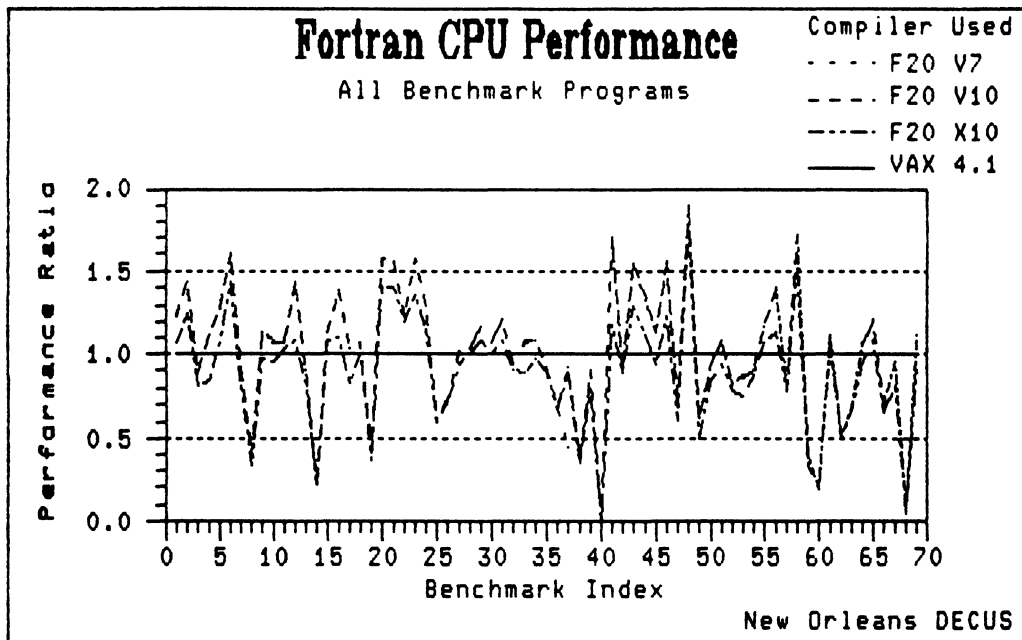
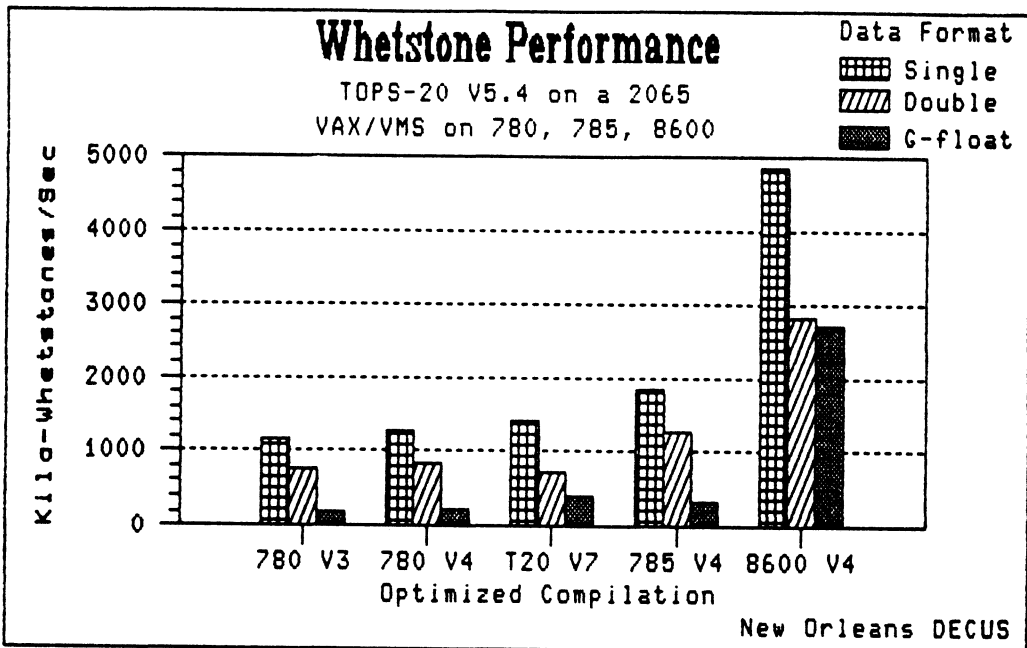


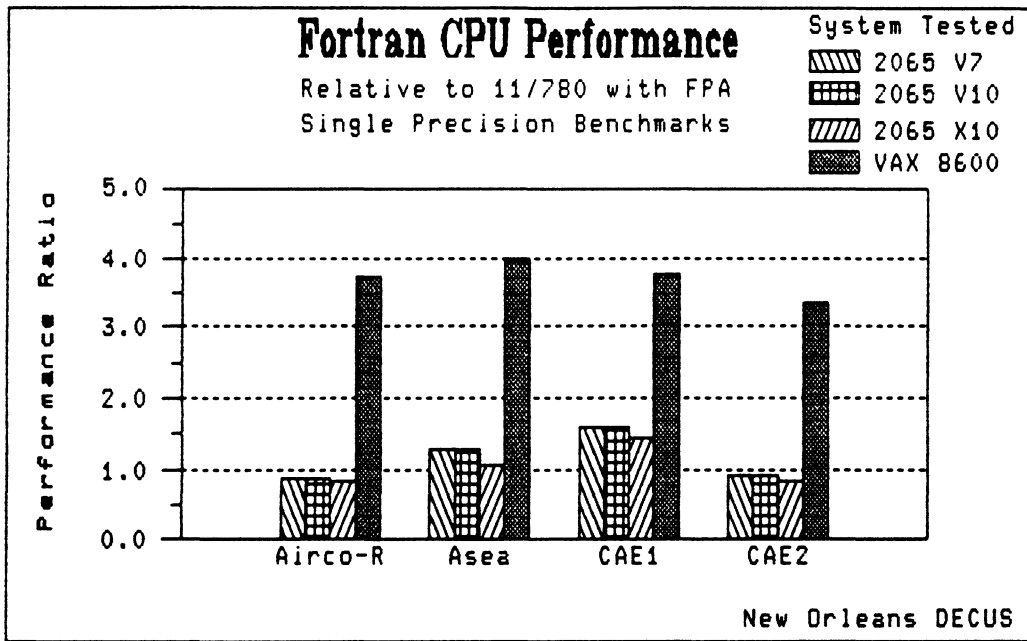
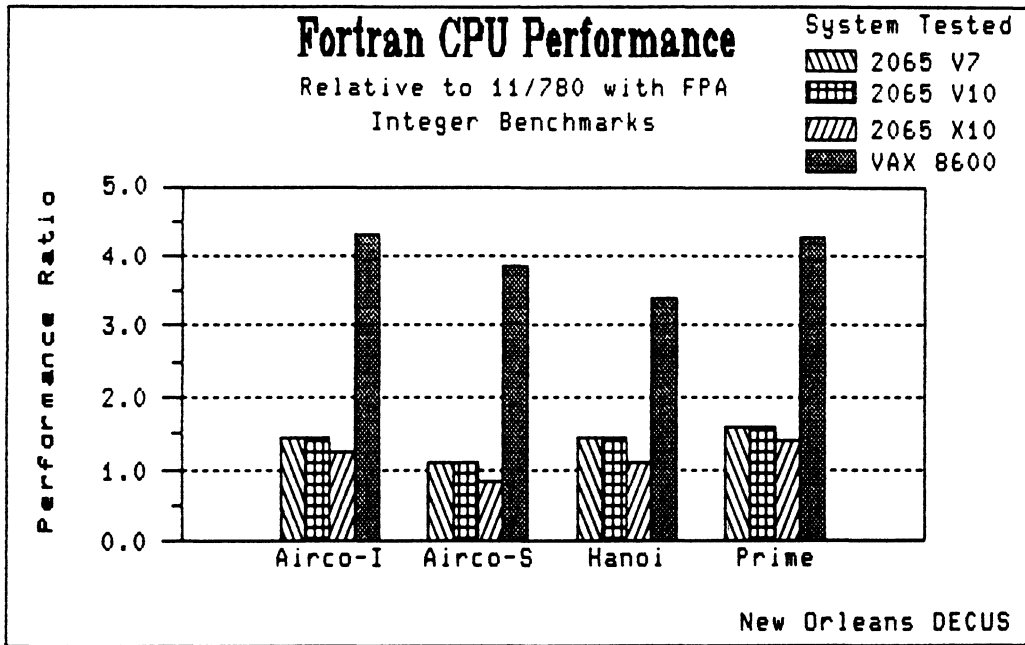


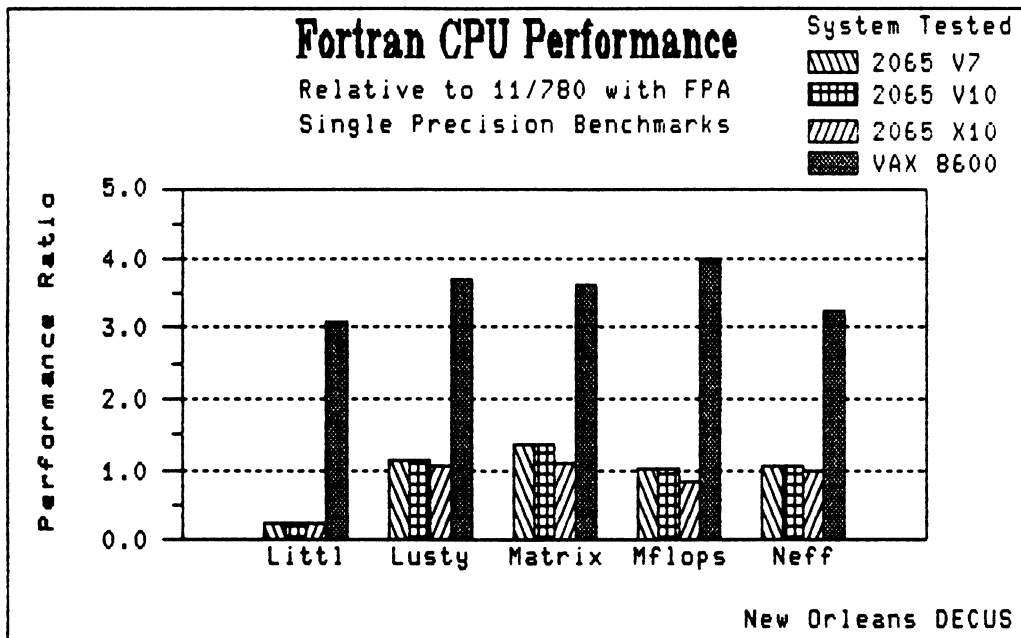
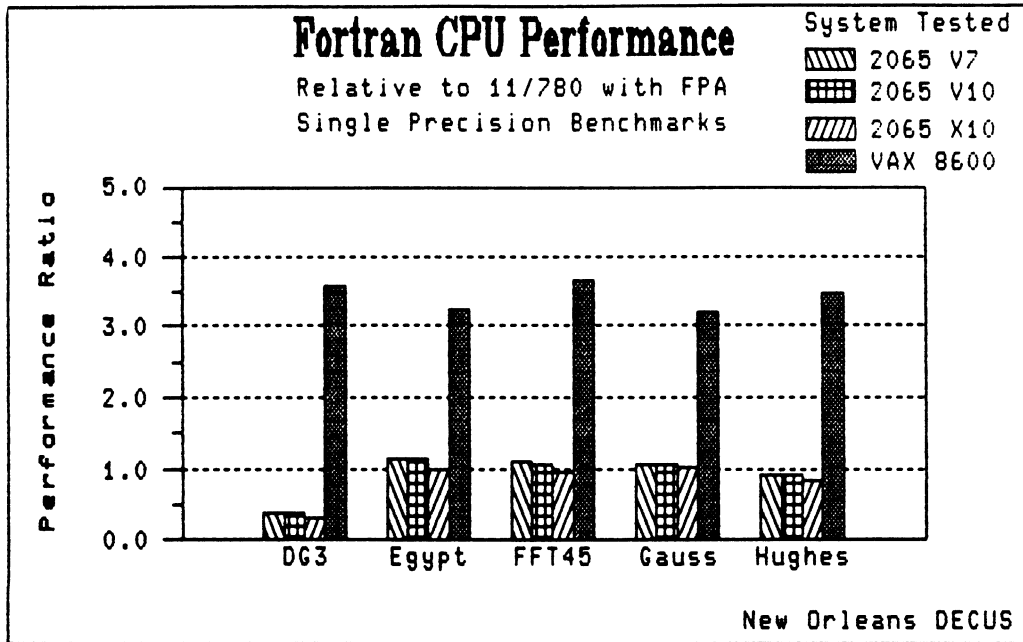


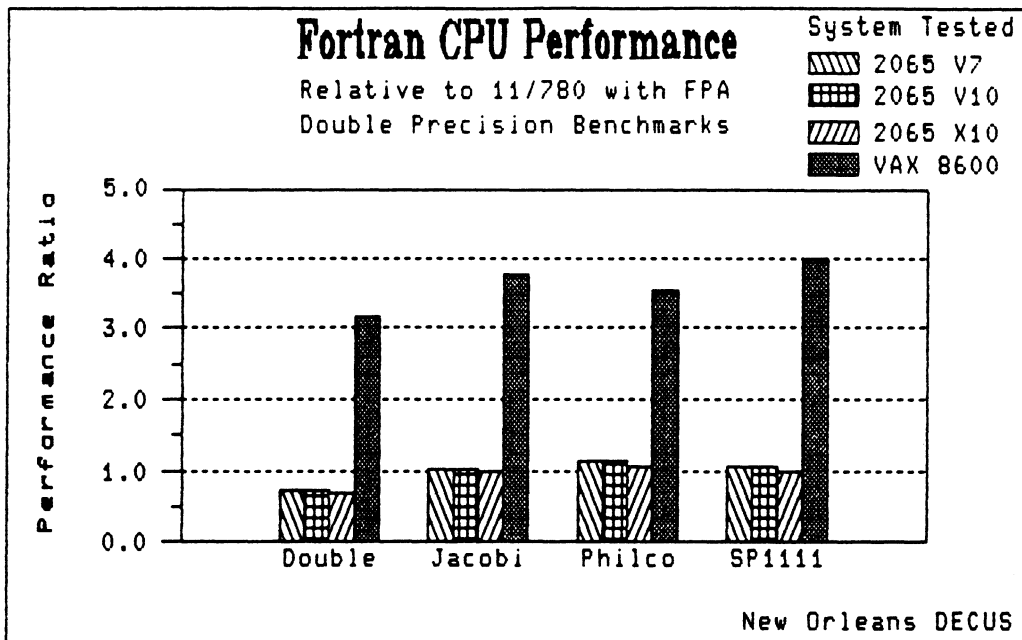
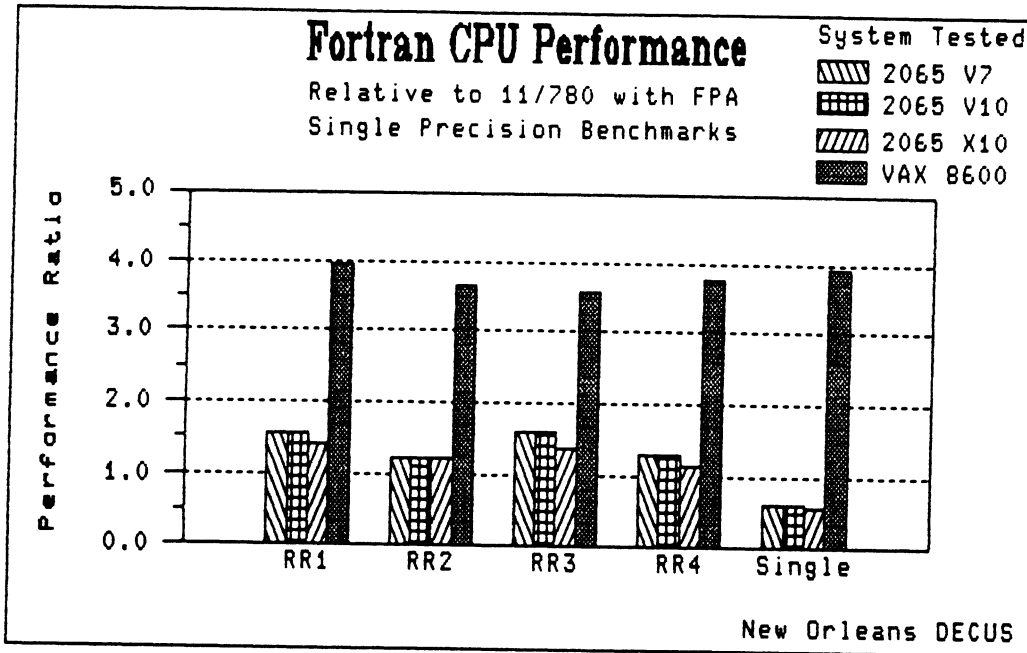


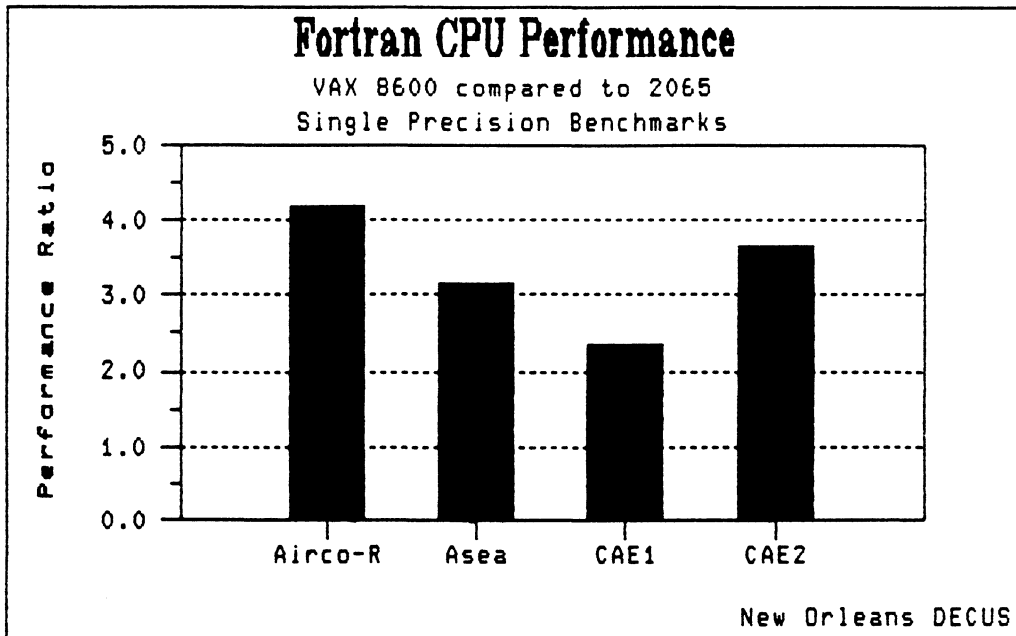
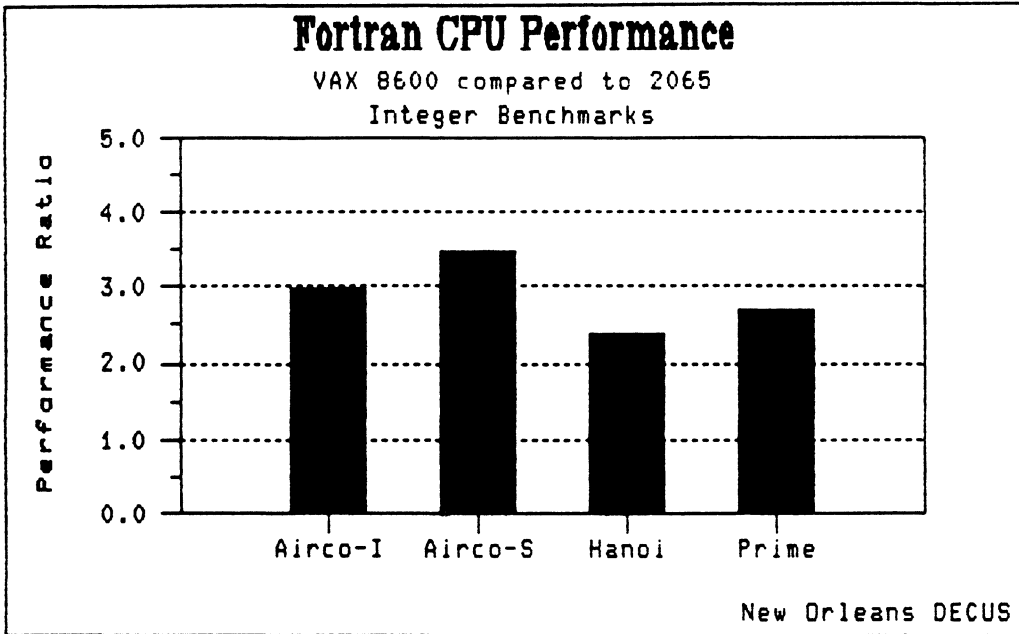


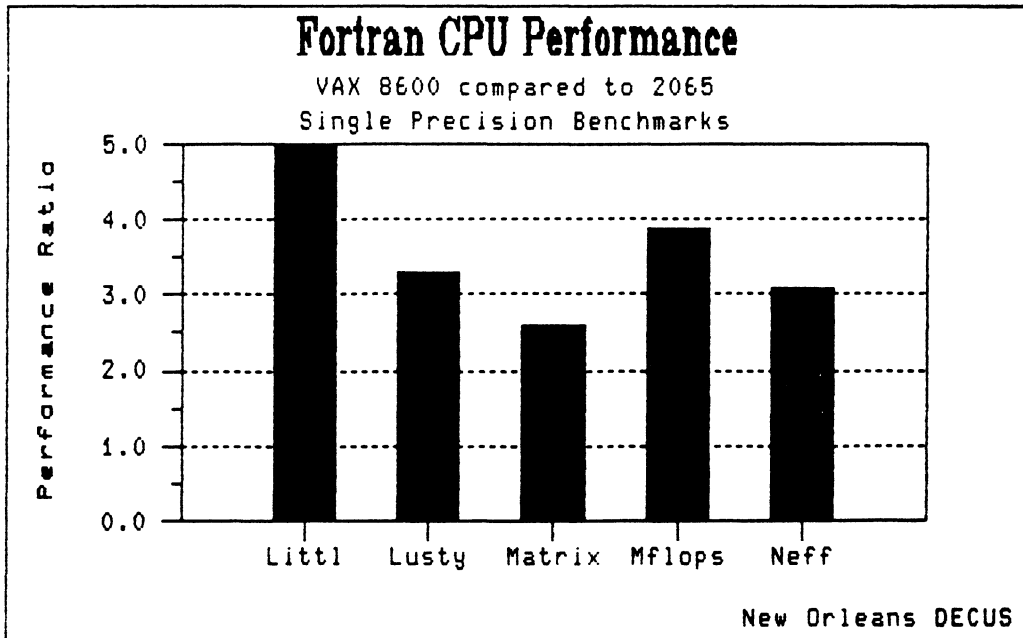
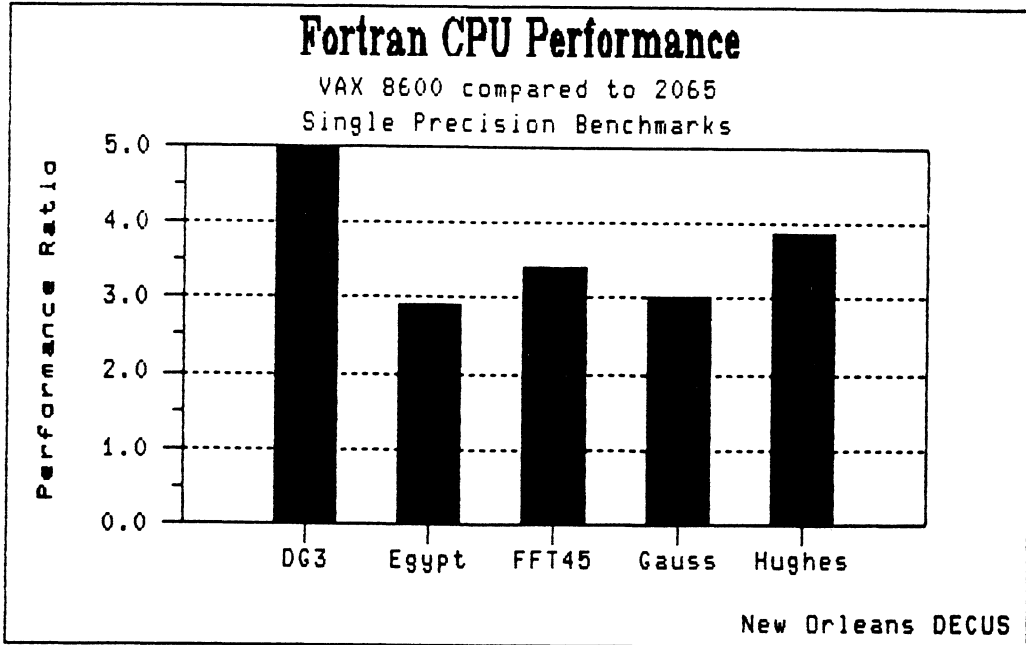




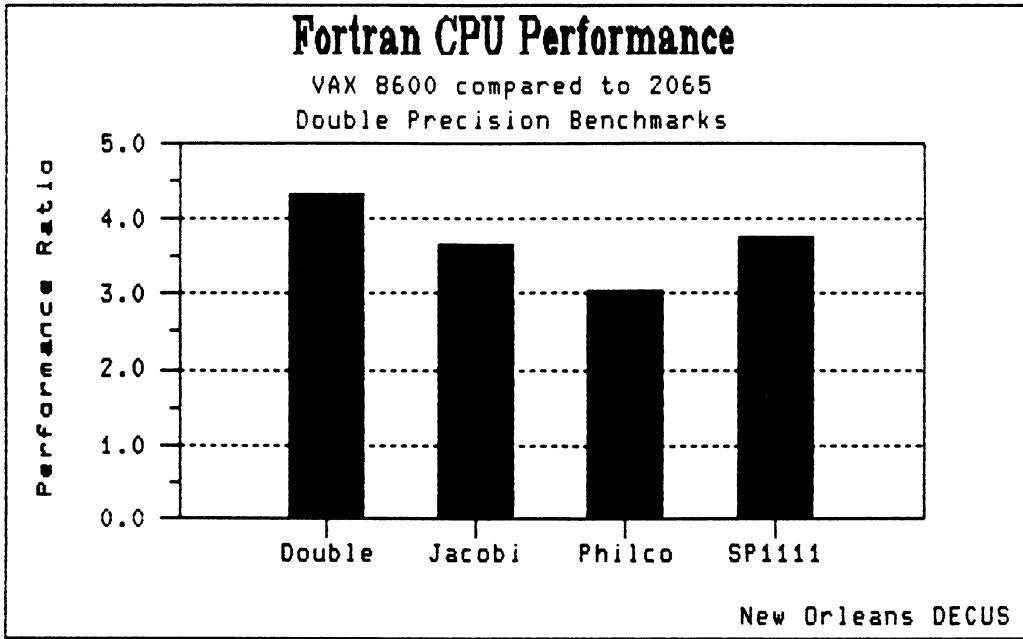
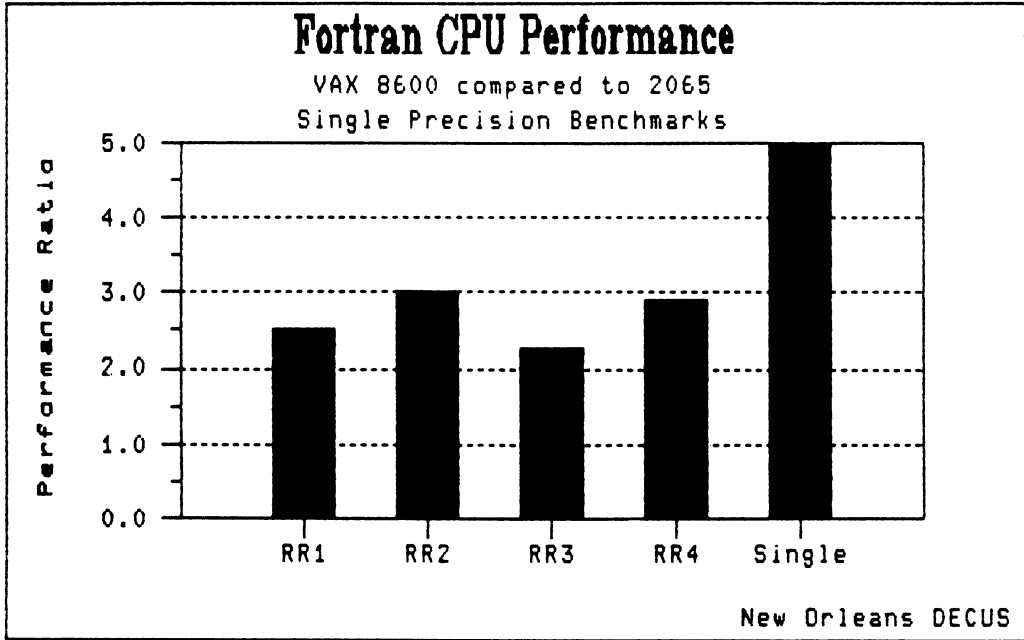


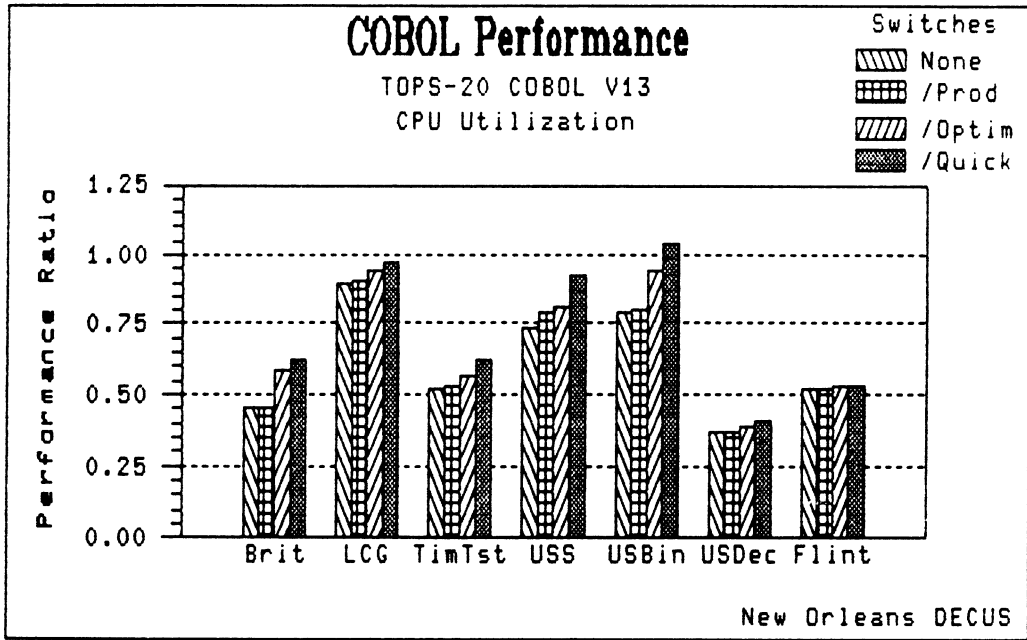
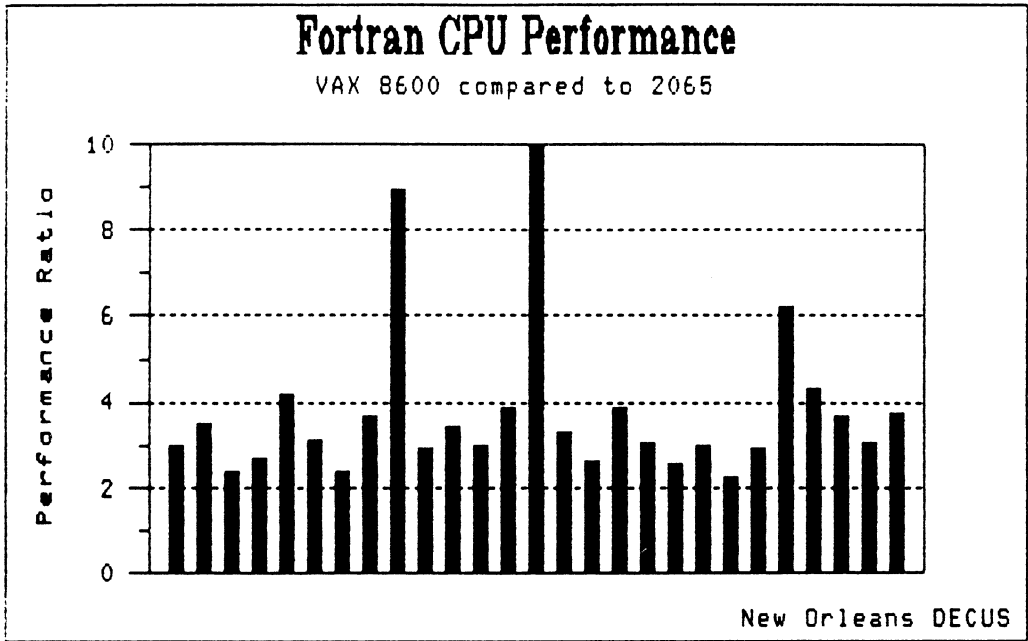


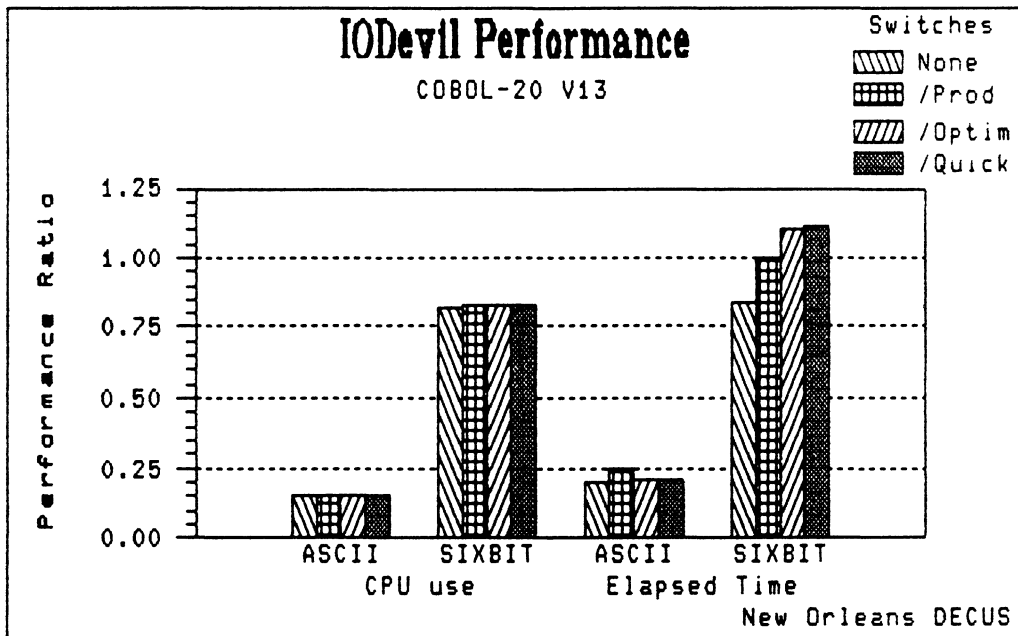
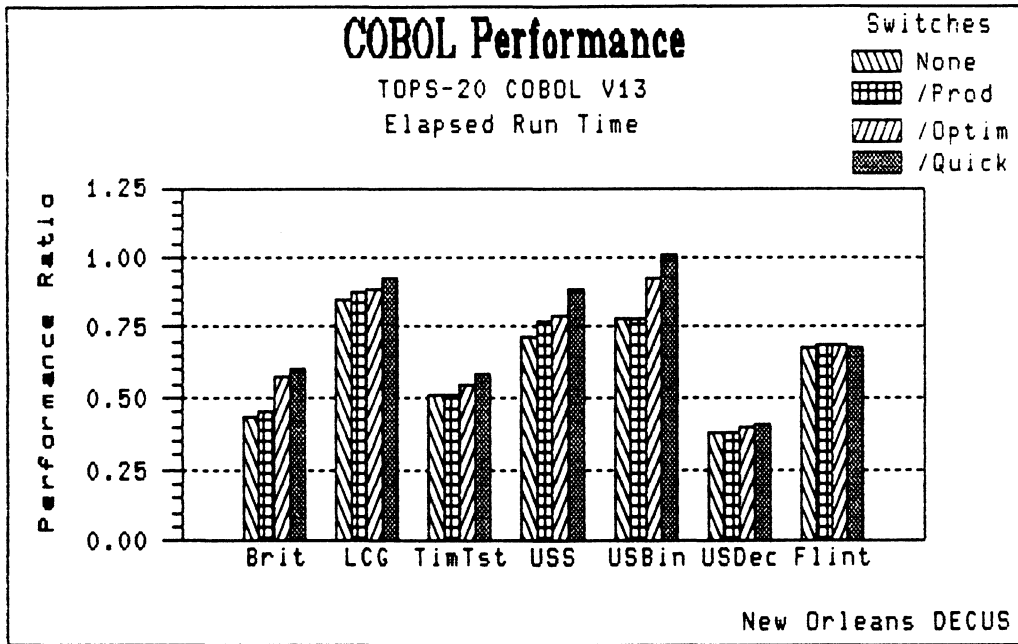


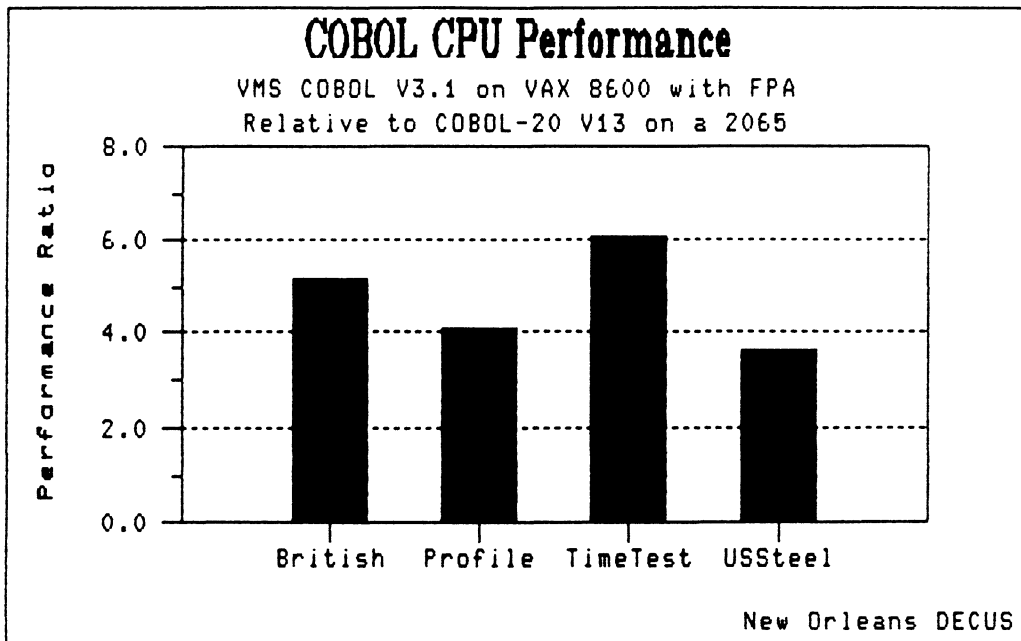
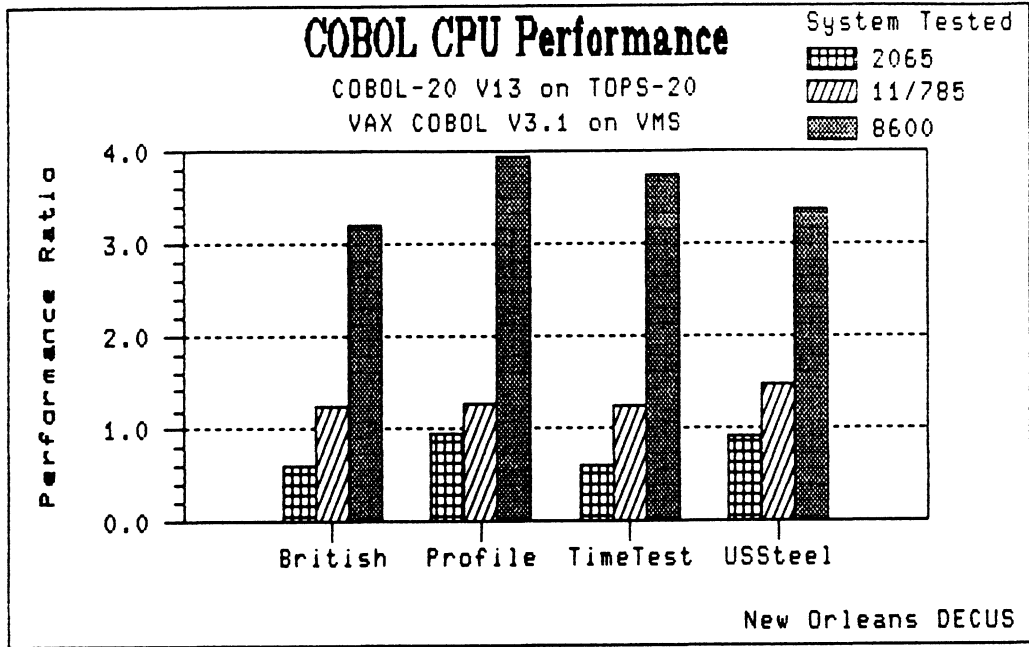




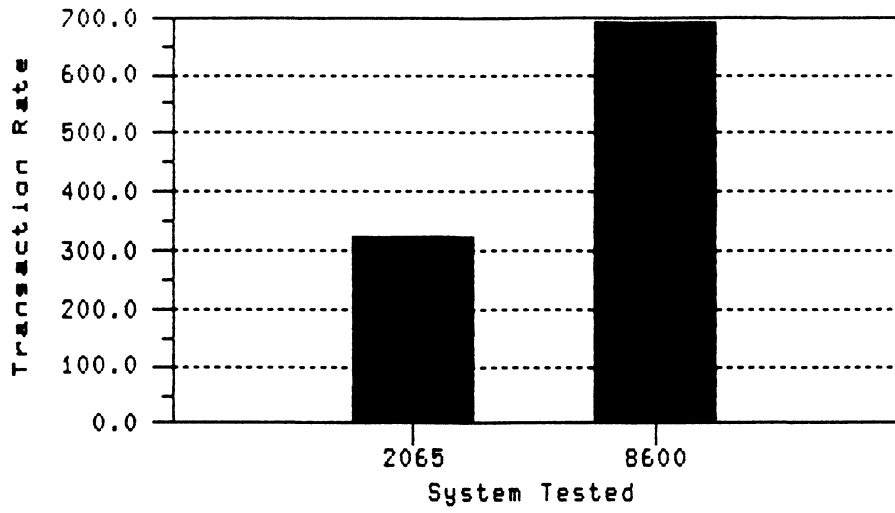






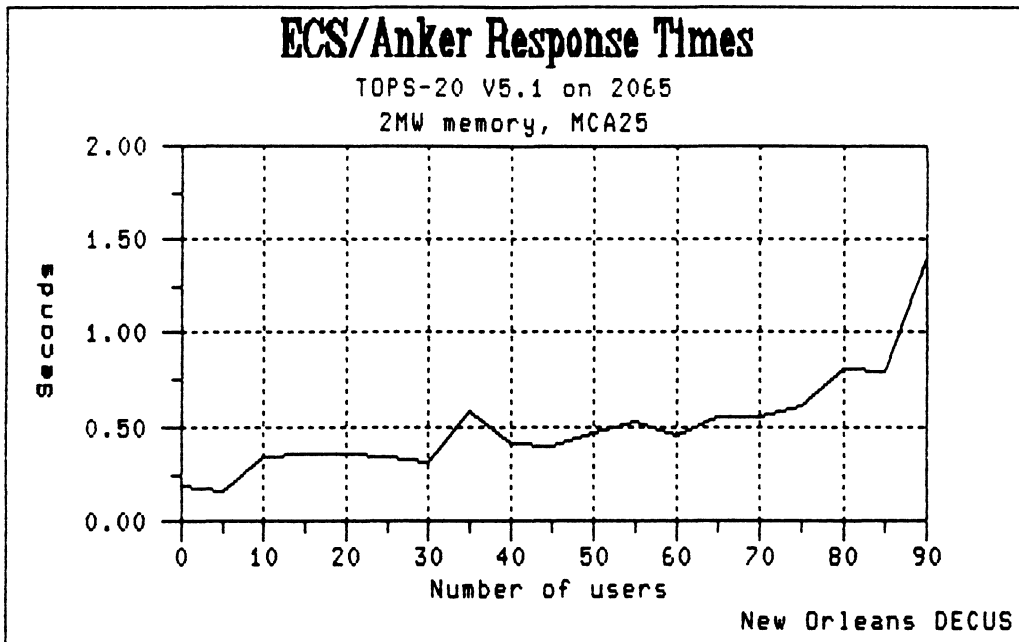


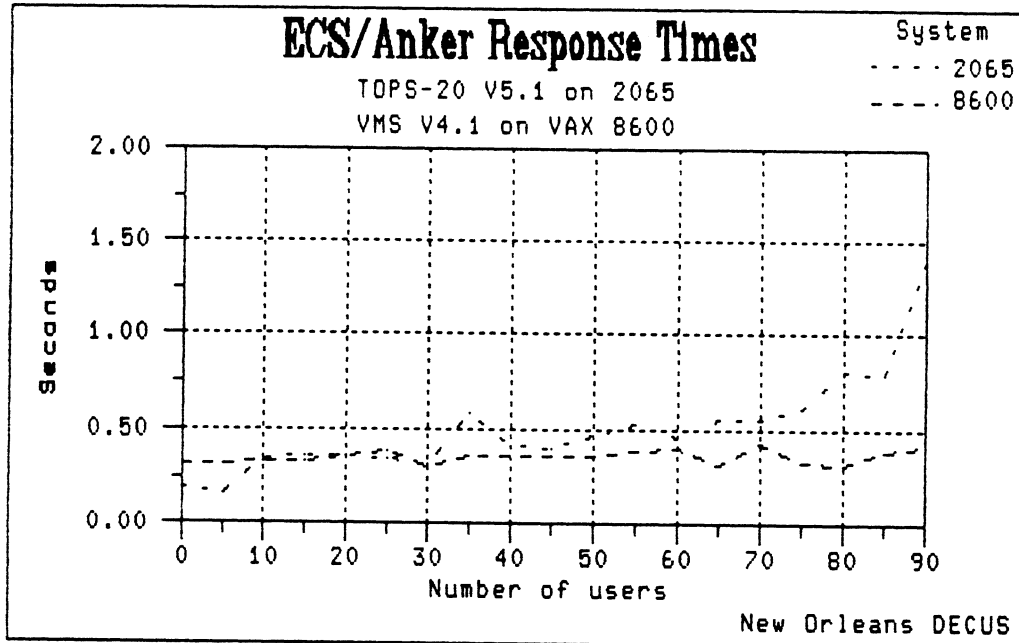
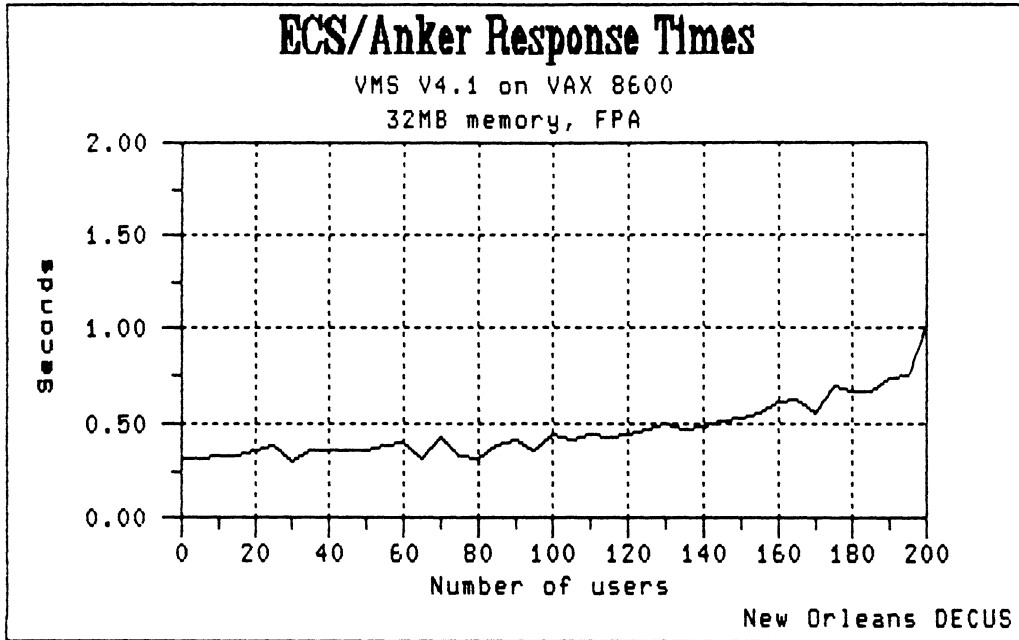
# Sorting



New Orleans DECUS

**M u l t i - u s e r**  
**" E C S / A n k e r "**  
**B e n c h m a r k**  
**R e s u l t s**





Peter B. Galvin  
University of Texas at Austin Computation Center  
Austin, TX 78712

ABSTRACT

Representatives of Digital Equipment Corp. discussed changes made to system software from version 5.1 to version 6.1.

AGENDA

- A. Introduction by Chairman.
- B. Presentations by Mark Pratt, Martin Palmiere, and David Lomartire of DEC's High Performance Systems and Clusters division.

General differences between V5.1 and V6.1 include:

- New hardware support: MG20, MCA25, CI20, NIA20, HSC50, RA81, and RA80 devices.
- Support for up to 4 megabytes of memory.
- New BOOT code, KL microcode, and RSX20F.
- Extensive use of extended memory by the monitor.
- Password encryption, with a one-way encryption supplied that is easily replaced by a user algorithm.
- Security enhancements, including: optional password rejection algorithm, ability to ignore passwords entered after N failures in a set amount of time.
- Autobaud detection from 110 to 9600 baud.
- The use of 6-1-SYSJOB.EXE AND 6-1-SYSJOB.RUN.
- Dumps placed in directory pointed to by the logical-name DMP:.
- POSTLD determines new PSECT origins upon overflow detection.
- Hardware numbers (serial and channel) are now reported in decimal.

CFS:

The computer interconnect (CI) facilities include access to HSC50 controllers and RA81 class disks, and allow the loose coupling of systems via CFS-20.

The CFS-20 environment consists of multiple DEC-20s (currently only 2 are supported) each having a CI-20 and CFS-20. One megabyte of memory is required for each system clustered. Dual ported MASSBUS disks (except the RP20) can be accessed from both systems, and single ported disks can be accessed by both systems via the MSCP server. HSC50 disks can be shared.

Generally, the CFS-20 environment is a natural extension of TOPS-20, works on homogenous systems only, allows shared access to disk structures, and is invisible to the users. It is a resource manager and permission arbitrator, and attempts to allow cluster recovery from a wide variety of failures.

DECnet:

V6.1 DECnet features include: Phase IV Level 1 router, elective endnode on the ETHERNET, large message buffers, end communication layer, network management, DECnet event logging, and CTERM support.

EXEC:

Major new enhancements to the EXEC are in the areas of multi-forking, ephemeral programs, and new commands.

Here is an abbreviated command by command breakdown of the changes incorporated into the V6.1 version of the EXEC:

- All commands which accept a device name as a field now allow the colon at the end of the device name to be optional.
- BUILD and ^ECREATE allow quotas of INFINITY, a PRESERVE option, the setting of TOPS10 PPNs, and never display passwords of directories.
- ^ECEASE requires confirmation and displays the node name of the system as well as the downtime requested. A NOW subcommand will cease the system immediately after confirmation.
- COPY has SUPERSEDE subcommands.
- DEFINE now allows input recognition.
- DIRECTORY now has a COMPLETE subcommand which will display all information about each file.
- Several INFORMATION commands have been added or enhanced.
- LOGIN now allows a /FAST switch to avoid the automatic taking of command files. This feature can be disabled dynamically or in the system configuration file. Also, LOGIN now displays the last date and time of login, and executes systems wide LOGIN and BATCH command files.
- LOGOUT now supports a LOGOUT.CMD file.
- A PERUSE command has been added to run EDITOR: with a read-only option.
- PUSH will execute the EXEC pointed to by DEFAULT-EXEC:.



- RECEIVE/REFUSE USER-MESSAGES is included for unprivileged TIMSGs.
- SEND now works for unprivileged users and has a new user name argument.
- SET HOST invokes CTERM-SERVER or a program pointed to by NRT:.
- SET STATUS-WATCH changes the control-character used to display program status.
- SYSTAT now displays originating hostname of network connections.
- SET [NO] TRAP JSYS /DEFINED and /UNDEFINED has been implemented.
- TYPE now has an UNFORMATTED subcommand to disable CCOC translation.

In addition to these commands, all of the previously unsupported multi-forking commands are now supported. In addition, PCL and a command-line editor can optionally be included in the V6.1 EXEC.

#### DUMPER:

Version 5 of DUMPER will be shipped with the V6.1 Monitor and Exec. It's features include:

- 5-30% decrease in CPU time used for all operations.
- Executable under any version of the monitor.
- Restores files from any DUMPER tape.
- The PRINT command can format output for CRT screens.
- More information is provided by the control-A command.
- Improved handling of the control-E command.
- Automatic handling of interchange tapes.
- Longer saveset names are allowed.
- MAIL messages are now optional for several operations.
- New help and documentation are provided.

#### GALAXY:

Most of the changes to the GALAXY subsystem of TOPS-20 were made to support the common file system. Additional changes were made to implement the QUEUE% jsys, and to allow GALAXY to produce BUGINFs and BUGCHKs through OPR.

#### SETSPD:

The SETSPD system startup program has added functionality in the area of CES, ETHERNET and DECnet control.

**OFFICE AUTOMATIONS SIG**



ALL-IN-1/WPS-PLUS  
DOCUMENTATION DIRECTIONS

Sue Franklin  
Digital Equipment Corporation  
Maynard, Massachusetts

ABSTRACT

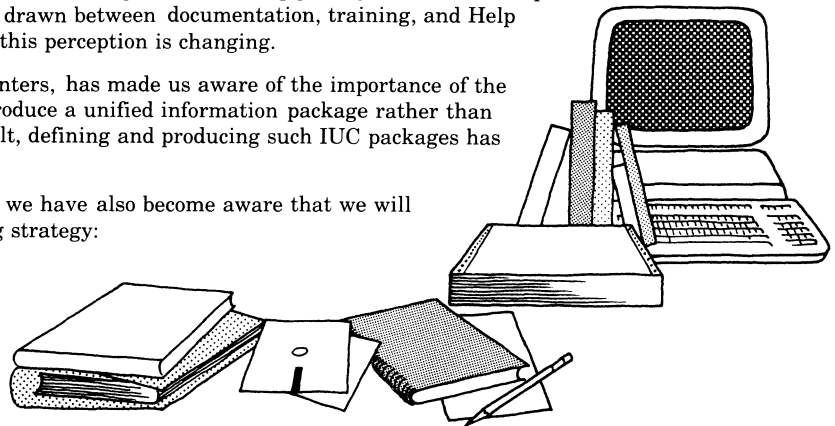
This paper discusses integrated user communications (IUC), a major goal of DIGITAL's Office Systems Documentation group. To meet this goal, the user interface, the on-line Help, the Computer Based Instruction (CBI), and the hard-copy documentation cooperate in a unified, coherent, consistent manner to teach the user the system. This paper also details the strategy that will produce integrated user communications, describes its design and implementation in the ALL-IN-1 and WPS-PLUS documentation sets, and indicates future strategic directions.

Office Systems Documentation (OSD) is a group of writers and editors responsible for providing hard-copy documentation for several of DIGITAL's office automation (OA) products. We are not chartered to produce training packages or on-line Help. In the past, the lines of responsibility have been sharply drawn between documentation, training, and Help (considered a software development activity). Gradually, this perception is changing.

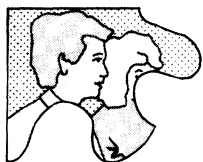
Working with OA software, both as users and as documenters, has made us aware of the importance of the user communications package and of our obligation to produce a unified information package rather than separate pieces of documentation and training. As a result, defining and producing such IUC packages has become a major goal for our group.

As we have become aware of the importance of this goal, we have also become aware that we will not meet it without careful planning. Thus, the following strategy:

- Phase 1: Definition
- Phase 2: Implementation of Product-Oriented Goals
- Phase 3: Expansion and Cross-Implementation of Goals
- Phase 4: Delivery of Complete Packages



Having completed the documentation for three versions of WPS-PLUS and for ALL-IN-1 Office Menu V2.0, we are now at Phase 3 of this strategy.



### 1.1 Phase 1: Definition

As stated, OSD is chartered to write hard-copy documentation for several of DIGITAL's OA products. We work closely with development to produce technically accurate, useful documentation. Traditionally, an Educational

Services group provides the training and the development group produces the user interface (menus, forms, screens, prompts) and the on-line Help. Therefore, the education of the user was the joint responsibility of at least three different groups with separate priorities, schedules, and constraints.

OSD believes that the user would benefit from a more consistent, complete, and unified approach to the product. As we explored alternatives, we coined the term "integrated user communications" and developed the following working definition.

Integrated user communications means applying a single set of guidelines and conventions to every aspect of the user interface. The user interface is the critical component because it is through the interface that the user comes to know the product. Often described as that point where man and machine meet, it is the medium through which the user receives information about the software's functionality. Particularly in the office automation arena, the user interface IS the product.

With this in mind, we define the user interface as the product interface (menus, forms, screens, prompts), the on-line Help, the CBIs, and the hard-copy documentation. Assuming that there are three general types of users (new, competent, expert) and using experience and customer feedback, we can begin to understand the nature of the interaction that occurs when user and product meet.

Instructional materials best serve the new user by providing basic, step-by-step, task-oriented information. Primers, other types of tutorial material, summaries, and executive overviews are desirable for this user. This type of documentation shows screens frequently and uses examples and illustrations everywhere. CBIs fall into this category as well and they are ideal for new users or for passing one-time-only information. They should be short enough to be quick refreshers and they get high marks if the user can do real work with the CBI.

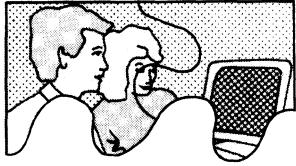
User guides, quick lookup guides, master glossaries, master indexes, Help, and all other types of on-line assistance (prompts, error messages, on-line documentation) serve the mid-range user who is competent but who, occasionally, needs help. This material should be context-sensitive and designed for the person who uses the system frequently.

At the expert level, we see a demand for technical details, a high tolerance for dense material, and a low need for illustrations and

examples. Reference material best suits this user who wants only the facts and wants them presented in as concise and reference-oriented a format as possible.

Our goal is to have all these components cooperate in a unified, coherent, consistent manner to teach the user the system. This is integrated user communications. It is an attempt to match the strengths of each medium to the individual's preferred learning style and to the types of information that must be transmitted.

This is the future, but it is a future we are beginning to implement today.



## Phase 2: Implementation Of Specific, Product-Oriented Goals

Early in the development cycles for ALL-IN-1 and WPS-PLUS, we established different sets of goals for each product based on what we perceived as the most critical needs. We formed our definition of "critical need" using experience, marketing information, and development goals. Nevertheless, we made every effort to work the principles of IUC into our current projects.

The documentation efforts for ALL-IN-1 V2.0 and WPS-PLUS remained separate because, initially, the two products were not as fully integrated as they are today.

### 2.1 ALL-IN-1 Version 2.0 Goals

Accepting input from marketing, product management, and development, OSD developed the following set of goals for the ALL-IN-1 V2.0 documentation set:

- Document the default system, the system as delivered to the customer
- Improve the quality (and quantity) of the existing Version 1 documentation
- Create a package that is both flexible and customizable
- Create a package that is both modular and integrated

Other goals included simplicity, consistency of terminology, organization and presentation, and use of material (paper, binders, and so on) readily available worldwide.

#### 2.1.1 User Documentation For The Default System

One of ALL-IN-1's strongest features is customization. This means the customer can:

- Change the appearance of menus and forms
- Add or remove menu options or commands
- Change the way a menu option or command operates
- Rearrange options on different or new menus
- Add or remove applications or entire subsystems
- Remove or redefine keys

In other words, the customer can alter ALL-IN-1's interface and functionality.

Nevertheless, it was apparent that we needed to provide a set of documents that guided the spectrum of potential readers – from new users to experienced programmers and system managers – through the product as delivered.

Addressing this requirement, the strategy provided for a complete documentation set covering all levels of expertise. At the user level are the following manuals:

- An *ALL-IN-1 Getting Started Guide* that contains exercises and examples based on the major components. This step-by-step primer gives users practice in performing basic office tasks using ALL-IN-1 Office Menu V2.0. Throughout the guide, readers are encouraged to use the on-line Help and CBIs.

To help readers move easily between manuals, the *Getting Started Guide* makes direct references to corresponding chapters in the *User's Reference* and to keypad layouts in the *Keypad Cards*.

- A two-volume *ALL-IN-1 User's Reference* based on an extended quick-reference format that describes ALL-IN-1 V2.0 functionality and the tasks that can be performed with the default system. The goal of this document is to provide, in one place, all the information on a particular subsystem or application.

The *User's Reference* is for users familiar with ALL-IN-1 system basics.

- A set of *ALL-IN-1 Keypad Cards* that illustrate the default keypad definitions for the system-wide features, major applications, and the editors. These cards are a useful tool for all users moving between applications. Each page shows a keypad layout for a particular subsystem or application and lists any Gold-key functions for that layout.

In addition to a technical update, the cards were expanded to more fully describe the keyboard functions.

#### 2.1.2 Technical Documentation For System Managers And Developers

The second goal for V2.0 was to provide complete and accurate technical information and to reformat that information to be compatible with documentation provided for VMS and VMS layered products.

The amount of technical information documented for ALL-IN-1 V2.0 represents a significant expansion over previous versions.

- The *ALL-IN-1 Application Programmer's Reference (APR)* is expanded to three volumes and provides a detailed overview of how ALL-IN-1 V2.0 works, as seen from the programmer's perspective.

- *Volume 1: Flow Control* discusses ALL-IN-1 initialization and flow control, form processing, field processing, and generic menu design.

- *Volume 2: Functions* covers the three types of ALL-IN-1 functions: control functions (analogous to VAX/VMS DCL commands), user-defined functions (analogous to commands created with the DCL COMMAND utility), and internal functions.

- *Volume 3: Applications* describes all aspects of application development, including the ALL-IN-1 File Cabinet, the sub-processes and subsystems, and application tools and design guidelines.

In addition, *Volume 3* provides a set of exercises designed as a self-paced ALL-IN-1 Office Menu programming course.

The *APR* also expanded to include flowcharts and similar illustrations and to point the reader towards further VMS references when it does not explain a necessary concept in detail.

Each volume contains a complete table of contents and a complete index.

- The *ALL-IN-1 Programmer's Mini-Reference* provides a quick reference for the developer. It lists and summarizes the qualifiers, functions, and main applications found in the *APR*.

- The *ALL-IN-1 System Manager's Guide* provides information on maintaining ALL-IN-1 V2.0. It covers the system management of integral ALL-IN-1 facilities and of applications implemented through ALL-IN-1.

Intended for those individuals responsible for the maintenance and efficient running of ALL-IN-1, the guide is divided into three parts: User Support Activity, ALL-IN-1 Management, and Multi- and Inter-Node Management.

- The *ALL-IN-1 Installation Guide* contains step-by-step instructions on how to install V2.0. It also covers setting up the DCL command to run ALL-IN-1, creating user accounts, verifying the installation, and converting from a previous version. It anticipates and discusses possible installation problems.
- The *Version 2.0 Release Notes* document features and changes not covered elsewhere. These notes are available on line at installation.

### 2.1.3 User Documentation for a Customized System

The documents most impacted by system customization are the documents that describe the user interface – the *Getting Started Guide* and the *User's Reference*. Once the product has been customized, the default user documentation no longer describes what the user sees.

Anticipating this issue, we developed the Customizable Documentation Kit. This optional kit contains the tools and the information the documenter needs to customize the user manuals and produce new manuals that physically match the default set.

The Kit consists of:

- The *ALL-IN-1 Style Guide* that discusses the ALL-IN-1 V2.0 documentation strategy, standards, conventions, templates, and specific documentation tools used in the manuals. This guide includes sample chapters to assist documenters in customizing the manuals. It provides technical specifications for materials and processes used, including specifications for type, size, and weight of paper, use of color, art, and other visual aids, and packaging information.

The guide also tells documenters how to access the on-line text files and describes the use of DIGITAL Standard Runoff (DSR) commands in the default documentation set.

- The *ALL-IN-1 Writer's Guide* that discusses the software documentation process, including planning, reviewing, production, and printing. Technical documentation and production terms are kept to a minimum.
- A magnetic tape that contains the full text of the *Getting Started Guide* and the *User's Reference* in DSR format. Because these text files contain the DSR commands that format the documents, we call them *source* files. The meaning of the word *source* in this context is analogous to its use in programming.

The tape also contains master files and command files to ease the rebuilding of customized documents.

### 2.1.4 Achieving Modularity And Integration In Documentation

ALL-IN-1 V2.0 is a highly modular system and, at the same time, a fully integrated system. Reflecting modularity, the *Getting Started Guide* and the *User's Reference* are composed of self-contained chapters. Each chapter consists of information about a particular subsystem.

Further, each chapter is written to follow a single template or pattern (one for the *Getting Started Guide* and one for the *User's Reference*). The templates make it easy to produce user manuals that are consistent in format, organization, and writing style. The templates also make these books simple to update, customize, or translate.

The ALL-IN-1 templates are based on the menu structure as the organizing element in the product. They copy the software itself by presenting a top-down view, the user's view, of the system. The reader gets an overview first and then is led through a discussion of each option in increasing layers of detail and complexity.

The user documentation templates for ALL-IN-1 are task-oriented. That is, they are task-oriented in as far as the menus themselves are task-oriented and form lists of activities that the user may wish to perform.

Reflecting modularity, the templates separate the user interface from the functionality. What the user does (that is, select options from menus to initiate actions, functions, applications) is covered by procedures (series of steps). What happens as a result is described in narrative paragraphs following the procedures.

This procedural approach means that users can quickly scan the *Getting Started Guide* and the *User's Reference* while getting a sense of the number of steps any activity. The steps provide handy reference points for starting, stopping, continuing, or skipping tasks.

The templates also provide consistency. The user need become familiar with only one format or style of presentation in either book.

Finally, templates give greater stability to those documenting a customized system. User manuals document the user interface, the area most affected by customization (from the user's perspective). This material is isolated in the procedures and can be easily dealt with there. Descriptions of functionality are isolated in paragraph modules and can be dealt with separately.

Used in combination with the Customizable Documentation Kit (which provides the on-line text files), a documenter can remove, add, or rearrange whole chapters or sections to reflect the user's final software configuration. Users perceive the resulting customized documentation as an integrated whole just as they see their ALL-IN-1 system as an integrated whole.

Documenters using the templates to produce customized books will find that they are structured enough to provide a general pattern and open-ended enough to allow the individual writer to express a measure of creativity.

## 2.2 WPS-PLUS Documentation Goals

With the ALL-IN-1 project, the documentation group faced the challenge of providing a set of books for an interface designed to be changed (customized). That challenge produced the Customizable Documentation Kit.

With WPS-PLUS, we faced a slightly different challenge. Here, the basic *core* of functionality stays the same while the interface changes slightly under different implementations and for different hardware configurations.

Currently, WPS-PLUS runs under several versions of VMS, under ALL-IN-1 V2.0, and under Rainbow/MS-DOS. Plans are also underway for a version of WPS-PLUS that will run on the Professional under POS. Because WPS-PLUS is available in so many configurations, there is interest in making the software and the documentation:

- Cost-effective
- Easier and faster to produce
- Fully international and easily translatable

However, none of these goals were seen to be as important as presenting to the user a single and consistent approach to learning WPS-PLUS regardless of the implementation.

The first step toward a solution involved two realizations:

- We realized that WPS-PLUS was basically the same wherever it operated. We called this *sameness* the *core* of WPS-PLUS.
- We realized that the differences between environments and versions of WPS-PLUS appeared, largely, in isolated areas of functionality.

Therefore, we knew that we would have to devise a documentation strategy that was largely environment and version independent.

The solution that developed represents a highly modular approach to documentation. It attempts to identify those aspects of functionality that remain the same across all WPS-PLUS systems. As this functionality is the *core* of WPS-PLUS, it forms the basis of what we call the *core* concept in WPS-PLUS documentation.

### 2.3 The WPS-PLUS Documentation Set

The core concept separates those parts of WPS-PLUS that are the same everywhere (*core*) from those parts that are different (*non-core*). Correspondingly, the documentation set is divided between core manuals and non-core manuals.

For each WPS-PLUS documentation set, there are two core books (books that are 90% or more system- and version-independent). All three published versions of these two books are nearly identical.

- *WPS-PLUS Editor Functions* describes the features of the WPS-PLUS editor for those users who are already familiar with the basics of WPS-PLUS word processing.
- *WPS-PLUS List Processing* explains how to use WPS-PLUS List Processing, List Processing Math, and Sort Processing.

There are two books that are approximately 70% system-independent.

- *WPS-PLUS Quick Lookup* for people who have used WPS-PLUS but who want a quick and brief reminder of the steps involved in an operation.
- *WPS-PLUS Getting Started*, a tutorial designed to teach the user a set of basic tasks: creating a document, editing a document, printing documents, and deleting documents. It does not cover all system functionality and should be used with the WPS-PLUS CBIs.

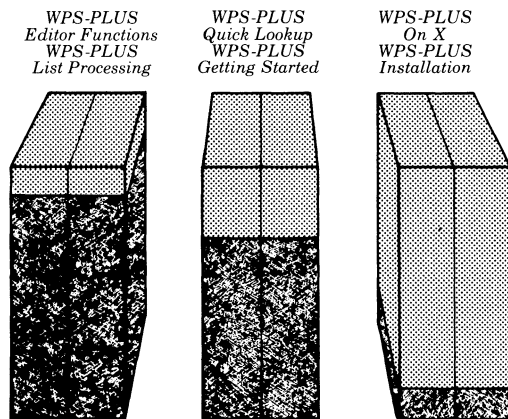
There are two books that are 10% or less system-independent (major differences exist between versions of the documents).

- *WPS-PLUS On X* (where X is the operating environment, such as VMS or Rainbow) handles product functionality that actually differs across implementations. Examples are print functionality, File Cabinet maintenance, User-Defined Key/Procedure maintenance, and document transfer functionality. In addition, this guide covers the various menus, forms, and keyboards encountered. It is the user interface book for a given WPS-PLUS implementation.

In the case of WPS-PLUS/ALL-IN-1, we were able to eliminate this book entirely because those aspects of product functionality were covered by the *ALL-IN-1 User's Reference*.



% of System Independence



- *WPS-PLUS Installation* describes how to install and maintain a particular version of WPS-PLUS within a particular operating environment. Logically, it cannot be a core document. The procedures for installing and maintaining WPS-PLUS differ radically from environment to environment.

The simplest illustration of this concept may be seen in the product names: WPS-PLUS/VMS, WPS-PLUS/ALL-IN-1, WPS-PLUS/Rainbow. The core name (WPS-PLUS) is the same in all three cases. Only the appendage that describes the implementation environment changes. You might think of /VMS, /ALL-IN-1, and /Rainbow as non-core elements in this case.

The issue becomes more complex when dealing with functionality differences. *WPS-PLUS Editor Functions* documents the WPS-PLUS editor and every version contains the same information – except for specific references to VMS, ALL-IN-1, or Rainbow features.

For example, since the *WPS-PLUS/Rainbow Editor Functions* documents WPS-PLUS on a personal computer with floppy disk drives, the user must be aware of the physical location of documents. To select a document, WPS-PLUS/Rainbow users must enter B: before the document title if the document is stored on Drive B:. The B: preface is called the path name of the document.

As a result, *WPS-PLUS/Rainbow Editor Functions* reminds users to include the path name when they select a Library or Abbreviation document. This difference appears only in *WPS-PLUS/Rainbow Editor Functions* and not in any of the other versions of this manual.

|                                                                     |                                                                                                             |
|---------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| The following equations may help make these concepts more concrete: |                                                                                                             |
| Core Software =                                                     | Functionality that is always present, regardless of the version of WPS-PLUS or of the operating environment |
| Core Documentation =                                                | Documentation of that functionality                                                                         |
| Non-core Software =                                                 | Functionality that varies across operating environments                                                     |
| Non-core Documentation =                                            | Documentation of those variations                                                                           |

To summarize the core concept, it:

- Gives consistency in organization and format to users migrating to and from different operating systems
- Factors out "generic" material to create general purpose, reusable modules of information
- Eliminates needless redundancy in instructional material
- Captures and preserves well designed, written, and tested modules of information
- Allows for the insertion or deletion of material at any audience level or for any version or implementation

We should be able to continue producing *core* documentation as long as engineering produces a *core* WPS-PLUS.

### 2.4 On-Line Help, CBIs, and the User Interface

During the development cycles for ALL-IN-1 and WPS-PLUS, the documentation group was able to provide considerable input in these three areas. We were directly involved in the writing of Help frames for both ALL-IN-1 and WPS-PLUS and served as consultants to CBI and user interface development. Particularly in the Help area, every effort was made to make the information consistent with and complementary to the hard-copy documentation.

### 2.4.1 The On-Line Help

The on-line Help was expanded significantly for both WPS-PLUS and ALL-IN-1 V2.0. You can get Help anywhere, anytime by pressing Gold H. Providing support to both new and experienced users, ALL-IN-1 and WPS-PLUS display context-sensitive messages in a window overlay at the top of the screen. The Help message includes a list of related Help topics as well.

Characteristics:

- Help on menus and forms (level 1)
- Help on menu options and form fields (level 2)
- Help on editing keys and general information topics (level 3)
- Modular – Each module a single Help topic
- Uses templates
- Builds cross-reference lists automatically

Guidelines and templates are also provided for those who wish to write their own Help or supplement the existing ALL-IN-1 Help. It may not always be possible to follow these templates exactly, but the more consistent the format, the easier it is to update and customize Help.

### 2.4.2 The CBIs

The CBIs for ALL-IN-1 V2.0 and WPS-PLUS embody important new principles. They are task-oriented, pulling in the user's actual files in the execution of a task. For example, while learning to read mail, users read their own real mail messages. If they have no mail, the CBI sends a mail message. Thus, the user is learning how while actually doing.

In this same sense, the CBIs are interactive. The user interacts with the product instead of the CBI simulating an interaction with the product.

Finally, the CBIs were designed to be easy to use. A user needs no prior knowledge of either of these products to activate and run the CBIs.

### 2.4.3 The User Interface

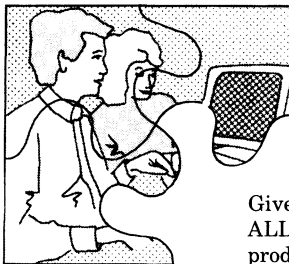
Finally, the software interface itself came under careful scrutiny during the development of ALL-IN-1 V2.0 and WPS-PLUS. Important design considerations were that the products be predictable in behavior and consistent in the presentation of menus, forms, screens, error messages, and prompts.

The approach is basically top-down and menu-oriented. This approach minimizes memorization by presenting a list of choices available to the user at each step in an activity. The user is further assisted through the use of:

- Descriptive phrases and mnemonics
- A small number of form types
- A small number of universal options that may be invoked from any subsystem
- In ALL-IN-1, an open, customizable architecture that may be personalized to suit each user

ALL-IN-1 and WPS-PLUS also support the more experienced user by allowing the entry of a string of commands at the initial screen or menu.

Finally, the products give users the ability to create User-Defined Procedures (UDPs), which are documents used to store frequently invoked commands or keystrokes. When the user invokes a UDP, the system executes this series of commands or keystrokes automatically. In this way, the system *watches and remembers* a sequence of steps and, in effect, can learn from the user as the user is learning the system.



### Phase 3: Expansion And Cross-Implementation Of Goals

Given our experience with WPS-PLUS, ALL-IN-1, and other OA systems and products, we have gained considerable expertise in turning out hard-copy docu-

mentation. We try to understand our audience and we try to understand the office environment. Our documentation packages attempt to embody what we know about both. During this phase of the IUC strategy, we will try to implement the lessons learned working with ALL-IN-1 to WPS-PLUS and its options (high quality print and graphics).

In particular, we plan to continue to:

- Refine the Core Concept  
We will continue the movement towards user communications packages that are increasingly modular and generic (core). This approach gives users greater consistency in the same way that software achieves consistency from modular and generic code.
- Increase the use of templates in documentation and on-line Help

Both core documentation and the use of templates give the user consistency in instructional approach and give us greater stability as development moves through the product release process. Despite the instability inherent in any development cycle, much of the core functionality remains solid from version to version and across versions. When the software is stable, the documentation is stable. Thus, we are able to concentrate on new areas instead of continually rewriting descriptions of basic functionality.

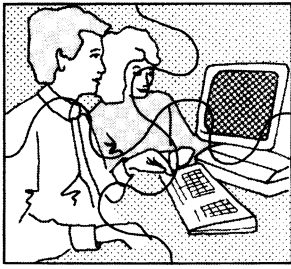
As discussed, the use of templates also minimizes the impact on both the user and the writer when functionality does change.

Expanding on our traditional role, we will try during this phase to influence development to provide:

- Simple (not simplistic), intuitive, logical systems
- Consistent and, therefore, predictable systems
- One way to perform one function
- Only the functionality needed to get the job done
- Customizable, open systems

Finally we plan to seek earlier and more direct involvement in on-line Help, CBI development, and user interface design. We support this expansion of traditional responsibility because it allows us to gain control over critical areas and to put into practice what we believe about the needs of our audience.





## Phase 4: Production Of Complete Packages

Our goal, then, is to produce complete and fully integrated user communications packages, packages that adhere to a single

set of guidelines and conventions. At its highest level, this set of guidelines consists of the following elements:

- Technical accuracy and completeness
- Consistency and clarity in terminology and presentation
- The treatment of a system as a system, not as a collection of products

After technical accuracy, consistency is perhaps the most critical element. Perhaps surprisingly, it is a difficult goal to achieve – even within a single area like documentation. During the final phase of our strategy, we must carry this goal even further until we achieve consistency across documentation sets, on-line Help, CBIs, and the user interface. The elements identified as critical to the issue of consistency are:

- Terminology (definition of terms)
- Word usage
- Format (presentation) templates

Another key concept during this phase is called *the systems approach*. The documentation efforts for ALL-IN-1 and WPS-

PLUS were designed to teach us, as communicators, how to document a system as a system (solutions to business problems) and not as a collection of products and tools. IUC is part of a systems approach because it delivers to the customer a single, organized instructional package. To produce such packages, we must constantly remind ourselves that:

- No one is primarily interested in the technology. People want to get their work done.
- Users are experts in the content of their jobs. The best we can do is to help them do tasks faster and more efficiently.
- People do not work in isolation. The office is a community and should be seen as a place where people constantly interact and share work, decisions, reports, and judgments.
- Tasks are not discrete entities and should not be automated or documented as if they were performed in isolation. Every task, decision, or judgment has antecedents and consequences and is, itself, part of a system.

In the areas of human factors and support of our customers, we can always improve. We believe that we understand, and have respect for, our audience. We read marketing reports, take courses, attend conferences, and talk to customers whenever we can. We have worked at audience definition and user profiles for years.

The lessons for us are clear, and I believe we are moving in the right direction. Our goal is to focus, not on theories or abstractions, but on the products and the people who will use them. The core concept, the templates, the Customizable Documentation Kit, the task-orientation, the emphasis on consistency and a systems approach and, indeed, the concept of integrated user communications itself, all support that goal.

Myron K. Hayashida  
American Management Systems, Inc.  
Arlington, Virginia

ABSTRACT

Office automation technologies span the Information Spectrum, i.e., information content, form and flow from the event which creates the information to the management or operational decision it supports. Accordingly, requirements analysis procedures and productivity measurements need to apply to the spectrum also. An information-based OA planning approach and methodology is offered to truly integrate OA into the total information systems environment of computers, organizations and people.

**The Office Automation Myth**

"Office automation improves the productivity of people. . ."

What could be further than the truth? It would appear that the explosion of office systems over the past few years has produced little in terms of the promised productivity for all but a few of the most typing bound secretaries. Organizations in search of technology got just what they sought, pieces of technology which contributed little to the office except faster typing, all this despite the enormous computing capability of these machines. One result was a glut of hardware which saw little use commensurate with their capability, especially among the professional staffs.

What is closer to the truth is that people improve the productivity of office automation. Office computers just sit and wait for someone with ambition and innovative ideas come along to put the computer to work. Without imaginative people to provide the impetus, the most sophisticated systems in the world will just sit with little to do. Office systems must be viewed as tools for people doing their jobs, and not simply as technology. Accordingly, office automation must be oriented toward providing solutions to people who have job needs.

One way of integrating office automation into the job environment is to provide a means for basing OA requirements on information needs in a substantive way. Merely measuring office workload to justify automatic typing and printing represents a word processing mentality which is not in step with the technology. Information is the principal commodity of the office and needs to be managed as a resource. Information should also be the basis for defining office automation system requirements. The perspective offered by the Information Spectrum approach provides the breadth of scope necessary to focus on information as the basis for systems development and acquisition.

The objectives of this paper are to review information planning methodologies, compare them with OA equipment justification methodologies, describe an integrated information-based methodology and point out some current OA issues.

Problems with Information Systems Planning

The increasing investments in computerized equipment in the office have caused managers to sit up and take notice. Additionally, it is becoming recognized that automation goes beyond just being a productivity tool, but it provides the competitive edge needed by businesses to survive. The wide and varied scope of office and information systems has fostered a concern that perhaps more planning and coordination is required for computer systems so that the maximum need will be satisfied by the least equipment. In general, the problems with the information systems planning and development process are:

- o Business Needs Unsatisfied
- o Systems Unresponsive to New Technology
- o Incomplete Systems Design

In many ways, we cause our own problems. The I/S planning process is often subverted by such things as arrogance on the part of our high technologists, ignorance of emerging or existing technology, or organizational turf. The notion of the Information Spectrum forces a total view of information systems and affords an opportunity to evaluate system needs from an information perspective across technologies to overcome arrogance, ignorance or turf considerations.

**Current Methodologies**

Business Systems Planning (BSP). This IBM methodology is a top-view approach with a strong orientation toward business processes. It employs

a team of senior organization people on an intensive several month study, and leads to the definition to those processes most essential to the survival and success of the business. It further goes on to define "data classes" created and used by each of the processes and arrays both processes and data classes into a matrix called the Information Architecture. Several other steps are taken to relate processes to organizational elements to fix responsibility and to identify problem areas which may or may not be related to information systems. The Information Architecture represents the organizations total information need based on a logical presentation of processes, data classes and their relationships. this architecture becomes the basis for the implementation of BSP.

The implementation of a BSP is a follow-on effort to the initial BSP study and includes the development of a data architecture, and applications architecture and geographic architecture. (Figure 1) The data architecture typically consists of an identification and definition of organizational entities, those things which the organization needs to keep information about. Entities are derived from the data classes of the Information Architecture and charted to show functional relationships among entities. This model then becomes the foundation for data base design. The applications architecture maps applications against processes to determine which processes are not adequately supported by data. Existing systems become the basis for making this determination. Those processes which neither create nor use data classes are suspect. The geographic architecture maps the distribution of data in a physical sense, the objective being to determine where data needs to reside in order to effectively support processes.

Another methodology is structured analysis, the most popular being the methodology known as the Yourdon-DeMarco methodology. It is a top-down decomposition of functions or processes with the goal being a level of detail suitable for the development of computer programs. It uses such tools as data flow diagrams, a data dictionary and structured English to document the data processes and flows of an organization.

The data flow diagram is most useful to chart the essentials of how an organization processes its information and for charting its data flows. The use of simplified symbols for processes, a circle, data flow, an arrow connecting processes and other data sources, a rectangle to denote data sources of sinks, and an underlined title to illustrate a data store, all make for an extremely easy methodology. An example is shown in Figure 2. Its weaknesses include its lack of a process validation step verified by management and its inability to reflect temporal factors.

Key product analysis is a methodology which attempts to weave information requirements into justifications for office automation systems by identifying costing "key information products" to be used as a basis for estimating cost savings. It focuses principally on the professional staff, noting that this part of the organization accounts for the great majority of payroll dollars. It supplements key product data with quantitative office workload volumes. This methodology does not validate the key products themselves or identify the processes they support.

The steps for key product analysis include the determination of a productivity baseline of key products, a macro-level office automation system design, a functional specification, a cost justification assessment and a post-implementation audit.

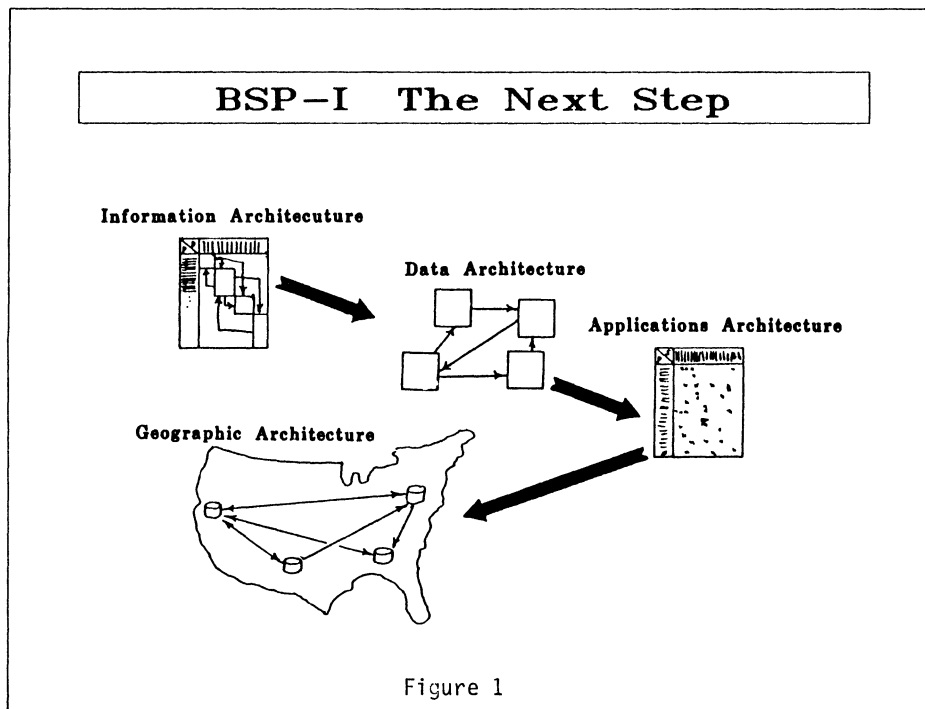


Figure 1

## Data Flow Diagramming - An Example

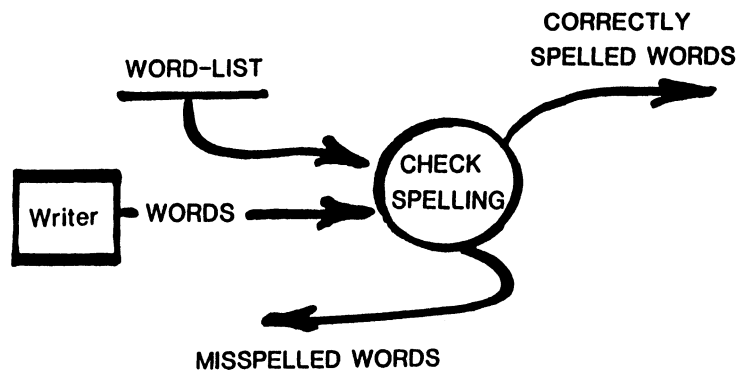


Figure 2

### Shortfalls of Current Methodologies

Only partial systems analysis. Each methodology offers only partial analysis. BSP is too top oriented to be of value to the system designer in any meaningful sense. Structured analysis is weak in addressing what should be as opposed to what is. While key product analysis comes close in objective to what office automation analysis should be, it falls short in providing data about other than key data, which may amount to a significant omission.

Inadequate communications at all levels. Whatever model is created by the methodology should be understood and usable at all levels in the organization. BSP does very well in providing top-management's view of information, but does not do the same for people lower in the organization or for those doing the system design work. Structured analysis does fairly well when decomposed to levels which can equate to system modules, i.e., a process which can be viewed as an integral set of software support data flows and inputs and outputs. Key product analysis is keyed for justifying costs and savings and not very useful to users and designers.

No help with hardware selection. All these methodologies leave the selection of supporting hardware to chance. It is expected that the analyst use abundant amounts of imagination in conjuring up a network of sorts to solve the functional problems and data flows. However, there are well established and recognized decision rules and principles which can and should be built into a methodology which will assist the user in configuring the best solution to the functional problem. For example, decisions on how to distribute data to best support business processes can logically lead to the selection of whether data should be maintained at the corporate level (mainframe), the department level or the workstation level (workstation).

No way to determine all information costs. Costs models can be built for each of the existing methodologies, but since they provide only a partial picture of the system, capturing all costs is not possible.

Poor transition into implementation. The absence of clearly defined bridges from plan to action is common trait among current methodologies. While intellectually appealing in approach, they leave one hanging when the plan must be implemented. The net result of any methodology should be a logically configured systems architecture consisting of information structures as well as hardware, software and communications. None of these methodologies provide this.

### **DIMENSIONS OF INFORMATION**

In order to be effective, any information systems methodology should address information needs from three perspectives: content, form and flow. (Figure 3)

Content pertains to the logical attributes of information. It includes its definition, its relationship to other bits of information and how it is logically represented to the system, i.e., how it is symbolically coded. Form includes the media the information is recorded on. It could be RAM, magnetic disk, paper, tape, film, optical disk, or any other means of recording data. Form also includes the physical symbology of the information, its representation to the hardware and software which must interpret the data to retain its logical attributes. Finally, flow embodies the movement attributes of information expressed in terms of volumes, distances traveled, routing and velocity or speed.

Next, the methodology should address the life cycle or path of information from its creation to use. When one evaluates the fundamental nature of

# Total Analysis Includes All Dimensions of Information

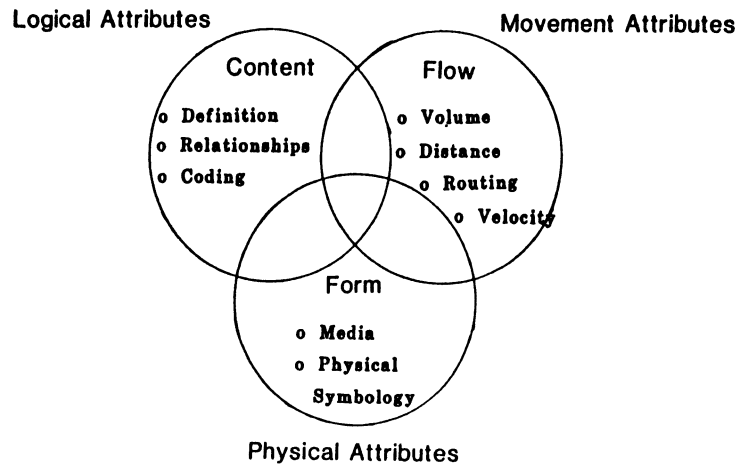


Figure 3

information, it becomes apparent that any information is a spinoff from some other activity or process; it does not exist for its own sake. In a given process, be it sales activity or manufacturing procedures, there are points in the process when something is observed, measured and recorded. The result is an information spinoff which flows to a decision-making process. It is typically a combination of these information spinoffs which cause decisions to occur.

The approach to analyzing this information cycle in conjunction with the three dimensions of information is what is referred to in the remainder of this paper as Information Spectrum Analysis.

The value of this approach becomes evident when an organization's information systems problem is identified as having too much focus on equipment and technology solutions and not enough attention to the business needs of the organization. Because equipment and technology has been emphasized in recent years, applications were developed for specific hardware technologies without regard to needs for sharing data and interfacing equipment. Hence we see an abundance of computerized micrographics, teleprocessing, word processing, microcomputing and mainframe system, each with its own set of standards for information and interfaces which are incompatible with other technology parts. There was little attempt to evaluate business processes, their information spinoffs and the need for information to traverse the gap between the event which creates it and the decision it supports in a systematic and orderly manner.

The Information Spectrum can be viewed as a continuum of information accessibility. In its simplest form for human consumption, voice, there is no need for a translation facility if the language spoken is the same. On the other hand,

more and more such translation facilities are needed as the information becomes increasingly difficult to read by humans, thereby making the information more remote from the user. Hence, as voice and paper, which are essentially readable without the aid of any special hardware and software devices, become more remote by distance or form, it takes more equipment to read it. The further away from voice and paper the information moves, it becomes less accessible to humans and more machines are needed to serve as intermediaries. Electronic signal and light pulses fall in the extreme category of requiring computerized devices to return information to a state of readability to humans.

As information travels the Information Spectrum, it moves in and out of various states of accessibility, some of which impact upon the timeliness of the information, the accuracy of the information and its value to the decision making process.

The total view afforded by Information Spectrum Analysis are benefits which include the following:

- o Total Systems Analysis - A top to bottom, end-to-end look at information processing and flow rather than by technology or hardware.
- o Bridge Between Planning and Implementation - A logical connection between business planning, information planning and the acquisition of computer systems to support information needs.
- o Communications Between Users and Developers - A better means to communicate user requirements to technicians and technical capabilities to users.

- o Justification for Hardware and Software Provides basis for determining productivity savings by business process, not just by hardware use, which has proven to be virtually impossible.
- o Total System Cost - Shows all information costs, not just hardware and software by technology category, i.e., MIS, micrographics, word processing, etc.

## BASING OA REQUIREMENTS ON INFORMATION

Virtually all the analytical tools to perform OA information requirements are currently available in existing methodologies. However, as pointed out earlier, not one of them is adequate to the total task. The solution is a composite methodology using the best and strongest features of what already exists, certainly not a novel approach. The following methodology attempts to lay out the steps which lead to a bonafide information-based OA systems architecture.

### STEP 1 Defining the Information Architecture

Planning. The starting point for an Information Architecture is the top of the organization. A top-view is essential for providing a business view of the organization in terms that non-technical personnel understand. This top-view must be maintained throughout to insure clear vision by those responsible for goals, strategy, and operations of the business. The technician's view rarely coincides with management and should not be used to bias this important first step.

Clearly, a BSP approach as espoused by IBM is appropriate to establish this top view. It is business process-oriented and provides a relationship between major processes and data needed to support the processes. It further provides designations as to who in the organization is responsible for both processes and data, and to what degree they share responsibility. This step is critical to clearing up any questions as to who is responsible for what and for educating the organization on how functions and responsibilities are actually apportioned.

Because of its business process orientation, it focuses on the professional staff of the organization instead of the administrative or clerical staff, which seem to be overemphasized in OA studies. Since OA grew out of word processing, there is an overwhelming urge to use typing statistics to justify OA systems while ignoring the capabilities which facilitate computing, analysis, and communications in support of the professional staff, which by the way typically comprise 95% of an organization from a salary point of view.

The net result is a process to data class model, or an Information Architecture. It can simply be represented in a matrix that shows which processes use or create data classes, or expanded to include high-level entity models of information used the the organization. Having such an architecture provides a reference model to refer back to insure

that implementation is sound and on track. Top management must subscribe to this model. If they do not, there is no top-view.

Process Decomposition. Once a top-view is established, a methodology such as Yourdon-DeMarco structured analysis should be used to decompose processes and document data flows. Each of the processes in the Information Architecture should be decomposed to the level that an identifiable organizational entity or a person can be assigned responsibility for the process at each level. Keying the decomposition process to actual organizations simplifies the process and promotes understanding. For example, if vice presidents comprise the second tier in an organization, then top level processes are assigned to that officer who has primary responsibility for the process, even though others may share in it at lower levels. If marketing is the major top-level process, it can be decomposed into market analysis, advertising and promotions. Advertising could be decomposed to the next level to ad design and ad budget. Note here that the ad budget, even though a process under the major process "Marketing," could be assigned as a subprocess to "Budgeting" under the Controller without violating the decomposition process. This procedure would then reflect the true interplay among processes as one moves down through the organization to its lower levels. Process decomposition is an extremely critical step in analysis and should be carefully managed to insure that all processes at each level are effectively identified.

Responsibility for Data. Using the Information Architecture as a base, responsibility should be established for all data classes for each of the following categories and levels in the organization:

- o Creation - Where does the information originate? Who is responsible for entering it into a system acceptable form?
- o Accuracy - Who, if anyone, is responsible for verifying the data?
- o Form - Who is responsible for establishing the standards for data each time its form is changed?
- o Media - Who is responsible for determining what media is used to record the data?
- o Meaning - Who is responsible for data definitions and context?

It will not be unusual to find many of these responsibilities shared by several organization elements, people, and levels in the organization. It is the data administrator's responsibility to keep track of these things to insure complete and coordinated action when data requirements change.

### STEP 2 Define Baseline Workload

Once the Information Architecture is built, it will be necessary to capture workload data to properly configure the OA system with needed

capabilities and capacities. One approach is to use Job Content Profiles as a basis for estimating amount of information handling and processing work being performed broken out by categories.

The people in an organization can be categorized as one of the following:

- o Executive Level - The Chief Executive Officer and the individuals occupying line and staff positions immediately below the CEO. This may vary by organization, but it should not be difficult to identify that tier which is considered to be the executive level. People at this level normally collect information prepared by others in the organization and disseminate decisions. Most of their information activity is in face-to-face exchanges with subordinates and others.
- o Management Level - Division level managers, normally at a level immediately below the executive level and who have major staff and operating responsibilities. Managers spend large portions of their time synthesizing information and preparing it for the executive level. Information is received from the professional level in somewhat summarized fashion and subjected to management scrutiny before being passed upward. There is much human interaction through meetings and calls to collect and disseminate information upward and downward.
- o Professional Staff - Professionals, including lower level working managers who have staff or operational responsibilities. Also includes support staff not serving in an administrative/clerical capacity. This level works most closely with the data base. Their activity is the analysis of detailed information and the formulation of recommendations for the organization. There is a heavy exchange of information laterally as well as upward as masses of information are sifted, analyzed and prepared for presentation. Original text entry/creation is typically one of the major tasks at this level.
- o Administrative/Secretarial - Secretaries and personnel who operate the information processing and distribution system. At this level, information is refined and produced in final form. Tasks include typing, filing, printing, copying and distributing.

Interviews and surveys are conducted at each level, the objective being the identification of information related tasks and quantification of how much time is spent performing each task. Such tasks include text entry (typing), retrieving documents, drafting documents, attending meetings, talking on the telephone, proofreading, data entry, copying, or delivering documents. The types of tasks and the amount of time spent on each will vary of course from level to level and

even within levels. The methodology calls for the profiling of personnel in the organization at each level so that average times spent on tasks can be calculated.

Once average times are calculated, the tasks can be analyzed for how much workload is represented by time spent. For example, if one hour each day is spent drafting correspondence, and a drafter can produce 2 typed equivalent pages each hour, than a weekly drafting workload of 10 pages each week can be inferred. If a secretary types for 2 hours each day and a rate of three finished pages can be determined, the workload is six pages each day, or 30 pages in a week, or an annual output of 1500 pages based on a 50 week workyear. Each page of original typing can be expected to be printed 2.5 times before the final product is attained, resulting in a printer workload figure of 1500 pages X 2.5 = 3750 pages annually for each secretary. Phone calls, meetings, spreadsheets, data entry transactions and other tasks can be determined the same way.

The Data Flow Diagrams and other documents developed in STEP 1, Defining the Information Architecture, will be invaluable in determining data flow volumes, content, timing, media and form. Each data flow should be analyzed from all three information dimensions. Estimates or actual traffic data should be derived to reveal data flow volumes, number of transactions over time, applications by type and any external communications requirements.

### STEP 3 Refine the Information Architecture

Armed with the data from STEPS 1 and 2, you should be able to drive the Information Architecture down to a more significant level of detail. Three subarchitectures are meaningful at this point: a data architecture, an applications architecture and a data distribution architecture.

The Data Architecture. If entities were defined in STEP 1, the next step is in determining the relationships among the entities. The result is predictably an entity relationship diagram which documents how each entity acts upon other entities. The next step is to decompose the entities down through its composite data classes down to the data element level so that a data element has a hierarchical relationship with only one data class. Once at the data element level, data dictionaries become handy tools to keep track of elements and their parent classes.

The Applications Architecture. This architecture reflects relationships among data and relates the relationships to office automation functionalities. Such relationships are shown in the Figure 4.

The Data Distribution Architecture. With multiple tiered system architectures, e.g., host mainframe, intermediate host distributed processor, and workstation, the distribution of data at each of these levels becomes a key management decision. No longer is the focus only on the mainframe host, for there are DBMS and other sophisticated capabilities in the intermediate host and workstations as well, these being office automation components. Hence the distribution of

## The Applications Architecture – Relationships Among Data

| Relationships                           | OA Function                |
|-----------------------------------------|----------------------------|
| o Among Text Documents                  | Word Processing            |
| o Within and Among Lists<br>or Records  | DBMS                       |
| o Within and Among Matrices             | Electronic<br>Spreadsheets |
| o Among offices and people              | Electronic<br>Mail         |
| o Between offices and<br>host computers | Communications<br>Networks |

Figure 4

**data among these tiers should be based upon:**

- o The need to share...
  - Among all offices,
  - Within the department
  - Not at all.
- o Size of the Data Base
- o Processing Power Required
- o Security Requirements
- o Availability of Communications

### STEP 4 Configuring the Office Automation System

There are two environments which must be dealt with:

- o The Workstation Environment
- o The Host Environment

The architectural configuration should be viewed from these two environments. (Figure 5) The

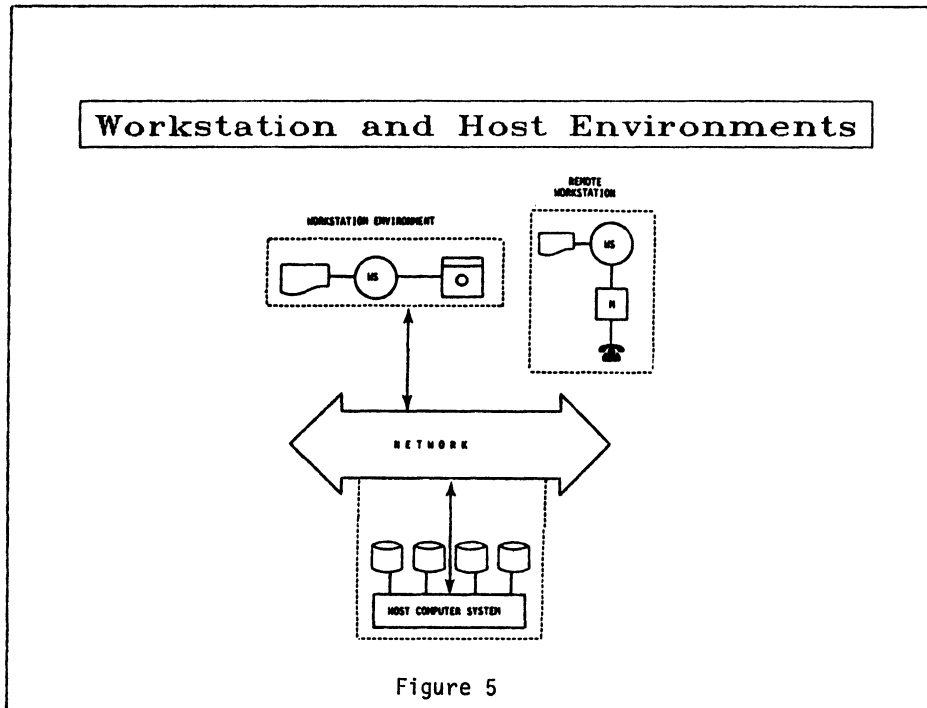


Figure 5



workstation environment is defined as the multiplicity of functions which can be performed in a standalone mode using the workstation as one would a dedicated personal computer or word processor. In this capacity it provide processing capability to the user for applications such as spreadsheet, data base management, word processing, graphics and others. If so equipped, the workstation environment may have access to the host environment through a communications capability, i.e., the hardware and software necessary to allow the workstation to emulate a terminal. In this mode, the workstation then functions as a gateway to electronic mail systems, data bases and software resident in the host environment.

Access to the Host Environment gives the user greater capacity and power as well as communications throughout the host environment. The primary or intermediate hosts often have more powerful software, greater memory and storage capacity and the ability to share scarce resources. Further, this environment serves as the gateway to other hosts for electronic mail and other applications.

The configuration of the workstation environment is based on the requirements analysis performed in STEP 3. Standard configurations should be established for workstations at each level in the organization from executives to professional to secretarial. Generally, executive configurations should be fully PC capable with emulation capability to both intermediate and primary host for electronic mail and limited data base access. Management and professional workstations should be PC-capable with a word processing, spreadsheet and DBMS applications available at PC level or intermediate host level. Reliance on the primary host for DBMS, 4GL and other capabilities should be dependent on specific needs for such capabilities, not because it is convenient or always done that way. The amount of processing at this level distributed to the workstation or intermediate host should be maximized. Finally, secretarial workstations should be full-function PC configurations, but with access to facilities at both host and workstation levels for document finishing, such as laser or letter quality printers. Applications software should be restricted to intermediate host so as not to tie up mainframe resources doing administrative tasks.

Once workstation quantities are determined by level, intermediate host, mainframe and communications configurations to support the workstations can be designed.

### **OA SUCCESS FACTORS AND ISSUES**

By taking the broad view of office automation across the Information Spectrum, it becomes clear that office automation systems have impact across total organizations, in particular:

- o Users
- o Development Process
- o Information Management
- o Productivity

### User Factors and Issues

The advances in technology and the microcomputer explosion have created a pervasive awareness of office automation among the general population. In the past, users tended to remain remote from the computing environment. Now, they want an active role in the automation of information relevant to their jobs. Office automation by its very nature delivers computing power to users and one can expect that interest will grow as more system capabilities become available. In some cases, user awareness at the workstation level will exceed that of the MIS professional. Particular attention is needed to carefully manage this awareness and interest to insure that they are channeled into activities which are constructive to the overall information management system and do not create troublesome incompatibilities in the future. In-house user groups provide excellent outlets for this enthusiasm and interest and offer the MIS manager an opportunity to be aware of what is happening at that level.

Productivity among the professional staff remains an issue. OA has yet to come up with a universally accepted method of measuring productivity of professionals. The method prescribed in this paper is but one of many which attempt to do so to the satisfaction of those who manage resources. While it is intuitively obvious to those who have extensive office automation, others will find precise measurement of productivity improvements an elusive entity. The impetus to do so may fall by the wayside as more methodologies are developed to show the impacts of office automation as a competitive business advantage. As a minimum, however, organizations must resist the temptation to use word processing productivity techniques to measure office automation productivity. It is professional productivity which has the greatest payoff.

### Developmental Process Factors and Issues

Microcomputer workstations have finally brought legitimacy to end-user computing. While end-user mainframe languages have been with us for awhile, their introduction into user organizations has been slow. Information Centers were introduced to provide user assistance in using these end-user languages. But it was not until now, with microcomputers proliferating in the office that the Information Center takes on real meaning. No longer did the user have to endure the long, traditional software development process. Here was total control over information right in the user's office with user information and user software packages. But it still is not "all" information, for there is still "corporate" data stored elsewhere, typically on the mainframe. This is where the marriage of end-user computing components, micros and mainframe take place. Organizations which do not recognize the importance of Information Centers in cultivating and facilitating end-user computing will not be able to exploit their investment in office automation.

In larger organizations, information is not managed well outside the the immediate unit which creates or uses it. Hence the notion of creating information once, and using it many times throughout the organization is difficult to achieve. This notion, which we will call "leveraging information," should be a data base design and organizational goal in establishing office automation systems. Without it, users will use their new found computing power to establish yet more data bases which are not synchronized with others and leading to more inconsistency on critical organization data.

A companion issue is how to administer data at the workstation level. How many rules are enough to establish control, consistency and sharing? What standards should be established for file formats, document formats or documentation? Too many rules frustrate users and are probably not enforceable. Too few perpetuate the problem of incompatible data.

Office automation should be viewed from the perspective of the entire Information Spectrum and not as an individual segment in the information pathway. The interaction of the various technologies is what produces the most effective and efficient set of OA system capabilities. The basis should always be information, that precious commodity for which all these system components are being put together. Current methodologies are available to establish information based requirements for OA and should be used. Information requirements should then be combined with office workload requirements to determine the optimum configuration for office systems. Finally, everyone should recognize that equipment alone does not improve productivity or provide the competitive edge. It takes people who are organized for the specific purpose of making the system do its job. Then the OA Myth, "Office Automation improves the productivity of people..." can be replaced by the truth, "People improve the productivity of Office Automation."



# DEVELOPMENT OF AN IN-HOUSE TRAINING PROGRAM FOR ALL-IN-1

Nancy R. Pflanz  
COMPUTER TECHNOLOGY ASSOCIATES, INC.  
Albuquerque, NM

## ABSTRACT

This paper defines the reason for an in-house training program, and the steps involved in implementing a working training program. Training room facility requirements and costs; instructor qualifications; course type, frequency, and content are included in this presentation, as well as insight into the background prompting the development of the in-house training program. Further, to adequately represent and qualify an in-house training program, much of the data presented has been specifically and accurately extracted from a current, successfully-operating training program. Additional recommendations and/or generic-type suggestions are also presented for consideration.

## 1.0 BACKGROUND

The Air Force Operational Test and Evaluation Center (AFOTEC) initiated a Pilot Project, to explore the use of computers for improving productivity and effectiveness in Operational Test and Evaluation (OT&E) management and conduct. This AFOTEC Pilot Project is being accomplished through the installation of a limited-scale, distributed, integrated, multi-function computer system, located at Headquarters (HQ) AFOTEC, Kirtland Air Force Base, NM; incorporation of selected functional capabilities (i.e., office automation, information management, and data processing functions); and measurement of system performance, including the effects on productivity, and other related factors that result from the availability and use of the system.

The training program includes formal classes and consists of ALL-IN-1\* classes in Desk Management, Word Processing, Electronic Mail, Graphics, OTEMIS\*\* (Operational Test and Evaluation Management Information System), RUNOFF, and an Electronic Spreadsheet--FLOW\_Calc\*\*\*. In addition to formal classes, time is spent in tutoring (on an individual basis), in answering questions, and in resolving trouble calls. Due to the continual turnover in Air Force personnel, there is a demand for a continuous training program.

\*Trademark of Digital Equipment Corporation (DEC).

\*\*Air Force Program.

\*\*\*Trademark of General Research Corporation (GRC).

## 2.0 SYSTEM DESCRIPTION

The AFOTEC Pilot Project computer system consists of a Digital Equipment Corporation (DEC) VAX 11/780, with 16 Mb of memory. Disk storage consists of four RA81 Winchester disks, with 456 Mb per drive. There are 112 terminal ports, of which 12 are connected to modems (to allow system access to remote users, and to allow dial-out access to remote systems by local users). The types of terminals currently in use include the DEC VT100s, VT102s, DECmate Is and IIs, VT125s and VT241s; and Zenith 2-100 and 2-150 microcomputers, with VT100 emulators. The most commonly-used (local) printer is the DEC LQP02, of which 24 are currently available.

AFOTEC's Office Automation System has been modified extensively from the DEC-delivered product, to include a graphics interface, a third-party spreadsheet, an on-line documentation and information system, and an external network and communication access, as well as customer-tailored additions/modifications to some of the standard ALL-IN-1 menus. Figure 1 illustrates the ALL-IN-1 Main Menu options.

### A F O T E C

### Automated Office Professional Workstation

|                                 |                           |
|---------------------------------|---------------------------|
| WP Document Processing          | GR Graphics Applications  |
| EM Electronic Mail              | CO Communications         |
| DM Desk Management              | PH Phone Utility          |
| IM Information Management       | CP Change Password        |
| BI Business Applications        | VI VAX System Information |
| Pd Program Development          |                           |
| PS Profession Specific          |                           |
| NU Instructions to the New User |                           |
| EX Exit ALL-IN-ONE              |                           |
| LO Leave Workstation            |                           |

Enter selection and press RETURN  
OR press the HELP key for help.

Figure 1. AFOTEC Main Menu Options.

Approximately 550 people presently have access to the AFOTEC VAX 11/780 computer system. The average user load, however, is 30. As of 01 October 1984, 44 percent of the users' log-on time was spent in word processing; and 26 percent was spent in Electronic Mail. Figure 2 summarize these statistics.

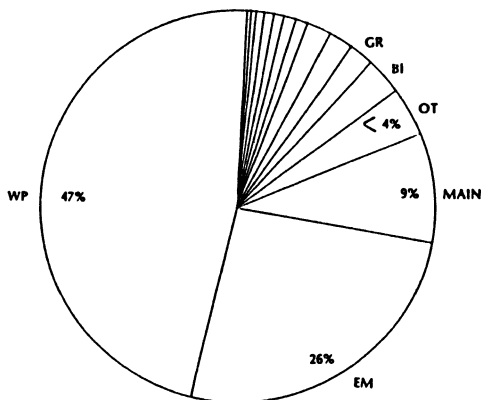


Figure 2. Utilization Of ALL-IN-1 Options By AFOTEC Personnel.

### 3.0 SELECTION OF COURSE SUBJECT MATTER

Once the AFOTEC Office Automation System was installed, some basic classes needed to be taught, in order to instruct the general user. The user community, as a whole, has an extensively-varied background of computer experience and knowledge, ranging from the experienced programmer, to an inexperienced clerk/typist (who has never seen a computer terminal). Basic classes need to be taught first, so the inexperienced user would not be intimidated.

Two types of classes are presently being taught in the AFOTEC training program:

- o Beginning Classes--The Beginning classes include the basic office automation tools, which are the most-frequently used office automation functions, as well as being relatively easy and straightforward (and, generally, do not overwhelm the new user). The three 'basic' classes include the following:

- (1) Desk Management--This course is selected as the first 'basic' to be taught, primarily to familiarize the user with the keyboard (especially the mini-keypad). In addition, the user becomes accustomed to procedures for filling out 'forms' and in their use.
- (2) Word Processing--This function, the second 'basic' to be taught, is fundamental to office automation, involving the use of a full-screen editor. Familiarity with this editor also assists in the understanding of the third basic, Electronic Mail (EM) (since EM also requires use of this editor).

- (3) Electronic Mail/File Cabinet Maintenance--This easy-to-use function provides the primary mechanism for interoffice communication (for such things as memos, notices, reports, official correspondence, etc.), needed by all users.

and

- o Advanced Classes--To date, the advanced classes include the following:

- (1) Graphics--The Graphics package is DEC's POLYGRAFIX, supporting a wide range of graphics applications. The graphics class only covers the Graphics Editor, the Slide Projection system, and the Data Plotting Package. The Graphics Editor allows the user to create and edit pictures interactively. Pictures created can then be stored in files, for use with the Slide Projection System. The Data Plotting Package performs interactive file and data manipulation for graphics plotting; and these can also be stored in files, for use with the Slide Projection System.
- (2) FLOW\_Calc--is a third-party software package, providing an "electronic spreadsheet" function. Worksheets can be constructed, stored, retrieved (back to the screen) for updating, and printed (hardcopy). The FLOW\_Calc class introduces the student to a spreadsheet: it instructs the student in procedures for constructing one, and 'walks' the student through the various commands, using the newly-constructed spreadsheet.
- (3) Digital Standard Runoff (DSR)--is a text-formatting facility. The Runoff class covers the basic DSR commands, which are entered directly into the text, and allow the user to control the format of word processing output.
- (4) OTEMIS--The OTEMIS program is a specialized course, developed by AFOTEC, and contains information about test programs. The class shows the user the operation and features of OTEMIS.

By request, new courses can be added to the schedule, in order to satisfy immediate or short-term needs. If there is continued demand for the same course, it can then be added as a regular class.

#### 4.0 SELECTION OF TRAINERS

The selection of trainers is a critical consideration. Two of the most common choices are: an engineer, or an in-house staff trainer. While most engineers know the subject matter proficiently, they generally lack the necessary skills and patience to relate the information to a new user. Further, most engineering professionals have a tendency (unintentional) to use terminology that is unfamiliar to the less-knowledgeable individual. Further, the engineer occasionally has difficulty understanding such an individual's questions. On the other hand, someone with an educational background or with teaching experience is more adaptable to this environment. Educators are generally more aware of the different levels of understanding, and realize that not everyone can listen to information, absorb it, and correctly understand it. Generally, there will be a variety of students in a class: some with computer backgrounds, and some without; and some who like to work with computers, and some who are afraid of them. A trainer with teaching experience is usually better adept in accommodating the varied understanding levels of all the students.

Engineers are also (generally) more expensive than an in-house trainer with an educational background and, additionally, are not usually content in a permanent role of a trainer. An in-house staff trainer could keep the cost of the program minimal; and the experience, patience, and dedication of a trainer with an educational background can practically ensure the success of an in-house training program. Another advantage of an in-house trainer is that he/she would also be available, when not in class, to answer any questions about ALL-IN-1, whenever users encounter difficulty or whenever problems occur.

When one (or several) person(s) is involved in the role of trainer (solely), a morale problem could arise: repetitive teaching of the same classes can lead to boredom and dissatisfaction with the job. Teaching under an in-house training program is different than teaching in a university environment: in a university program, the same class might be taught more than once in a day; but the students are moved through the subject matter daily. In ALL-IN-1 training, the same class must be taught over and over. Whether it is taught once a week or once a month, the subject matter is always the same; and, after teaching the same material many, many times, a class can become extremely monotonous for the trainer. In order to break the tediousness of training, backup personnel are essential. A rotation schedule should be developed, so that the trainer is taken away from the classroom for a specified time period, to perform other, non-training tasks; and a backup trainer performs the training. With such non-training tasks occurring periodically in the trainer's schedule, the monotony in training will subside considerably.

#### 5.0 THE DEVELOPMENT OF CLASS NOTES AND LESSON PLANS

The development of a class requires several steps, to be performed in a specific order:

- (1) Learning the material--Depending on the trainer's background, time needs to be set aside for the trainer to learn the material and to practice (hands-on experience) on the system. From this, the trainer can develop class notes.
- (2) Developing class notes--Class notes are basically for the trainer's benefit, and contain notes that the trainer needs to review in order to properly present the material. Class notes generally contain detailed information about the subject, not all of which will be related to students; but, if questions arise or if users are having difficulty while working on the system, the trainer will have a larger (detailed) pool of knowledge on which to rely in answering questions or in solving problems. If training is being performed on a large scale, backup personnel are essential. Not only will these class notes be a tremendous aid to backup trainers, the lead-time for alternate trainers would also be greatly reduced, since the class notes and lesson plans will have been previously prepared by the primary trainer. (Theoretically, backup trainers should be able to step in at the last minute and conduct the class.)

The backup personnel (especially if they do not have an educational background) should perform a 'dry run' of each class they are going to teach, under the direction of the permanent trainer. This would allow the trainer to critique the backup's teaching techniques, so that the quality of instruction is comparable to the trainer.
- (3) Constructing lesson plans--From the class notes, a lesson plan is derived. Lesson plans contain a detailed outline of the topics to be taught, and the order in which the material will be presented. The lesson plan can then be used as a guide during teaching. It establishes consistency in covering the material, as well as assuring consistency between trainers and their backups. Table 1 illustrates an example of a brief lesson plan, for the Word Processing class.
- (4) Preparing handouts--Handouts covering the subject matter can be distributed to the students. Such handouts supplement the material being covered, thus making the subject matter easier to understand. It also reduces the

amount of notes students would need to take, allowing them to concentrate more fully on the instructor's presentation. Handouts also serve as a quick-reference guide for the student, once he is on his own with the system. Since the DEC ALL-IN-1 User's Guides are not very convenient to use, and most ALL-IN-1 users do not have the time to read the entire manuals, it is difficult and time-consuming for the student to locate the information he needs. Consequently, the handouts serve as quick and convenient substitutes for the User's Guides. Table 2 illustrates an example of the first page of a brief handout for the Word Processing class.

TABLE 1. SAMPLE LESSON PLAN FOR WORD PROCESSING

1. Mail Message
  - a. File word processing practice document using <FM> option from the electronic mail <EM> menu
2. <WP> Word Processing Option
  - a. <C> Create a document
  - b. <CL> Create long form
    - (1) Header information
    - (2) Setup--available choices (4 on mini-keypad)
  - c. <SEL> Select a document
    - (1) Recognition
    - (2) Selection of word processing practice document
  - d. <E> Edit option
    - (1) Leaflet--WPS editor
    - (2) Handout
    - (3) Movement within document
    - (4) Deletion of items from document
    - (5) A character string Search (and Replace)
    - (6) Addition of text
    - (7) Centering of text
    - (8) Cut and paste function
      - (a) Moving a selected portion of text
      - (b) Canceling a selection
      - (c) Copying a selected portion of text
    - (9) Uppercase and lowercase
    - (10) Transposition of two characters
    - (11) Re-'painting' of the screen
    - (12) Insertion of date and time
    - (13) Pagination
    - (14) Help
    - (15) Insertion of a document into text
    - (16) Desk calculator access
    - (17) <GOLD M>
    - (18) Ruler setting
    - (19) Exiting from the document
  - e. <D> Delete option
  - f. <T> Display option
  - g. <P> Print option--Available printers
  - h. <I> Document index option
  - i. <SEA> Search file cabinet option
  - j. <SC> Spell check option
  - k. <WF> WPS file cabinet
  - l. <RP> Read protect option

(5) Teaching the class--Completion and utilization of the above steps and materials promotes effective class instruction, in addition to providing useful preparation, guidelines, and notes for teaching the class.

TABLE 2. SAMPLE HANDOUT FOR WORD PROCESSING CLASS

1. To Move Around The Document
  - a. Down Arrow < >--moves the cursor down one line.
  - b. <GOLD # >--moves the cursor down as many lines as specified (i.e., #).
  - c. Up Arrow < >--moves the cursor up one line.
  - d. <GOLD # >--moves the cursor up as many lines as specified (i.e., #).
  - e. <Advance> key--moves the cursor to the right one position; if the cursor is at the end of a line, pressing this key will move the cursor to the first position of the next line.
  - f. <Back Up> key--moves the cursor to the left one position; if the cursor is at the beginning of a line, pressing this key will move the cursor to the last position of the previous line.
  - g. Distance keys (blue keys)--These five keys are used to move through the document after the direction has been set (through first pressing the <Advance> or <Back Up> key):
    - (1) <Sent>--moves the cursor to the beginning of the next (or previous) sentence (looks for a period ".").
    - (2) <Word>--moves the cursor to the beginning of the next (or previous) word.
    - (3) <Para>--moves the cursor to the beginning of the next (or previous) paragraph.
    - (4) <Line>--moves the cursor to the beginning of the next (or previous) line.
    - (5) <Page>--moves the cursor to the beginning of the next (or previous) page.
  - h. <GOLD B>--moves the cursor to the bottom of the document.
  - i. <GOLD T>--Moves the cursor to the top of the document.
  - j. <GOLD Next Screen> (<GOLD Advance>)--moves the cursor forward 21 lines.
  - k. <GOLD Previous Screen> (<GOLD Back Up>)--moves the cursor back 21 lines.
2. Delete Items From Document
  - a. <Rub Char Out> key--deletes the character to the left of the cursor.
  - b. <Rub Word Out> key--deletes the word to the left of the cursor.
  - c. <Del Char> key--deletes the character on which the cursor is positioned.
  - d. <GOLD Del Char> key--replaces the last character that was deleted with <Del Char>. (This will also work if the <Rub Char Out> key was used.)
  - e. <Del Word> key--deletes the word to the right of the cursor.

TABLE 2. SAMPLE HANDOUT  
FOR WORD PROCESSING CLASS (Continued)

- f. <GOLD Del Word> key--replaces the last word that was deleted with <Del Word>. (This will also work if the <Rub Word Out> key was used.)
- g. <GOLD Rub Line> (<GOLD Rub Char>)--deletes the line to the left of the cursor.
- h. <GOLD Rub Sent> (<Gold Rub Word Out>)--deletes the sentence to the left of the cursor.
- i. <CTRL U>--deletes everything to the beginning of the line.
- 3. Adding Text--Position the cursor where text is to be added, and begin typing.
- 4. Reformatting Text--When white diamond (<>) appears at the end of a sentence, the text needs to be reformatted: use <GOLD Wrap Paragraph> or <GOLD Para>.
- 5. Searching For Character Strings (And Replacing)
  - a. <GOLD ,>--searches for a specified character string.
  - b. <GOLD .>--continues search.
  - c. <GOLD ;> or <GOLD S>--(global search and replace) searches for a specified character string and replaces it with another specified character or character string.
- 6. Centering Text--<GOLD C>--The cursor must be positioned to the right of the text to be centered.
- 7. Moving or Copying Sections of Text
  - a. Cut and Paste--moving text
    - (1) <Sel> key--Selects the text to be moved.
    - (2) <Cut> Key--Removes the selected text and stores it in the "Paste" buffer.
    - (3) <Paste> Key--Inserts the contents of the Paste buffer at the cursor position.
  - b. Canceling a choice  
After text has been selected, if removal is not desired, press <GOLD Reset> <GOLD Sel>.
  - c. <GOLD Copy> (<GOLD Cut>)--copying Text
    - (1) <Sel> key--Selects the text to be moved.
    - (2) <GOLD Copy> (<GOLD Cut>) Key--Copies the selected text and stores it in the "Paste" buffer.
    - (3) <Paste> Key--Inserts the contents of the Paste buffer at the cursor position.
- 8. Changing Cases
  - a. <Upper Case> Key--changes one character from lower to upper case.
  - b. <GOLD Lower Case> (<GOLD Upper Case>)--changes one character from upper to lower case.
  - c. To change a word, line, or sentence:
    - (1) Press the <Upper Case> key or press <GOLD Upper Case>, depending on the specific need
    - (2) Press the <Word>, <Line>, <Sent>, or <Down Arrow> key
    - (3) Turn off by pressing either the <Advance> or <Back Up> key.

TABLE 2. SAMPLE HANDOUT  
FOR WORD PROCESSING CLASS (Concluded)

- d. To change the case of a section of text
  - (1) Select the range (<Sel> and a distance key)
  - (2) Press either <Upper Case> or <GOLD Lower Case>, depending on whether the selected text is to be changed to upper case or to lower case.
- 9. Changing Margins
  - a. <GOLD R>--the default value for the left margin is 1; for the right margin, 70.
  - b. To change the left margin--move the cursor to the desired position for the left margin and type a <W>.
  - c. To change the right margin--move the cursor to the desired position for the right margin and type a <R>.
  - d. To save a ruler--Once the left and right margins are set (see b. and c. above), press <Shift S>, and type a number from the main keyboard (between 0 and 9). Up to 10 rulers can be saved.
  - e. To recall a ruler--Type the appropriate number (the number under which the desired setting was saved).
  - f. To exit from the ruler--press the <RETURN> key.
- 10. Transposing Two Characters
  - a. <GOLD Right Arrow>
  - b. <GOLD Enter>
- 11. Redo-ing the Screen--<CTRL W>
- 12. Inserting Date and Time--<GOLD >
- 13. Paging
  - a. <GOLD Page>--automatic, to see where the page break will occur
  - b. <GOLD N>--inserts a new page (puts in a permanent page marker when it is used)
  - c. <GOLD P>--Puts in a temporary page marker; the printer ignores this type of page marker.
- 14. Help Screen--<GOLD H>
- 15. Inserting Document or Message in Text--<GOLD G>
- 16. Using the Desk Calculator
  - a. <GOLD #>
  - b. The result is in the paste buffer-(Bug)
- 17. Using the Editor Menu--<GOLD M> brings up the editor menu, which includes:
  - Calendar Management
  - Desk Calculator
  - Enter DATATRIEVE
  - Read Next Message From Inbox
  - List Of ALL-IN-1 Users To Paste Buffer
  - Update Document Header
- 18. Updating Document Header Information--<GOLD U>
- 19. <GOLD F> and <GOLD Q> (or <GOLD K>)
  - a. <GOLD F>--saves changes and returns to the WP Menu.
  - b. <GOLD Q> (or <GOLD K>)--does not save changes; but also returns to the WP Menu.



## 6.0 TRAINING SCHEDULES

In the AFOTEC training program, each class is allotted 2.5 hours, with a 5-to-10-minute break during that period. There are three basic ALL-IN-1 classes: Desk Management, Word Processing, and Electronic Mail/File Cabinet Maintenance). These classes are taught consecutively, one each day for three days. The teaching frequency of this series is contingent upon user demand. The Advanced classes (Graphics, RUNOFF, FLOW Calc, and OTEMIS) are also offered relative to user demand. Generally, the basic classes will be taught more often than the Advanced. An example training schedule is illustrated in Table 3.

TABLE 4. SAMPLE TRAINING SCHEDULE  
April 1985

| DATE   | TIME      | SUBJECT                                               |
|--------|-----------|-------------------------------------------------------|
| 01 Apr | 0900-1130 | ALL-IN-1 Desk Management                              |
| 02 Apr | 0900-1130 | ALL-IN-1 Word Processing                              |
| 03 Apr | 0900-1130 | ALL-IN-1 Electronic Mail/<br>File Cabinet Maintenance |
| 04 Apr | 0900-1130 | RUNOFF                                                |
| 05 Apr | 0900-1130 | FLOW Calc Spreadsheet                                 |
| 15 Apr | 0900-1130 | ALL-IN-1 Desk Management                              |
| 16 Apr | 0900-1130 | ALL-IN-1 Word Processing                              |
| 17 Apr | 0900-1130 | ALL-IN-1 Electronic Mail/<br>File Cabinet Maintenance |
| 18 Apr | 0900-1030 | OTEMIS                                                |
| 22 Apr | 0900-1130 | GRAPHICS                                              |

The basic ALL-IN-1 classes (Desk Management, Word Processing, and Electronic Mail) should be taken in the order as listed. All new VAX users MUST attend the Desk Management course.

Individual tutoring is available on request for any of the above subjects. For appointments, call the trainer.

A course catalog is available through the VAX INFORMATION option in ALL-IN-1. To access the catalog, type VI at the main ALL-IN-1 menu; and select the document "Current Available Training".

Students who have enrolled in a class and subsequently cannot attend should notify the trainer of the cancellation as soon as possible. This will provide an opening for others who are waiting to enroll for the class.

Students wishing to enroll in a class, or who have questions regarding the training/tutoring should contact the trainer.

## 7.0 TEACHING TOOLS AND TECHNIQUES

There are several teaching tools and/or techniques that can make the trainer's job easier. One major consideration is the class location and facilities. Because a 'hands on' program is proven more effective than a 'lecture-only' format, a special training room is recommended, with terminals for students and instructor. When the room is not being used for training or tutoring, these terminals will be accessible to users, for performing their daily tasks.

The physical layout of the room is also of vital importance. A suggested training room layout, similar to the one supporting the AFOTEC training program, is illustrated in Figure 3. At present, the AFOTEC training room (measuring 15 X 24 feet) accommodates seven terminals (DEC VT100s and VT102s), all of which have access to the VAX: six for student use, and the seventh for the instructor's use (located in the rear of the classroom). A large screen is located in the front of the room, with a video projector positioned in the center of the room. This projector is connected to the instructor's terminal, so that the students can see the commands as they are being entered, thus verifying the correctness of their own entries. Through his/her location at the rear of the room, the instructor has visual access to the students' terminals, thus affording greater awareness and responsiveness to the needs/problems of each student. The instructor is able to offer immediate assistance to the students, thereby maintaining a unified class. (The immediate availability of the instructor reduces/eliminates the frustration that can occur when one or more student becomes disoriented or confused with the procedures or instructions.) Further, the instructor is always available outside of the classroom, for responding to questions and/or problems that may be encountered when students apply what they have learned, during the performance of their routine, daily tasks.

Another teaching aid, especially with a training room setup like the one described above, is proper lighting. A dimmer switch is highly recommended, in order to accommodate the need for viewing the projection screen, as well as the need for seeing the characters on the keyboards, reading handouts, and taking notes.

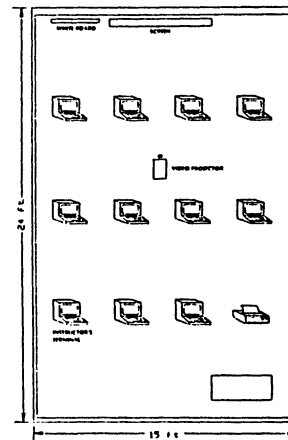


Figure 3. Training Room Layout.

In setting up a training room, the possible 'traffic' hazard existing by the presence of the many wires necessary for the terminals and the projector equipment must not be overlooked. In the AFOTEC training room, this problem was avoided by keeping all the wires along one wall, away from the major traffic. In addition, power poles have been used through which to run the terminal communication lines. Where the presence of wires in the traffic area could not be avoided, rubber strips have been placed over the wires, protecting the wires while concurrently eliminating a potential hazard.

A switchbox, which will allow the instructor to project (on the screen) the display from any of the students' terminals, can be a tremendous teaching aid, especially in projecting a problem being experienced by one of the students and, subsequently, explaining (to that student and to the whole class) the reason for the problem, and the solution for exiting out of it and for avoiding it in the future.

Another useful item is a lighted pointer, projecting an arrow on items that need to be stressed or brought to the students' attention.

The typical cost of setting up a training room as described above (to accommodate six students--recommended) is \$27,450 - 35,450, a breakdown of which is illustrated in Table 4. Although up to 10 students could be taught without jeopardizing learning (limited primarily because of visual access to the screen), a class of 6 students is recommended (with 10 being a suggested maximum).

TABLE 4. COST OF EQUIPMENT FOR A TRAINING ROOM BASED ON SIX STUDENT TRAINEES

|                                     |               |
|-------------------------------------|---------------|
| VIDEO PROJECTOR                     | 6,000-14,000  |
| PROJECTION SCREEN                   | 1,000         |
| TERMINALS (SIX VT220s)              | 7,770         |
| PRINTER                             | 2,000         |
| FLOOR SPACE (13 FT <sup>2</sup> /Y) | 4,680         |
| FURNITURE                           | 6,000         |
|                                     | 27,450-35,450 |

As mentioned previously, handouts are very beneficial for many reasons: as a convenient reference during class; as a class guide, maintaining the order in which the material is being covered in class; in providing answers to commonly-asked questions and/or topics that were previously covered in the class; and in providing reference and guidance support when the student is working independently.

The hands-on environment (i.e., one student per terminal) is recommended, as opposed to the lecture environment, because it not only limits the number of students in a class (thus guaranteeing more individualized instruction), but also allows the student to practice, on the spot, what he is learning. In a lecture environment, on the other hand, more students can be taught; but the amount of information absorbed is minimal, especially since most of the students are unaccustomed to taking notes and recalling (absorbing) the material covered in this fashion.

The order in which the various class topics are presented must also be carefully planned, since many options are contingent on others. (e.g., it is difficult to demonstrate the "get document" option in Word Processing, if there is no document in the student's file cabinet to 'get'.) The class must be planned in such a way that it moves smoothly, reduces confusion, and makes sense to the students (even the apprehensive or slower students, who can become very discouraged and possibly less productive with an unfamiliar system).

In the AFOTEC training program, classes are taught in a specific order, to optimize the effectiveness of the program. For the first day of the beginning (basic) classes, Desk Management is taught. At the start of class, the students are shown how to login and change their default passwords. A system overview and the various menus are also presented, to provide the students with a general knowledge of what is available, in addition to that which is covered in the classroom. Some time is then spent on the Instructions to New Users option, followed by the Desk Management Menu. The primary purpose of this class, besides introducing the students to the system, is to familiarize the new user(s) with the keyboard and the necessary procedures for filling out forms.

Before the Word Processing class (the second beginning class), two electronic mail messages are sent to all of the students, both of which contain documents: one, to demonstrate the Spell Check option; and the other, to practice the word processing commands. Once the students have properly processed their mail messages, the Word Processing menu options are explained. The Create option is used to familiarize the student with several word processing options: creating a document; viewing the Create Long-form option; using the Select Document option, to select the word processing document that was sent to the students; and using the Edit option, to edit that document. The students are then issued a handout containing the word processing commands.

The last basic class (Electronic Mail/File Cabinet Maintenance) can be a very enjoyable class, even for the instructor. Prior to the class, several mail messages are sent to the students, usually containing information directly relative to the option being covered. Several more mail messages are also sent to the students during the class; and all the Electronic Mail menu options are covered.

Since the Air Force has made several additions/modifications to the original ALL-IN-1 Electronic Mail menu, an additional handout is also distributed to the students, which briefly summarizes the AFOTEC Electronic Mail options as well as the the basic ALL-IN-1 Electronic Mail options.

The last subject covered in the Beginning classes is File Cabinet Maintenance (FC), from the Desk Management menu. After looking at the menu options in File Cabinet, the last items to be perform by the students are several multiple deletes to clean out their file cabinets so that, when they have finished the three basic classes, they leave the training room with a clean user account. To assist in this cleanup process, all documents created during the three classes were filed in the folder "temp". Therefore, the delete is simplified: all file folders marked "temp" are deleted, as well as all "sent" and all "read" messages.

## 8.0 SUMMARY/RECOMMENDATIONS

The primary considerations for establishing an in-house training program include three focal points:

- (1) The Trainer's Background--A decision must be made regarding whether an engineer or an in-house trainer with an educational background is to implement and manage the program. A trainer with an educational background is recommended.

In addition, if the training program is to be performed on a relatively large scale, backup personnel for the trainer are highly recommended.

- (2) The Training Room Setup--A training room, supplied with an video projector, is a necessity. Further recommendations include individual terminals for each student in the class (6-to-10), and one for the instructor, with a switch box between each terminal and the projector.
- (3) The Environment--A hands-on program (one student per terminal), supplemented with handouts, is the most effective means of instruction of this type. The three basic ALL-IN-1 classes, Desk Management, Word Processing, and Electronic Mail, provide a solid foundation for system use, and (depending on the type of work performed by the students) advanced classes should be selected and taught only as specifically needed.

**MUMPS SIG**



A WALK THRU THE FOREST  
HOW TO FIX YOUR MUMPS TREES

Denise Simon  
Digital Equipment Corporation  
Hudson, Massachusetts

ABSTRACT  
-----

This paper talks about the general structure of DSM-11 globals and how to repair minor corruptions using the system utilities. It does not delve extensively into the global structure but discusses global creation and growth. It is assumed the audience has a general knowledge of DSM-11. After reading this paper, you should have a basic understanding of DSM-11 globals and be able to perform simple database reparations. It is assumed that the reader is familiar with the utilities mentioned and therefore detailed instructions are not provided for running these.

WHAT IS A GLOBAL?

A DSM-11 global variable or GLOBAL is a variable stored on disk. It is semi-permanent in that it exists on disk until you specifically delete it with a KILL command.

Global Creation  
-----

A global is created when it is first referenced in a SET command or by %GLOMAN, the global management utility. In the simplest case, this would be something like:

S ^X=1

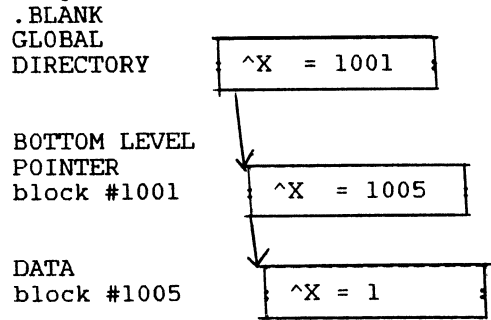
When a command like this is issued, the following events occur:

1. An entry is made in the Global Directory for the UCI in which the global is being created. This entry contains information such as collating sequence, journalling status and protection codes. The block number for the first block belonging to the global is also stored in the global directory. This is known as a POINTER to the first block.

2. A block is taken from the database and given to this global. This first block is known as a POINTER block. In the case of the above SET statement, it will contain the name of the global (^X) and a pointer to a data block. Pointer blocks can point to other pointer blocks or to data blocks. In the above case, there will be one pointer block and it will point to a data block. Pointer blocks that point to other pointer blocks are known as POINTER blocks. Pointer blocks that point to DATA blocks are known as BOTTOM LEVEL POINTER BLOCKS.

3. A block is taken from the database and given to the global as a DATA block. The data block contains the name of the global and the data.

Therefore, conceptually, you have created a global that looks like this (figure 1):



(Figure 1)

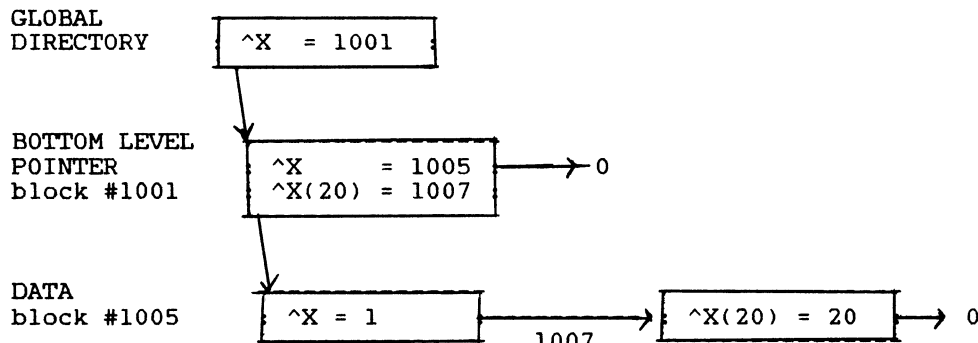
This structure is known as a TREE. Suppose you now want to add some information to this global and you do the following:

```
F I=1:1:1000 S ^X(I)=I
```

Each piece of data added to the global ^X will be added to the DATA block until it becomes full. When the next SET after the block becomes full is done, a BLOCK SPLIT occurs. What happens here is this:

1. A new block is allocated.
2. A pointer from the first data block to the second is created. This is called a RIGHT LINK POINTER. The rightmost block will have a right link pointer pointing to block 0.
3. A pointer is inserted in the pointer block that points to the DATA block.

Assume that ^X(20) was the first node in new block (after a block split). This is what the global would look like (figure 2):

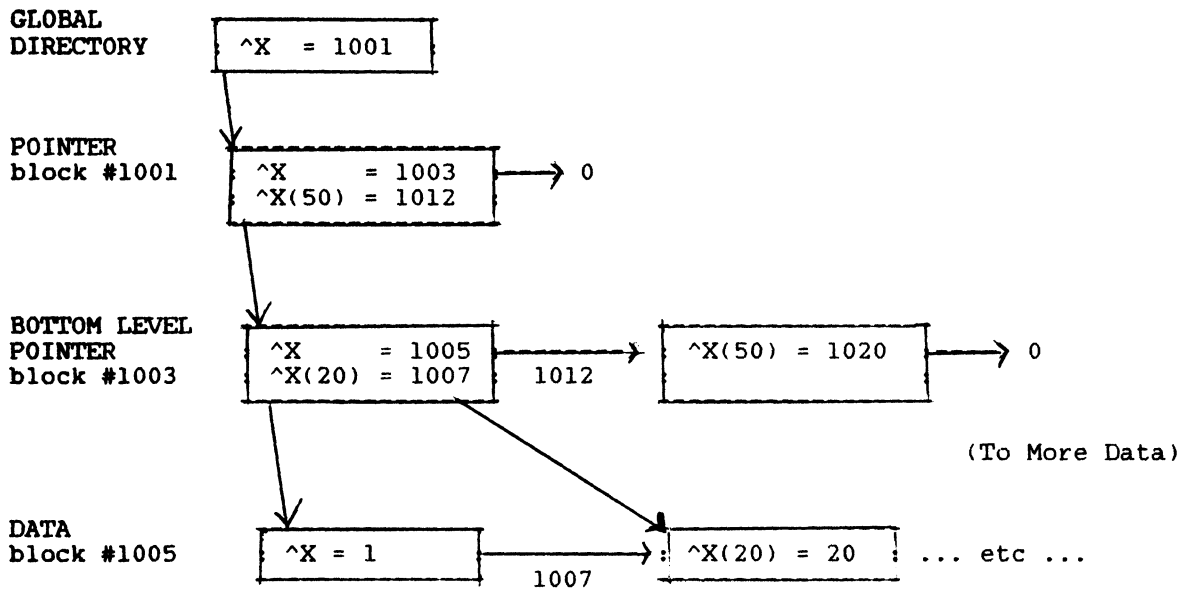


(Figure 2)

As you add more and more data, more data block splits occur and more pointers are added to the pointer block. Finally, the pointer block fills and, it too, must split. When this occurs, the following happens:

1. If the pointer block splitting is the top level pointer block, a new row consisting of 2 blocks is created. These are linked together with a right link pointer.
2. The old information as well as the new pointer that forced the split is copied into the two blocks in the new row.
3. The original pointer block is cleared of all old down pointers and down pointers to the new two blocks are added.

This means that once a global is created, the number of the first block remains the same throughout the lifetime of that global. The result of the above operation looks like this (figure 3):



(Figure 3)

This structure is very effective. You can have a global dispersed all over your database, but DSM-11 merely follows the pointers down a section of the tree to access specific portions of the tree.

Types of Blocks

Summarizing, the above, a global is made up of several types of blocks. These are:

1. A Global Directory block which contains pointers to the first block of each global in a given UCI.
2. Pointer Blocks which point to other pointer blocks.
3. Bottom Level Pointer blocks which point to data blocks.
4. Data blocks which contain the actual data.
5. Map Blocks keep track of the usage of database blocks.

"Wait a minute, Denise, you didn't mention Map Blocks", you might be saying. You are right. Let me do so now:

DSM-11 disks are separated into chunks of 400 blocks. Every 400th block starting with block 399 is called a MAP block. Each map block contains a one word entry called an ALLOCATION WORD that tells us about the status of each of the 399 blocks preceding the MAP block. It tells us if each block is free or in use and if it is in use, who owns it and who set it. The only time you have to worry about the Map block while repairing a minor corruption is if you have a hopelessly corrupted block you wish to remove from a broken global. If you do this, you may wish to go into the map and free the block so it will be available for use in the future. I'll show you a corruption later where we will have to free a block.



It should be noted in the illustrations above, that each pointer contains the name of the data node which is the first node in the block below.

The above explains how a global being created sequentially is built. If you are inserting nodes into already existing blocks, block splits occur in much the same way. The exception is that when the split occurs, some data is copied from the old (splitting) data block to the new one. This is done to allow for future insertions and thus reduces the amount of subsequent splits.

WHY DO GLOBALS BREAK?

There are a number of reasons that globals break. They can break due to hardware problems. If memory breaks or a disk goes bad, you might see problems. Corrupted system software or a bug in the operating system might be responsible for a break. If you use the VIEW command, a mistake can cause problems. In other words, almost anything can cause a problem if the circumstances are right.

HOW DO YOU KNOW A GLOBAL IS BROKEN?

<DKSER> or <DKHER> Messages

More often than not, these errors indicate a block has gone bad and must be entered in the bad block table. Occasionally, however, you will see one of these caused by a global corruption.

<DBDGD> Errors

<DBDGD> indicates a database degrade. You will get one of these if you or your software try to to access a global that has become broken.

System Hangs During Startup

This can also be due to hardware. It can sometimes indicate a corruption to the system definition global ^SYS.

Integrity Checker Messages

If you get any of the above errors, you should run the ^IC or integrity checker program. It will try to follow all of the pointers in a global and check the validity of the data and print error messages indicating any errors if they exist. This program should be run once a week or so on your entire database so you can become aware of minor corruptions before they get worse.

What do you do?

Once you know a global is broken, there are a few things you should and shouldn't do:

Panic - This is not a good idea. I would avoid panic at all costs.

Cry - Although sometimes helpful, don't do this right away. Wait until you know the extent of the damage.

Breathe - Take a few deep breaths and look at the situation. Things are probably not as bad as they look. This is highly recommended.

ISOLATING THE PROBLEM

There are several utilities that will allow you to get an idea of what your global looks like. The most important part of fixing broken globals is the ability to draw pictures. These utilities will help you do that. I will use small globals throughout this paper for ease of handling. The method of handling larger globals is the same but you may have to draw portions of your global.

%GE

The Global Efficiency Routine will give you a "Bird's Eye View" of what a global looks like. Figure 4 shows a sample run:

>D ^%GE

Global Efficiency

Global(s) ? > X  
Global(s) ? > <CR>

TST,DPS Global Directory Block: 4811

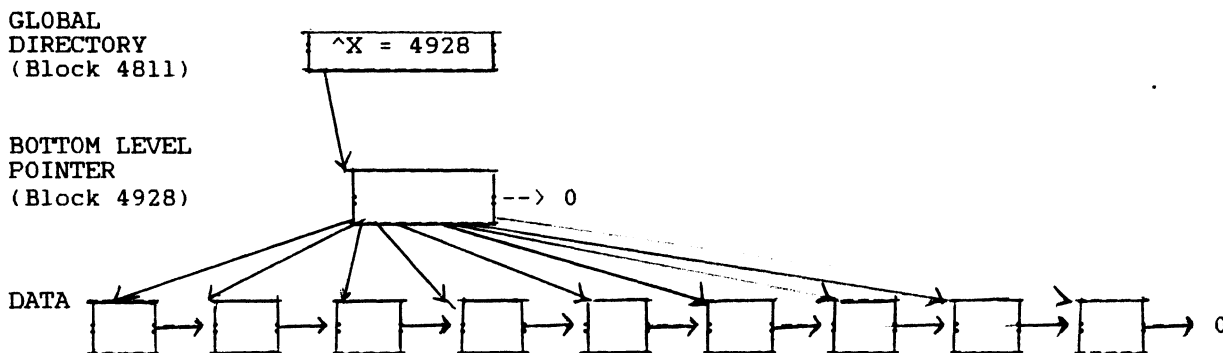
Global ^X First block: 4928

Bottom pointer level 1 - 1 8%

Data level - 9 88%

(figure 4)

You are told what block is the global directory block, which is the first block for the global and some information about what the global looks like. The 1 to the right of the words "Bottom pointer level 1" and the 9 to the right of "Data level" indicate the number of blocks on each row. The percentage numbers tell you how full each block is indicating how efficiently the data is stored. The 8% is somewhat misleading in that the data is efficiently stored but there aren't many pointers so the block usage is low. From the information in figure 4, you can actually draw a picture of the global (figure 5):



(figure 5)

At this point, you don't know the block numbers, but you have a good idea of what the global looks like. Occasionally, a global is corrupted in a way that %GE will fail. If this happens, don't despair, there are other ways to picture what the global looks like. This must be run from the UCI in which the global resides.

%G or %GL

-----

The global list utilities will not really help you draw a picture. In some cases, for example a right pointer being corrupted, you may get a partial listing and this might help you in figuring out what is wrong. This must be run from the UCI in which the global resides.

^IC

---

The integrity checker (^IC) reads through the global and tries to give you a message indicating where an error exists. The ^IC program must be run from the system manager's UCI. Figure 6 is a sample ^IC report:

>D ^IC

1. Print existing report  
2. Compile new report  
Enter one of the above options > 2  
Do you want the report to automatically print when it's done ? <Y> N

Check data base integrity for which volume set ? > S0

Structure S0: UCI ? > TST  
Global ? > ^X  
Searching directory...  
Global ? > ^ <<CR>  
Check Routine Directory ? <Y> N

Structure S0: UCI ? > <<CR>

Selected UCIs for Structure S0

UCI #6 (TST)  
^X

Integrity checker running in background  
A report will be compiled into ^IC

>D ^IC

1. Print existing report  
2. Compile new report  
Enter one of the above options > 1 Output Device ? > 0

--- INTEGRITY CHECKER REPORT ---

6-Apr-85 13:26 CHECKING UCI #6 (UCI NAME: TST, STRUCTURE: S0)  
6-Apr-85 13:26 CHECKING ^X  
ERROR in block 4931 (pointed to by 4928): Right pointer: expected 4932 got  
4933  
6-Apr-85 13:26 FINISHED

1. Print existing report  
2. Compile new report  
Enter one of the above options >  
> <<CR>

(Figure 6)

You can use the integrity checker report to help identify the problem as you draw your picture. If you run the integrity checker on a global that is in use, you will occasionally see an error message when an error does not exist (for example, a block splits while you are checking). If you get an error where none existed before on a routine ^IC, re-run the integrity checker on that global only to double check that an error exists.

The block dump utility can be used to dump the contents of a given block to a terminal. It can be used in a major corruption to determine if any data in a corrupted block is salvageable. To use it, you must have a knowledge of the layout of DSM-11 block structures (discussed in 13 of the DSM-11 User's Guide).

I am only talking about minor corruptions so I will mention no more on the block dump utility. It must be run from the MGR UCI.

^FIX

This is the major utility used for both analyzing the DSM-11 global structure and for repairing broken trees. There are many different things you can do with the FIX utility. I will mention these here and then show some of them to you as we actually fix a broken global. The ^FIX utility must be run from the MGR UCI. The functionality of ^FIX changes with the type of block you are accessing:

Global Directory Block - If you are looking at a Global Directory Block you can do the following:

1. Punt leaving the block unchanged. (This option was no doubt invented by a frustrated armchair quarterback).
2. File the block saving all changes you have made.
3. List the block printing the block to a terminal.
4. Clear the block eliminating all data
5. Insert a Global by adding a pointer to that global.
6. Erase a global by removing it's pointer.
7. Change the offset. The offset is the number of the next available \ byte in the block. If you lower the offset, you effectively eliminate data following the new offset.
8. Change a right link pointer.
9. Change a down pointer.

Of the above, you can expect not to use the clear or offset options and I will not discuss these further.

Pointer Block - If you are accessing either a pointer or bottom level pointer block, you can do the same things as above with the exception that options 5 and 6 (insert and erase globals) become insert and erase pointers.

Data Block - If you are accessing a data block, your options are again the same but options 5 and 6 allow you to insert and erase global nodes (data) instead of pointers.

Map Block - If you are accessing a map block, you are very limited in what you can do. You can Punt, File and List the block as above. Additionally, you can Reset the Free count. This option causes ^FIX to read the entire MAP block and record the number of allocation words indicating free blocks in a special byte in the map block. You can also change the allocation word. This is the option you will use to free a block you have deleted from your global.

COMMON CORRUPTIONS

We will look in detail at a few of these corruptions, but for now, let me list the more common corruptions:

Pointer Corruption - Either a down pointer a right link pointer has become corrupted. This is the easiest and, by far, most common.

Data Block Corruption - Somehow, something gets written over the data in a data block. When this happens, there is usually some loss of data. This is sometimes the whole block and sometimes part of a block.

Pointer Block Corruption - Something gets written over a pointer block. This may or may not cause data to be lost. Often you can reconstruct the information lost at this level.

Garbage in Subscript - Occasionally, we see something like control characters in a subscript rather than what we expect to see. If this is at a data level, we sometimes lose data. At a pointer level, this can usually be fixed without a loss of data.

I'm sure I have left other corruptions out, but these cover the more common problems you may encounter.

## HOW DO YOU REPAIR THE DAMAGE?

OPTION: PUNT (LEAVING BLOCK UNCHANGED)  
OK ? <Y> <CR>

There are two methods of repairing damage to a global. These are:

(Figure 7)

**^FIX** - This is used to reconstruct ---- damaged globals. In most cases, this is what you will use.

**^%GTO** and **^%GTI** - These can sometimes be ----- used when you have lost many pointers but can verify that your data level is in tact. (a level being a row on the tree).

These will both be used in fixing some broken globals.

Once you enter the block number, the first thing you see is the number of the UCI that owns the block and the number of the UCI that set it. **^FIX** gets this information from the map. You are then prompted for Block type, Global name and collating type. The defaults are the current values and you should type carriage returns through them until you get to the **OPTION** prompt. If any of these appear blank, you probably have a completely corrupted block. If you know what the global is and what the block type should be, you can fill these in and try to list the block, but chances are the information it contained is lost. In that case, you want to either restore from a backup or remove the block and re-enter the information. In an upcoming example I will show you how to do this.

### SOME SAMPLES

#### Pointer Corruption

Take a glance back at figures 5 and 6. Figure 5 shows us what global **^X** looks like and figure 6 shows us the **^IC** results. It claims that we have an error in block 4931 which is pointed to by block 4928. We are further informed that the error is a right pointer error and that the integrity checker expected to find block 4932 and found block 4933. What we have to do now is play integrity checker. Let's draw a picture and see if we can find the problem. To do this we will use the **^FIX** utility. We don't have to worry about drawing the global directory block. Figure 7 shows a listing of the first block of **^X**. The number for that block was found in the **%GE** results (figure 4). You could also find the first block by running **^UCIADD** to get the number of the directory block and then using **^FIX** to list that block in the same manner as we list the first block below.

>D **^FIX**

```
Enter block # > 4928 4928:S0
MAP block entry indicates block is in UCI 6 set by UCI 6
Block type: <BOTTOM LEVEL POINTER> <CR>
Pointer block for: <^X> <CR>
Numeric collating? <Y> <CR>
```

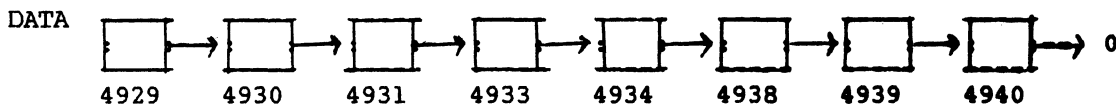
OPTION: LIST BLOCK

Output Device ? < 0 > <CR>

BLOCK 4928:S0

```
OFFSET: 70 RIGHT LINK POINTER: 0:S0
 0: X = 4929
 6: X("137") = 4930
 16: X("260") = 4931
 25: X("506") = 4933
 34: X("629") = 4934
 43: X("752") = 4938
 52: X("875") = 4939
 61: X("998") = 4940
 70:
```

At the option prompt, I chose to list the block. The first block is a bottom level pointer block. This means it points to data. The listing gives us the block number and right link pointer. It also gives you the offset. The offset shows the next free byte available in the pointer block. Each pointer line shows the offset in the block where the pointer is located. It then shows the full global reference of the first global in the block it points to and then you are given the number of the block that global or the next pointer to that global resides in. Furthermore, the listing tells us there are eight blocks on the data level. Each pointer points to one block. Immediately, we become suspicious. A glance back at figure 4 tells us that there should be 9 blocks. For each down pointer, there should be a corresponding right link pointer in the next row. Therefore, the pointer block listed above tells us that the next level down (in this case, the data level) should look something like this (figure 8):



(figure 8)

The PUNT option allowed me to exit the ^FIX program. Now, let's go back into ^FIX and look at the DATA level to check the right link pointers (figure 9):

>D ^FIX

```

Enter block # > 4930 4930:S0
MAP block entry indicates block is in UCI 6 set by UCI 6
Block type: <DATA> <CR>
Data block for <^X> ^<CR>
Numeric collating? <Y><CR>

```

```

OPTION: RIGHT LINK POINTER
BLOCK # 4931:S0> No action taken.

```

```

OPTION: PUNT (LEAVING BLOCK UNCHANGED)
OK ? <Y> <CR>

```

Enter block # > 4931 4931:S0  
MAP block entry indicates block is in UCI 6 set by UCI 6  
Block type: <DATA> <CR>  
Data block for <^X> ^ <CR>  
Numeric collating? <Y> <CR>

OPTION: RIGHT LINK POINTER  
BLOCK # <4932:S0> No action taken.

OPTION: PUNT (LEAVING BLOCK UNCHANGED)  
OK ? <Y> <CR>

Enter block # > 4932 4932:S0  
MAP block entry indicates block is in UCI 6 set by UCI 6  
Block type: <DATA> <CR>  
Data block for <^X> ^ <CR>  
Numeric collating? <Y> <CR>

OPTION: RIGHT LINK POINTER  
BLOCK # <4933:S0> No action taken.

OPTION: PUNT (LEAVING BLOCK UNCHANGED)  
OK ? <Y> <CR>

Enter block # > 4933 4933:S0  
MAP block entry indicates block is in UCI 6 set by UCI 6  
Block type: <DATA> <CR>  
Data block for <^X> ^ <CR>  
Numeric collating? <Y> <CR>

OPTION: RIGHT LINK POINTER  
BLOCK # <4934:S0> No action taken.

OPTION: PUNT (LEAVING BLOCK UNCHANGED)  
OK ? <Y> <CR>

Enter block # > 4934 4934:S0  
MAP block entry indicates block is in UCI 6 set by UCI 6  
Block type: <DATA> <CR>  
Data block for <^X> ^ <CR>  
Numeric collating? <Y> <CR>

OPTION: RIGHT LINK POINTER  
BLOCK # <4938:S0> No action taken.

OPTION: PUNT (LEAVING BLOCK UNCHANGED)  
OK ? <Y> <CR>

Enter block # > 4938 4938:S0  
MAP block entry indicates block is in UCI 6 set by UCI 6  
Block type: <DATA> <CR>  
Data block for <^X> ^ <CR>  
Numeric collating? <Y> <CR>

OPTION: RIGHT LINK POINTER  
BLOCK # <4939:S0> No action taken.

OPTION: PUNT (LEAVING BLOCK UNCHANGED)  
OK ? <Y> <CR>

Enter block # > 4939 4939:S0  
MAP block entry indicates block is in UCI 6 set by UCI 6  
Block type: <DATA> <CR>  
Data block for <^X> ^ <CR>  
Numeric collating? <Y> <CR>

OPTION: RIGHT LINK POINTER  
BLOCK # <4940:S0> No action taken.

```

OPTION: PUNT (LEAVING BLOCK UNCHANGED)
OK ? <Y> <CR>
Enter block # > 4940 4940:S0
MAP block entry indicates block is in UCI 6 set by UCI 6
Block type: <DATA> <CR>
Data block for <^X> ^ <CR>
Numeric collating? <Y> <CR>

```

```

OPTION: RIGHT LINK POINTER
BLOCK # <0:S0> No action taken.

```

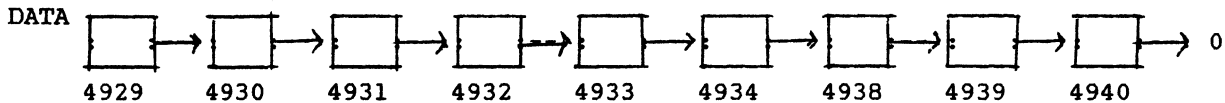
```

OPTION: PUNT (LEAVING BLOCK UNCHANGED)
OK ? <Y> <CR>
Enter block # > <CR>

```

(figure 9)

What I did above was look at each right link pointer. When the right link pointer option is chosen, the current value is given as the default. A carriage return leaves it unchanged, so the above left everything unchanged but told me the data level looked like this:



(figure 10)

Well, this is different from what we expected. We seem to have picked up an extra block in here. That is block 4932. This means that one of two situations exist. Either we have an extra block or we have lost a pointer. Let's list block 4932 and take a look at it's contents (figure 11):

D ^FIX

```

Enter block # > 4932 4932:S0
MAP block entry indicates block is in UCI 6 set by UCI 6
Block type: <DATA> <CR>
Data block for <^X> ^ <CR>
Numeric collating? <Y> <CR>

```

```

OPTION: LIST BLOCK
Output Device ? < 0 > <CR>

```

BLOCK 4932:S0

OFFSET: 1002 RIGHT LINK POINTER: 4933:S0

```

0: X("383") = "383"
12: X("384") = "384"
20: X("385") = "385"

```

. (data omitted)

```

994: X("505") = "505"
1002:

```

```

OPTION: PUNT (LEAVING BLOCK UNCHANGED)
OK ? <Y> <CR>

```

(figure 11)



A look back at figure 7 shows that this information would fit nicely between blocks 4931 and 4933. We must insert a pointer into block 4928 (figure 12):

>D ^FIX

```
Enter block # > 4928 4928:S0
MAP block entry indicates block is in UCI 6 set by UCI 6
Block type: <BOTTOM LEVEL POINTER> <CR>
Pointer block for: <^X> <CR>
Numeric collating? <Y> <CR>
```

```
OPTION: INSERT POINTER
Insert Global ^X(383)
Pointer block # 4932:S0 Inserted
Insert Global ^ <CR>
```

```
OPTION: LIST BLOCK
Output Device ? < 0 > <CR>
```

BLOCK 4928:S0

```
OFFSET: 79 RIGHT LINK POINTER: 0:S0
 0: X = 4929
 6: X("137") = 4930
 16: X("260") = 4931
 25: X("383") = 4932
 34: X("506") = 4933
 43: X("629") = 4934
 52: X("752") = 4938
 61: X("875") = 4939
 70: X("998") = 4940
 79:
```

```
OPTION: FILE BLOCK
OK ? <Y><CR>FILED
Enter block # > <CR>
>
```

(figure 12)

To insert a pointer, chose the INSERT option. I was then prompted to enter the full name of the first global in the block and the block number. Note the "S0" after the block number. This indicates volume set 0. It is required and you can specify either a specific disk (eg. "DL0") or a volume set number. I listed the block to make sure it looked okay and then I FILED the block. This is very important. All changes are temporary until you FILE them. If I had exited without filing the block, nothing would have been changed.

The global is fixed. There is only one thing to do now; That is to run the integrity checker to assure the global is fixed (figure 13):

--- INTEGRITY CHECKER REPORT ---

```
7-Apr-85 9:16 CHECKING UCI #6 (UCI NAME: TST, STRUCTURE: S0)
7-Apr-85 9:16 CHECKING ^X
7-Apr-85 9:16 FINISHED
```

(figure 13)

The printed report shows no errors in the global. We have fixed it.

A right link pointer corruption is dealt with in the same manner. When you draw a picture, you will see a discrepancy between the down pointers and the right link pointers. Once you determine what is wrong, you take appropriate action to repair the damaged pointer.

If you get garbage in a pointer, you might see a "bad first node" error. A look at the pointer and a look at the look at the information in the block will tell you what the pointer should point to. You can then erase the bad pointer and insert the proper one. Most corruptions can be handled as pointer corruptions once the problems are isolated.

Of the common corruptions, I would say that corrupted single pointers and missing or extra pointers can be handles as above.

## Data Block Corruption

-----  
In the case of a data block corruption, you must once again draw a picture of your global. A "bad first node" error is a sign you might have bad data. If you do, your picture will look okay, so you have to start looking at the data. Here is an ^IC report from a global with a bad data block (figure 14):

```
--- INTEGRITY CHECKER REPORT ---
7-Apr-85 9:35 CHECKING UCI #6 (UCI NAME: TST, STRUCTURE: S0)
7-Apr-85 9:35 CHECKING ROUTINES
7-Apr-85 9:35 CHECKING ^Y
ERROR in block 4931 (pointed to by 4928): Bad first node
ERROR in block 4931 (pointed to by 4928): Bad first node
7-Apr-85 9:35 FINISHED
```

(figure 14)

Note that there are two errors listed. Sometimes you will get multiple messages for one error. Address one error at a time and then run the integrity checker and this presents no problem.

Assume you have drawn a picture of your global and it looks in order. Now we must look at the block we think the error is in (figure 15):

>D ^FIX

```
Enter block # > 4931 4931:S0
MAP block entry indicates block is in UCI 6 set by UCI 6
Block type: <DATA> <CR>
Data block for ^ (block not filed) <CR>
Enter block # > <CR>
>
```

(figure 15)

The first indication we have a bad corruption is that there is no global name listed. We can go into ^FIX and try to reconstruct (figure 16):

>D ^FIX

```
Enter block # > 4931 4931:S0
MAP block entry indicates block is in UCI 6 set by UCI 6
Block type: <DATA> <CR>
Data block for ^Y
Numeric collating? <Y> <CR>

OPTION: LIST BLOCK
Output Device ? < 0 > <CR>

BLOCK 4931:S0

OFFSET: 1001 RIGHT LINK POINTER: 4932:S0
0:
<UNDEF>WRT^FIXDATA1:2
```

(figure 16)

After supplying the name of the global and trying to list the block without success (the <UNDEF> is not due to a bug in ^FIX but rather the result of a nasty corruption). my only choice is to remove the bad data block.

The first thing we want to do is remove the pointer to the bad block, list it to see if it is okay and then FILE it (figure 17):

>D ^FIX

Enter block # > 4928 4928:S0  
MAP block entry indicates block is in UCI 6 set by UCI 6  
Block type: <BOTTOM LEVEL POINTER> <CR>  
Pointer block for: <^Y> <CR>  
Numeric collating? <Y> <CR>

OPTION: LIST BLOCK  
Output Device ? < 0 > <CR>

BLOCK 4928:S0

OFFSET: 34 RIGHT LINK POINTER: 0:S0  
0: Y = 4929  
6: Y("137") = 4930  
16: Y("260") = 4931  
25: Y("383") = 4932  
34:

OPTION: ERASE POINTER  
ERASE Global ^Y(260) Erased  
ERASE Global ^ <CR>

OPTION: LIST BLOCK  
Output Device ? < 0 > <CR>

BLOCK 4928:S0

OFFSET: 25 RIGHT LINK POINTER: 0:S0  
0: Y = 4929  
6: Y("137") = 4930  
16: Y("383") = 4932  
25:

OPTION: FILE BLOCK  
OK ? <Y><CR>FILED  
Enter block # > <CR>  
>

(figure 17)

The next thing we want to do is adjust the right link pointers on the data level so they match the pointer block we just changed (figure 18):

>D ^FIX

Enter block # > 4930 4930:S0  
MAP block entry indicates block is in UCI 6 set by UCI 6  
Block type: <DATA> <CR>  
Data block for <^Y> ^<CR>  
Numeric collating? <Y> <CR>  
OPTION: RIGHT LINK POINTER  
BLOCK # <4931:S0> 4932:S0 Done.

OPTION: FILE BLOCK  
OK ? <Y> <CR> FILED  
Enter block # > <CR>

The last thing we want to do is go into the map block and make the removed block a free block. The following formula will give you the block number for the map that keeps track of a given block figure 19:

Map block # = Block number \ 400 \* 400 + 399  
(Figure 19)

Applying the above formula to block 4931, we find it is mapped by block 5199. Now we can adjust the allocation word to indicate the block is free (figure 20):

>D ^FIX

Enter block # > 5199 5199:S0  
Map Block  
Map Type: <DATABASE> <CR>  
OPTION: ALLOCATION WORD  
For Block# 4931  
Allocation word: <USED BLOCK> FREE BLOCK Done.  
For Block# <CR>

OPTION: FILE BLOCK  
OK ? <N> Y FILED  
Enter block # > <CR>

(figure 20)

We are now done, but we have one problem. We have lost data and we need to know what is lost. To do this, you can use the ^FIX program and list the blocks that were on either side of the removed block. In this case that is blocks 4930 and 4932. Note the last node in block 4930 and the first in 4932. Any data that would fall (in sort order) between these two nodes is what was lost. You must re-enter the information or copy it from a backup. You cannot do a partial restore, so this would involve restoring to another disk and then use the global save (%GTO) and global restore (%GTI) utilities to recover the data.

The above two samples are very simple ones, but used as guidelines, they should help you get started on repairing a simple degradation.

#### A FEW RULES

1. Analyze the situation. If you find many errors in one global, it may be faster to restore from a backup and dejournal than to try and repair the situation.
2. DON'T delete corrupted globals before you fix them. If you have a corruption where one global points into another, you may delete more than you wanted. The garbage collector uses the right link pointers and these should be repaired prior to deletion.
3. If you don't have a backup, backup before you repair. The ^FIX utility can also be a ^BREAK utility. You may make a situation worse, so be sure you have a backup before you attack the corruption.
4. Take notes. Everything you do with ^FIX can be undone if you keep track of your changes.
5. Run the integrity checker on fixed globals before you give them a clean bill of health.

## SUMMARY

### WHAT NEXT?

Now that you have fixed the global what should you do? If you have a good idea of what caused the corruption (eg... Corruption occurred after a power failure), I would recommend you do what you normally should be doing. That is, regular backups, and periodic integrity checks. I'd also recommend you periodically use ^FIX to obtain listings of the global directories for each UCI (so you have a record of global name and first block) and that you run a weekly slow disk block tally (^DBT). This may be more than necessary, but I am cautious and would recommend you be cautious as well.

If you don't know what caused the corruption, I would recommend you assume the worst. Assume the corruption will occur again and use the %GTO program to save a copy of the repaired global. You might also increase the frequency of backups you perform until you have isolated a problem or are relatively confident the corruption was a one-time event. This can not always be definitely determined, but if the database stays in tact for many days, I would assume that the problem has gone away.

If the problem recurs, try to identify any software or hardware changes that might be responsible. If you have changed your software or applied patches, go back and make sure you didn't make any mistakes.

In the limited amount of space I have here, I have just touched the surface of database repair. I hope you have gotten some insight into how to repair a broken database. In my samples, I used very small globals. With larger globals, you might have had multiple levels of pointers, but the methodology would have been the same. The biggest piece of advice I can give you is to tell you to be patient and draw pictures. If you look closely at broken globals, you will get the answers. Database reparation, in most cases, is a very straight-forward procedure.

I recommend you create a test UCI on your system (if you don't already have one). Create some globals and use the previously mentioned utilities to look at the globals and draw pictures of them. Once you are comfortable with that, use ^FIX to remove a pointer or change something then run the integrity checker to see what it reports. Fix the globals. If you do this a few times, it may help you when you have a real corruption. Good Luck!

**MUMPS Programming Standards**  
or  
**How I Stopped Worrying and Learned to Love MUMPS**

by

Robert C. (Chris) Richardson  
Computer Scientist  
Computer Sciences Corporation  
P. O. Box 2217  
Ridgecrest, California 93555

ABSTRACT

The MUMPS programming language has been called cryptic by its detractors and concise by its advocates.

Applications are all too often developed as network models rather than a heirarchical models, and this can make the program flow mathamatically unprovable. The result may well be difficult to decipher let alone support.

At the Naval Weapons Center in China Lake, we have developed some programming standards to aid our programmers in developing good coding techniques. This paper provides a list of these guide-lines. These standards should be a guide-lines and not a stick. We are providing a compliance checker for our programmers, which allows our programmers to test themselves against the standard. They then develop a consistent programming style which any of the other programmers may follow.

BACKGROUND

The major MUMPS application at the Naval Weapons Center (Code 084) is the Integrated Disbursing and Accounting (IDA) system. We are currently running DSM-11 on a PDP-11/34 (the production system) and using DSM running on a VAX-11/780 (our development system). We will soon be upgrading the 11/34 to an 11/44.

We have also been using MUMPS for some scientific applications. This includes a parser for 30,000 lines of assembler for the processor on a missile. The parsed tables were then used to generate pseudo-Pascal code.

Additionally, we have Dr. Richard Walter's Micro-MUMPS running on several DEC Rainbows.

"I never met a piece of MUMPS code that I didn't want to rewrite."

- Anonymous.

"MUMPS is an Anarchist's dream."

- Anonymous.

The Standards

As stated in the abstract, a standard should not be a stick used to make a programmer conform. It should be a series of guide lines on how to shop or a group of programmers should develop code. This allows any of the members of the group to support the code of any of the others.

RATIONALE:

1. Routine Naming Conventions

A. The first label of a routine should be the same as the name of the routine itself.

This allows us to make a backup or second version of a routine without impacting the production version and not losing the identity of the original source routine.

B. Routines in the same package should share the same first three letters.

Quite often a MUMPS application package will contain more than a single routine. We recommend that the routines directly

associated with a package share the same three character prefix. This allows the routines that are closely related to be listed together. There are usually utilities that allow the selection of groups of routine names. This allows the grouping of related routines.

### C. Utility routines

Utilities are a big feature of MUMPS. MUMPS allows the creation of special routines that may be invoked by, but not changed by the user. These routines all start with a "%" character.

- \* Utility routine names should begin with a '%' (percent) character.

Manager utility routines are an important feature. They allow code to be accessible to users elsewhere on the system. They may use this code, but may not modify it.

- \* Utility variables should also begin with a '%' character.

Percent variables are reserved for utility routines. This keeps the utility routines from impacting the applications routines which call them. There can't be any confusion over utility variables that start with '%' and application routine variables where the first character may not be '%'

## 2. First Line Conventions

The %FL (First Line) utility supplied with most MUMPS implementations creates a list of the first lines out of each of a series of routines. As mentioned earlier the first three characters of each routine name in a package must be the same. This allows %FL to sort routines of a single functional area (package) together. This in turn allows us to keep track of routines and packages.

### A. A single "DO" command

This constraint causes every application to have a consistent execution structure, and gives the support programmer an initial understanding of how the routine works.

- \* In an effort to simplify and unify the execution of application segments, we have divided the execution of any program module into at most, three parts. The arguments following the

"DO" command in the first line should be one or more of the following:

#### a. INIT - Environmental setup.

This establishes the require files and variables that must exist before the production part of the application routine.

#### b. START - The production portion of the application.

This is where the real work of the application is performed. It is in this area that we will establish the restart capability of the application. Should the application halt, the analyst may look to this area to restart the application run.

#### c. EXIT - The cleanup from the application.

This section allows us to clean up the scratch variables and globals that are not needed after the production. This is extremely important if the routine is called from a control routine.

- \* The first operational block below the "DO" command should be one of the labels above, (usually INIT or START).

### B. A brief statement of the routine's purpose.

Since the %FL routine lists the first line of each of a series of selected routines, it allows the tracking of related routines and utilities. Routines which share similar or related function can then easily be grouped.

### C. The last date the routine was updated.

The last date a routine was updated must be included on the first line. This chronicles back-up copies of a routine, and allows the most recent change to be identified. %FL makes the scanning of these routines easy.

### D. The name of the author of the routine.

The author's name is important, but less so once these standards have been adopted. "Programming standards should be so thorough that to an outside observer, all programs appear written by the same person." ("Productivity now! (Why Wait?)", by George Proudfoot, Computerworld).

A sample first line is illustrated in EXAMPLE 1:

#### EXAMPLE 1:

```
COMPLY DO INIT,START,EXIT; Compliance Checker Driver ; 2-APR-85 ; RCR
```

### 3. Block Structure Conventions

To be consistent with block structuring, the labels mentioned in the title line should be defined immediately after the control block and should be placed in the order called. The block structuring used in this process is not the classic stepped block structure. In this technique, all blocks are defined at the same level. Only three levels are required at a time:

- 1) The block in question.
- 2) The block level that calls this block.
- 3) The blocks that are called from this block.

MUMPS allows the start of execution at any block level. Each block below the current level should be able to be treated as a discrete piece of code. The symbol table may be initialized to simulate the state of execution prior to entry into the block being tested.

A. Each block should have a single label for entry.

Consistent with structured design, this restraint allows for the easy definition of the hierarchical model of the application.

In MUMPS, this is doubly important since the MUMPS language allows the start of execution at any label or an offset from a label.

B. A Label should not be numeric.

MUMPS allows the use of numerics as labels. This practice conveys little practical information for support and there may be difficulty with this technique in certain implementations. Some implementations may try to evaluate a numeric label as a value rather than a character string.

C. No label reference should be a displacement off of a label or an alternate entry point into another routine, i.e. Avoid "DO ABLE+5" and "DO BAKER^SORT".

GOTO and DO commands use label references. The use of displacement and alternate entry point references are usually used to circumvent structured techniques. These should be carefully watched.

D. No block should call itself. Avoid recursion.

Recursion is a useful tool when used well. Unfortunately, few programmers use this technique properly. It is also a disguised loop. These guidelines are supposed to make loops and levels of control easy to follow.

Potential recursion, or the possibility that a routine may unintentionally call itself, must also be avoided. For example, routine A calls routine B. Then routine B calls routine C. Routine C invokes routine A. Who is controlling whom?

E. No block should call a block defined above it.

This simple rule assures that the hierarchical model does not degrade to a network model. This should extend to the application level. At the routine level, there should be no chance of inadvertent routine looping because of accidental recursion.

F. No program line should be over 80 characters in total length.

The importance of this recommendation is not immediately clear. MUMPS has a simple block structuring built into it. The control structure types, FOR, IF, ELSE, and GOTO, provide a line execution capability. This ability to control commands to the right of the control command implies a type of block structure. The maximum line length on a line in a routine in MUMPS is 255 characters. By limiting the length of a line to 80 characters, we force these implied blocks to be brief and concise. Another advantage is that the routines may be edited on a terminal with full screen capability. This also insures that %INDEX listings of a set of routines will not need to have rap-around lines on a 132 column listing.

G. The last command in every block should be a single unconditional "QUIT" or "HALT" command.

The ending of a block is critical to real block structuring. This insures that control does not inadvertently fall into the next block. The flow and visibility of control is essential in a support environment.

The QUIT is usually used to end a block. It provides a means of returning from a block. The use of inline QUITs is normal. It provides the means of conditionally terminating FOR commands and block return. The HALT is an emergency exit from an application.

H. No block should be longer than 20 lines of code.

MUMPS is a very concise means of coding. Most functions that should be contained in a block, can almost always be completed in less than twenty lines. This limitation also keeps the programmer from getting concepts jumbled together in the same block.

### 4. Command Conventions

The commands in MUMPS are on the surface reminiscent of similar commands in FORTRAN or Pascal. There are some other commands that don't have analogs in any other language.



A. Avoid the "BREAK" command in production routines.

The BREAK command is advantageous in the evaluation and testing of code. It has no place in production code, however, since it could allow the user into Programmer Mode and thus destroy any security in the system. Nothing could be more devastating than an uninitiated end user being suddenly confronted with an operating system prompt. BREAK commands must NOT have potential for execution during a live application.

C. Avoid the "ELSE" command.

The MUMPS ELSE command is unlike the 'ELSE' command in any other language, and it is potentially ambiguous. In MUMPS it tests a system environmental status flag, \$TEST. It is true that \$TEST does reflect the condition of the last IF command, i.e., 1 or 0 (true or false). It also provides for the testing of the successful completion of timed I/O and data base lock features.

The line of code containing the ELSE command may be preceded by variable logic that will affect the outcome of \$TEST. The \$TEST variable may be changed by a variety of sources. By the time \$TEST is evaluated, the command that last changed \$TEST may be unpredictable. The only way the ELSE should be used is directly after a device control command. See EXAMPLE 2:

It was a cute idea, but it still needs work. A waiver is still required.

D. Avoid the "GOTO" command.

The GOTO is the most dangerous command in a hierarchical model. The GOTO relinquishes control of the current process to whatever label or line of code is mentioned. All of a sudden, a hierarchical model becomes a network model.

There is no CASE command in MUMPS. The GOTO is the closest thing MUMPS has to this useful command. It must be implemented very carefully as in EXAMPLE 3:

In the preceding example, the GOTO only references blocks defined immediately after itself. Any of the quit commands actually terminate block XSPLIT and not the block it belongs to. The evaluations are attached to the GOTO command after each label and colon (: ) attached to the GOTO command. These are known as postconditionals. The last evaluation, 1, is always true. If all of the other evaluations are false, this last label will be jumped to. As soon as one is found to be true, that label is selected to control the continuing execution. The QUIT at the end of the XSPLIT block is never executed.

E. Avoid the "XECUTE" command.

The XECUTE command is a really neat idea. It allows the MUMPS programmer to store MUMPS code in a string (local variable or stored as a data base). This may be used to hide non-standard code. Usually this can be useful for performing difficult activities utilizing the routine buffer.

F. Avoid the "VIEW" command.

Much of the VIEW command structure and facility is left to the implementor. Ostensibly, it is a means of viewing (and possibly changing) locations in memory. The locations in memory that are usually used are implementation dependent. This means that a waiver for this command is critical.

G. Avoid the "Z" - class commands and functions.

The "Z" - class commands and functions are extensions to the standard. This is a

#### EXAMPLE 2:

```
USE 0 READ !,"enter name>".NAME:10 ELSE WRITE "time out"
```

#### EXAMPLE 3:

```
DO XSPLIT
QUIT
XSPLIT USE 0 READ !,"enter value>".X
 GOTO ABLE:X<1,BAKER:X<1.25,CHARLIE:X=2,DELTA:X>2,EPSILON:1
 QUIT
ABLE SET X1
 QUIT
BAKER SET X1X=1+X
 QUIT
CHARLIE SET X1X=1-X
 QUIT
DELTA SET X2
 QUIT
EPSILON WRITES !,"X is a value between 1.25 and 2.0",?32,X
 QUIT
```

means for a software implementor to provide features to the language for inclusion in the standard. It is a laudable idea, but it inhibits the transportability of the code. This is a warning level of error. A waiver is still required to justify these features.

## 5. Control Structure Conventions

FOR, GOTO, IF and ELSE are the control commands. They are used to control the execution flow of commands near them. The FOR, IF, and ELSE control the execution of commands that lie to its right. The GOTO may control the re-execution of code to its left or the redirection of control away from code to its right. This is illustrated in EXAMPLE 4:

The FOR loop will continue to ask for the name until an entry is made that contains no control characters.

A. A control command should be the first command on the line it controls.

Control commands are important. It is important to make them prominent. We do this by putting the first control command as the first command on that line. The 80 character line restriction helps to keep the line blocking from being too complex. One of the reasons for these constraints is to teach the programmer to program in "baby talk".

B. No more than two control level commands should be on the same line.

There is a strong tendency in most MUMPS programmers to try to stick as many commands as possible on a single line. It is possible to have many nested FOR loops on a single line. This can make the code difficult to follow.

The control commands are important. The complexity of commands around these control commands can obscure the real control of a piece of code. If more complexity is needed, a block may then elaborate the complexity as a series of discrete lines.

C. Variables in the "FOR" command should be LPnn if they are used for driving a loop.

Loops are fun. Loops are easy. Loops are trouble, especially in MUMPS when the same variable may be used and changed within a FOR loop construct.

### EXAMPLE 4:

```
FOR LP011:1 R !,"entername>".NM QUIT"NM'?.ELC.E W "no cntrl chars"
```

## 6. Variable Conventions

Variables are always a big problem. Many languages try to write a book in order to follow the variables. MUMPS maintains the symbol table intact during and after the run. There are very few assumptions involved in the use of MUMPS. The length of the variable name is not quite as important in MUMPS as in other languages. However, the use of shorter variable names in MUMPS increases the speed of execution. So there are some conflicting concerns in the MUMPS language.

The restrictions on variables are similar to those specified in other language standards. A variable should be descriptive. MUMPS has a dynamic symbol table and symbols should not survive longer than they are needed. MUMPS does allow a percent character to be used as the first character of a symbol. These are reserved as state variables for utilities (such as %AB for abort and %ERR for errors encountered within a utility process). By restricting scratch variables to one or two characters, we can quickly segregate important variables from scratch variables. Short variable names are not a problem. MUMPS maintains the symbol table even if the process errors off. It is usually easy to tell the purpose of a symbol from its contents.

Variables in MUMPS have self attributes which are available to the programmer. These are length, descendency, numeric value, and dynamic existence. Other languages require specific typing to make such attributes available.

A. Variables which are used in generalized utilities should start with a percent character.

The percent sign is valid if used as the first character of a local variable name. These may be created and destroyed by the programmer at will. As a matter of convention, we reserve these variables to be set by generalized utilities. Other routines may interrogate these utility interfaces upon return.

The percent sign may also be used in global names. Only routines running in the manager's partition or account may kill or create percent global nodes.

B. A variable should be created only when it is needed. It should be killed as soon as it is no longer needed.

The dynamics of the MUMPS language allow for the run-time creation and destruction of local variables. This allows the application to clean up as it goes. Should a problem occur, only those variables which are still in existence will remain available in the symbol table.

C. A scratch variable (only used within a single block) may be a single character. All others should be three characters or more.

There is a strong chance that the reuse of variables outside of a particular block will cause a problem, since all local variables are common, it is easy to unintentionally step on the wrong variable. Care must be taken to insure that variables that are assumed to be only local are not inadvertently changed. By the use of the %INDEX routine, it is easy to identify where a variable is just referenced, created, or changed.

D. Avoid Indirection.

Indirection allows the programmer to use a variable to hold a string containing another variable name or MUMPS code. It sounds like using mirrors, but it is a valuable technique.

Indirection is a powerful tool in MUMPS. Like recursion, there are many who try to use this technique, unfortunately there are few who use it well. This technique is a way that a clever programmer can write un-supportable code. (You will notice I used 'clever', not 'good' programmer.) A waiver is required for the use of this powerful technique.

## ENFORCMENT:

The enforcement of a standard can be a nerve-racking experience for many programmers and their egos. This process can be helped along by the programmers having access to the compliance checking routines. These are the same routines that the quality assurance staff will be using to validate the programmer's style. He should have a waiver ready for each of the exceptions he has had to use. The programmer should be confronted with very few surprises from quality assurance.

The waiver process of this standards system is important. Any standard may be waived (if the standard provides no other way of implementing the logic). The standards effort is an attempt to provide consistent, repeatable techniques for the generation of applications code. The waiver should be intentionally slightly difficult to generate, but definitely not impossible. It should be incentive to develop standard conforming code. The intent of the standard is two fold, 1) quality control and 2) programmer training.

The management staff will notice that the programmers will begin to program in similar patterns. This means that the initial software design staff doesn't have to be chained to the maintenance staff. The documentation staff will like this because there will be less 'clever' code to decipher and any exceptional code has waiver documentation to cover it. See EXAMPLE 5 for a sample waiver.

### EXAMPLE 5:

#### 1. XECUTE Waiver for COMSTORE

```
X CNTRL
1
<***4D XECUTE COMMAND Found ***> START+6^COMSTORE
```

This XECUTE command controls the loading of the selected routines into the routine buffer. Each routine is dismantled one line at a time and stored into a global called ^COMP("TEXT". The second key is the routine name. The third key is a sequential number indicating which line of this routine is stored in this node. There is a series of lines of MUMPS code created within the symbol table. The names and calling order of these lines are elaborated here.

CNTRL - Invokes TRANS and reloads the COMSTORE routine when done.

TRANS - Selects each routine in turn and invokes LOAD.

LOAD - Loads each routine in turn and invokes SAVE.

SAVE - Extracts each line from the routine buffer using TRNT and TXT.

TRNT - Contains a template for loading the routine lines into ^COM.

TXT - This string is modified from TRNT by SAVE.

All of these symbols are killed when this block is exited.

## CONCLUSION:

These are the standards being used by the Computer Sciences Corporation MUMPS programmers at the Naval Weapons Center. These are just a suggestion or a starting point. DO NOT ADOPT THESE STANDARDS, adapt them to your situation.

Be sure to include a mechanism for waiver. There should be a way of circumventing any of these standards, but don't make it too easy to get a waiver. Make sure there is a good reason for the waiver and there is not a way to implement the same algorithm within the standard. Execution speed is seldom a reason for a waiver.

We are working on a routine to serve as a compliance checker. This routine will allow our programmers to develop their own programming style. We will publish this code when completed.

## BIBLIOGRAPHY

- (1) "Productivity now! (Why wait?), by George Proudfoot, Computerworld, Vol. XIX, N. 5, February 4, 1985, page 29.
- (2) "Programmer Productivity, Myths, Methods, and Morphology", by Lowell Jay Arthur, A Wiley-Interscience Publication, John Wiley and Sons, 1983.
- (3) "American National Standard, Information Systems, Programming Language, MUMPS", Draft ANSI X11.1 - 1983.







Leonard J. Moriarty  
Goddard Space Flight Center  
Greenbelt, MD

#### ABSTRACT

This report details the installation and testing of communications protocols and software that are used to link an IBM-3081 mainframe with DIGITAL minicomputers. The application environment in which the computer interfacing is done is scientific processing. User programs and scientific data flow in either direction between the mainframe and the minis. The minis are used for the analysis of spacecraft data that are resident to the mainframe mass storage cartridges and a large tape library. The mainframe runs both CMS and MVS. The minis run VAX-VMS. The current implementation is with the mainframe MVS system. The communications medium is currently twisted pair with Gandalf 56kb modems. The distance between the mainframe and the minis is approximately 300 meters. Both binary and character data are transmitted.

Attached to the Unibus of the minis are custom 68000 micros. Within each mini, there are detached processes running that perform character conversion, protocol translations and queue/job management. The hardware-software system is COMBOARD-HASP marketed by Software Results Incorporated of Columbus, Ohio.

#### Organizational Environment

This paper discusses the implementation of COMBOARD-HASP(CH) at the Laboratory For Extraterrestrial Physics(LEP), Goddard Space Flight Center(GSFC). CH is a computer to computer communications system marketed by Software Results Incorporated(SRI) of Columbus, Ohio.

GSFC is one of the seven NASA sites and is designated as a space flight center. In terms of people GSFC is second in size with 3545 civil servants and 6000 contractors. During the 85 fiscal year \$1.2 billion will flow through GSFC. The implementation to be discussed is within the Space Sciences Directorate of GSFC. Space sciences has 760 civil servants. LEP is one of five laboratories within this directorate. The goals/charter of LEP are the scientific investigation of the planets with emphasis on electric and magnetic fields, plasma physics, radio emissions and astro-chemistry. The current contingent of LEP is 110 civil servants. Of this number 15 are clerical or technicians, 58 are PHDs and the remainder act in technical support of the scientific effort. In addition to the civil servants there are 10 contractors and 8 post docs. LEP has experimental data from over 30 spacecraft.

#### Computer Environment

The Space Sciences Computation Facility(SCF) has recently been reorganized and now includes a IBM 3081 model K, an Amdahl 270 and a Cyber 205 as the primary computation and production facilities. At present the Amdahl and the Cyber are collocated. The 3081 is approximately 600 meters from them. Within the near future the 3081 will be collocated within the SCF computational center. Figure one shows the SCF IBM-3081 facility. The primary in-house LEP computational facilities include a VAX 11-780 and a VAX 11-730. The 780 was acquired in September 80 and was a replacement for an IBM-1800. The 730 was acquired in May 84 and is used primarily in support of laboratory balloon flights. The 780 is used in the development of programs and the analysis of data as derived from production runs on the SCF facilities. The LEP primary computer facilities are shown in Figure two. The implementation of CH to be discussed is with the 3081, designated the central and the LEP VAX 11-780, designated as the remote.

CH is a hardware-software system. The hardware consists of a 68000 based micro attached to the 780 unibus, a 1231 DEC to IBM interconnect four wire twisted pair and two synchronous Gandalf S260 56kb modems. The software system consists of a VAX detached process running a C based software system that in conjunction with the 68000 micro performs job management and protocol transformations necessary for linking the 780 with the 3081. To the 3081 the CH system appears as a RJE station acting in the JES2-MVS environment. The VAX RJE station is configured at the IBM side to have two readers, four printers and four punches. Each punch is of a different class.

The CH command SEND is the basis for the transfer of files from the VAX to the IBM. From the IBM side the JES2 ROUTE card is the basis for directing the central IBM job output to the remote Digital minicomputer. In addition to the rules for an IBM RJE station the following rules apply at the remote side.

- (a) All print files and one punch class requires EBCDIC to ASCII conversion.
- (b) Most VAX files SENT to the CENTRAL require ASCII to EBCDIC conversion.
- (c) All files sent from the remote to the central must be configured to be a valid JCL set.

Rather than attempting to familiarize the LEP staff with the intricacies of JCL the LEP implementation of CH has emphasized the use of VAX COM files that prompt the user for JCL parameters and have the COM files develop the appropriate JOB, DD, EXEC, ROUTE and other JCL related statements. The JCL file developed at the VAX is then sent for processing at the 3081. This does not preclude any LEP users from developing their own JCL files and using the CH SEND command. There are however, local ground rules for data routing that must be followed and they are documented subsequently. The basic COM files developed for the LEP CH system included options for:

1. Routing an IBM sequential file to a user daily scratch space as a print file.
2. Routing a IBM PDS to a user daily scratch space as a print file.
3. Routing a remote file(s) for storage on the central.



Figure 1. Scientific Computational Facility(SCF).

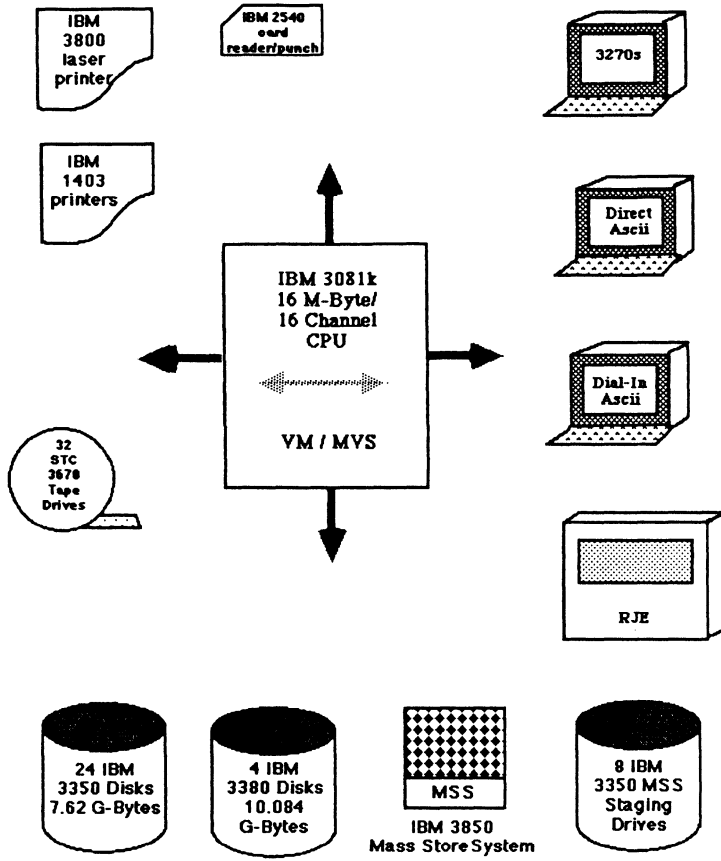
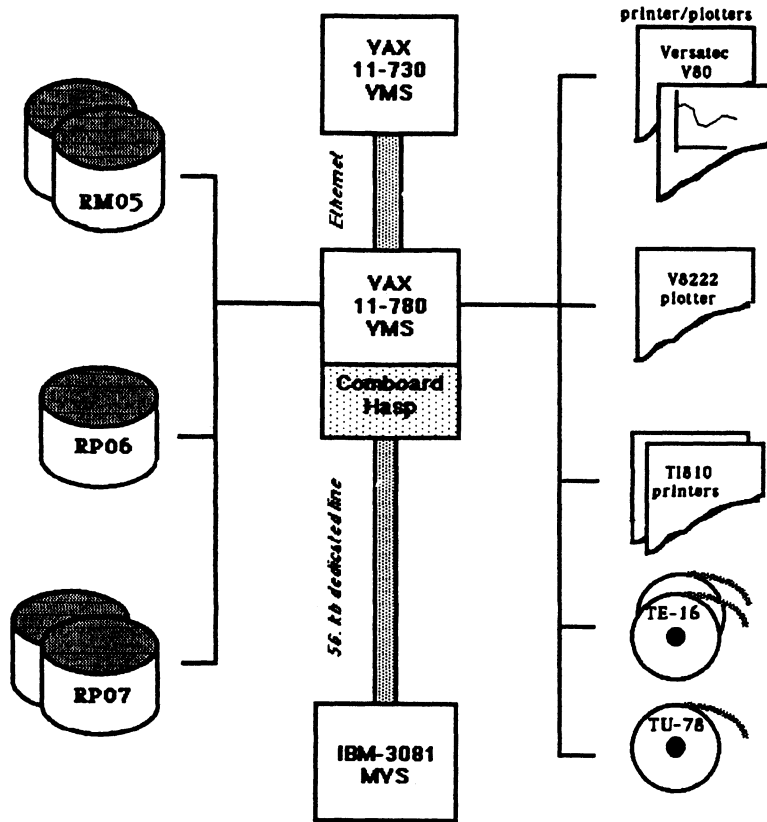


Figure 2. Laboratory For Extra-Terrestrial Physics Computer Facilities



4. Process central sequential datasets (disk or tape) at the central selecting records by type and/or time or selecting all records. Within each record selected the user can specify fields for processing and conversion to DEC format. The records/fields are sent as binary Fixed Block(FB) packets to the remote over SYSOUT classes C, D and E punch files. These classes are not processed for EBCDIC to ASCII conversion. The binary records as formed at the central are then directed to a user specified file at the remote. User file specification is done by two route records. The first route record applies to punch files. The second route record applies to print files and is developed by the IBM banner records that precede print files. The user comment contained on the IBM jobcard is used for routing to disk and/or a printer. A user input comment record is also used to alter the processing functions and generate print records rather than punch records. If the user includes the string "PRINT" in the user comment input record, the items selected are processed as print files and the resultant ASCII printout directed to a user daily scratch space at the remote.
5. Process a DSR remote file(s) for printing on the central IBM 3800 printer. This file can be processed at the remote prior to sending converting DEC form feeds to EBCDIC carriage control.
6. Route remote files to become members of a central PDS. The user can specify the ADD and REPLACE options of the IBM utility IEBUPDTE.

The IBM TSO user can invoke CLISTS whose function is the routing of print or punch output to the remote. These CLISTS create a JCL file and this file is used with the TSO SUBMIT command.

At the remote side, the COM files have been developed to minimize user responses through the use of COM file defaults. In most cases the user can select the default by merely pressing the return key. The experienced COM file user can further minimize responses by including COM file parameters at invocation. These parameters allow the user to:

- (a) Gain experience and create a reference remote file for SENDING by including TYPE or PRINT as the first COM file parameter(P1).
- (b) Include as the second parameter(P2), the numeric value associated with the COM file option as shown above.
- (c) Include as third parameter(P3), the string "NO" and thereby select all defaults. The COM file then omits most prompts. It should be pointed out that including this P3 string requires that the user be familiar with the COM file functions and defaults. It makes no sense to select a default IBM cpu time of ten seconds for a job that might require several minutes.

The execution of option 4, REBLOCK, is the mechanism used to process IBM files and retrieve the output as binary Fixed Blocked(FB) data packets formatted in a DEC format. This option invokes a CH related IBM executable load module which uses the IBM QSAM access method for reading the user specified disk or tape file. This program processes one input file and can generate up to ten punch files of class C, D and E. It requires that the user know:

- (a) The IBM dataset name and/or tape volume serial number.
- (b) That the user can define a record by type or be able to select all types.
- (c) That the user furnish start/stop times or be willing to select any record without regard to time.
- (d) That the user can identify fields within a record type that have an algorithm for DEC conversion. The user selected fields are the only items that are converted and punched.

Before going into further detail let it be pointed out that the record selection and conversion algorithm were based on the LEP data dictionary bible, "Commonly Used Digital Tape, Disk And Card Formats". After reviewing this document, and in particular the requirements associated with VOYAGER Magnetic(LFM) field and Plasma(PLS) experiments, the algorithm for retrieving data and converting it to DEC format was developed. There are approximately 5 gigabytes of VOYAGER I and II, LFM and PLS data stored on the central mass store cartridges. In addition to the data associated with an experiment VOYAGER data contains engineering records(ENGR), trajectory data(SEDR), and header records(HDR1 and HDR3).

The header prefix consist of 20 distinct data elements. LFM consists of 33 distinct data elements. ENGR consists of 2 data elements. SEDR consists of 9 distinct data elements. HDR1 consists of 27 distinct data elements. HDR3 consists of 10 distinct data elements. PLS data word sections have up to 133 distinct data elements.

In order to be able to process these varied data elements, a Pseudo Format (PF) was developed. PFs are similar to but distinct from the familiar FORTRAN FORMAT statement specifiers. The REBLOCK program recognizes eleven type PFs and allows groupings by parenthesis up to a level of three. The PFs types are shown in Table 1. They are the road map and the mechanism by which the user selects IBM based data and converts it to DEC binary data. There are fifteen format specifiers, seven for positioning and eight for data conversion. Table 1. lists these types.

Table 1. Pseudo Format Types

| Type | Description                                                                                                                                                                                                                                                                                                                                                                                                   |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| B    | Reference data location from the beginning of record in bytes.                                                                                                                                                                                                                                                                                                                                                |
| C    | Character data of length one; EBCDIC to ASCII conversion is implied.                                                                                                                                                                                                                                                                                                                                          |
| D    | REAL*8 data converted from IBM to VAX format.                                                                                                                                                                                                                                                                                                                                                                 |
| E    | REAL*4 data converted from IBM to VAX format.                                                                                                                                                                                                                                                                                                                                                                 |
| H    | INTEGER*2 data converted from IBM to VAX format.                                                                                                                                                                                                                                                                                                                                                              |
| I    | INTEGER*4 data converted from IBM to VAX format.                                                                                                                                                                                                                                                                                                                                                              |
| L    | LOGICAL*1 data, no conversion done.                                                                                                                                                                                                                                                                                                                                                                           |
| P    | REAL*4 data converted from IBM to VAX format after having been examined for range. Since the range of IBM floating point numbers is greater than that of the VAX, a validity check is made before the conversion process. A number is rescaled if its absolute value is less than 1.E-37 or if it is greater than 1.E37. The rescale algorithm is<br><br>new value = 1.E37 x natural log of (original value). |
| Q    | REAL*8 data converted from IBM to VAX format with an algorithm similar to that of the P format above.                                                                                                                                                                                                                                                                                                         |
| R    | Reference data location from the beginning of the data record in words.                                                                                                                                                                                                                                                                                                                                       |
| S    | Reference data location from the beginning of the header record in words.                                                                                                                                                                                                                                                                                                                                     |
| X    | Space up by a byte.                                                                                                                                                                                                                                                                                                                                                                                           |
| RB   | Equivalent to B.                                                                                                                                                                                                                                                                                                                                                                                              |
| RD   | Equivalent to S.                                                                                                                                                                                                                                                                                                                                                                                              |
| RH   | Equivalent to R.                                                                                                                                                                                                                                                                                                                                                                                              |

NOTE

Variables using the P or Q format incur a one byte overhead per value as a logical array of length equal to the pseudo format repetition factor precedes the data. This array will flag as true all values that have been rescaled by the defined algorithm. Positive rescaled values are = 1 while negative rescaled values are = to 3. The user in their VAX application program must make any adjustments to rescaled data. The user should also be cognizant that the overhead logical array associated with P and Q pseudo formats is part of the output record which must be less than 8000 bytes.

It is also possible for the user to group format types with enclosing parenthesis and a prefix repetition count. Up to three levels of groupings are possible.

In addition to the user PFs the REBLOCK program requires as input:

1. A value of A, B, E or T for defining what the record selection criteria is. These values equate to All, Binary, EbcDic and Time.
2. The algorithm to be used for time conversions.
3. Per record type the user must furnish:
  - (a) The length of any header prefix data area.
  - (b) The location of a record string identifier.
  - (c) The string pointed to by b.
  - (d) The location of the record time specifier.
  - (e) The record start/stop processing times.

A value of 0 for (a) and (b) terminates the input of this input record class.

4. The user PF string. A PF can be up to 240 characters contained in three records. A double blank signals the termination of a PF for a record type.

This input set then consists of one each of input record type 1 and 2, n+1 of record type 3 and n of record type four. Although a PF can be up to 240 characters, in practice they are generally less than 80 characters.

There is an association of how IBM punch and print classes are processed at the remote. CH has as part of its sysgen input files, an AUTOOPR.CBA file which defines the RJE station at the remote. At the LEP, CH has defined that punch class B and all print files require EBCDIC to ASCII conversion. Punch classes C,D and E are for binary data such as that developed by the REBLOCK program. Table 2 shows the LEP AUTOOPR.CBA file.

Table 2. LEP Auto Operator Configuration File.

```
.I1 /HO /RT=IBM1
.I2 /HO /RT=IBM2 /NOTR /STREAM
.P1 /HO /RT=IBM1 SYSSCRATCH:PRINT###.PR1
/PRO=(O:RWED,S:RWED,G:RWED,W:)
.P1 /HO /RT=IBM1 SYSSCRATCH:PRINT###.PR1
/PRO=(O:RWED,S:RWED,G:RWED,W:)
.P2 /HO /RT=IBM1 SYSSCRATCH:PRINT###.PR2
/PRO=(O:RWED,S:RWED,G:RWED,W:)
.P3 /HO /RT=IBM1 SYSSCRATCH:PRINT###.PR3
/PRO=(O:RWED,S:RWED,G:RWED,W:)
.P4 /HO /RT=IBM1 SYSSCRATCH:PRINT###.PR4
/PRO=(O:RWED,S:RWED,G:RWED,W:)
.P5 /HO /RT=IBM1 SYSSWEEKLY:PUNCH###.PU1
/PRO=(O:RWED,S:RWED,G:RWED,W:)
.P6 /HO /RT=IBM1 SYSSWEEKLY:PUNCH###.PU2
/PRO=(O:RWED,S:RWED,G:RWED,W:)/STREAM /NOTR
.P7 /HO /RT=IBM1 SYSSWEEKLY:PUNCH###.PU3
/PRO=(O:RWED,S:RWED,G:RWED,W:)/STREAM /NOTR
.P8 /HO /RT=IBM2 SYSSWEEKLY:PUNCH###.PU4
/PRO=(O:RWED,S:RWED,G:RWED,W:)/STREAM /NOTR
```

The .I stands for input channels and .P1 thru .P4 designate print output channels. The values .P5 through .P8 designate output punch channels. The CH keywords HO stands for hot, i.e. the channel is active; RT= directs the output to one of two output queues designated IBM1 and IBM2; NOTR designates no translate to ascii; STREAM is used to designate a binary output file; PRO= defines the default protection class for the output dataset. The default file designator is defined to be to HASP account daily or weekly scratch space. The string ### is a designator to CH that a count value associated with a channel should be incremented and this value becomes part of the filename.

Since all punch files are by definition 80 or less characters, variable length records must be reconstructed at the remote. This process is done by having the central include as the first two bytes of the variable length record the record length in DEC integer form. The FB 80 byte packets are then used with a remote based program to form the user specified variable length record. The transformation of the FB to V format is done after CH has received the binary data.

Routing of data to remote user files is done primarily through the identification of route records that are sent with the data. In the case of punch output a formatted record allows the user to specifically route to a remote file. Print files are handled differently. Since all print files are of the same class there is no way to direct print to a remote file definition unit. Instead a banner string that is prefixed by JES2 to SYSOUT=A files is the mechanism for directing printout to use daily scratch space. The user must manage the files routed to the remote. Print files must be copied and punch files either copied or run through a Fixed Blocked(FB) to Variable(V) reformatting program. This program resides within CH space and is accessible to the user.

The punch route record is formatted into 5 fields.

1. The route identification string `***COMBD` (8 characters).
2. The jobname as contained on the user IBM jobcard(8 characters).
3. The channel number associated with the punch file(1 character).
4. A batch submit field consisting of the string "SUB." or "FAS.". Either string designates a batch submit with the later directed to a fast batch queue.
5. The name of a user COM file to be used with the batch submit. This file must be within a user subdirectory `<usrid.COMBD>`. This field is 12 characters long.
6. The fully qualified name of the user file that is to be the destination of the FB data packets.

The punch route card is terminated by a double blank followed by an underscore. The underscore is included to ensure that JES2 punches the two trailing blanks following the file identification. This route card, a user specified comment card, and two REBLOCK generated header cards appear before the binary data. The user comment record is used for additional punch file routing. Normally each FORTRAN `WRITE(n,m)` statement directs the punch output to a different punch class. If the string `MRGHDR` appears within the user comment string all punch output from the n record types are redirected to `CLASS=C`. If the string `PRINT` appears in the user comment string the data elements defined by the user PFs are used to generate a printout of the data, i.e. there will be no binary punch files generated by the REBLOCK program. If the string "NOHDR" appears then only the first route header record is included in the punch output stream. Route headers are included with each punch class and are converted to ASCII at the central prior to transmission. The REBLOCK generated headers(3 and 4) contain information on user input values and input file characteristics as derived from central control blocks.

Print files are identified for routing by the appearance of one of the following strings: `VAX.TO.IBM`, `IBM.TO.VAX`, `VAX.TO.AMDAHL` and `VAX.TO.CYBER`. These strings appear on the user `JOBCARD` in the job comment section and are part of a print banner string included with all `SYSOUT=A` datasets. Once one of these strings has been identified CH installation written software retrieves the remote userid from the section of the banner string that contains the IBM jobname. The first two strings can have suffixed to them the strings `V` or `VD`, which in turn generates a print batch submit to the remote `VERSATEC` printer/plotter. The `D` specifies that the reference print file is to be deleted after printing. A sample use of the REBLOCK option is shown in CH recognizes the batch submit string on the punch file routing card and invokes a user COM file to create a VAX variable length record. In this test case the batch file also

- (a) Generates a VAX variable length record and computes ancillary data parameters. This file is written to user `SYSS$WEEKLY` space.
- (b) The file defined by (a) then passes thru the LEP Interactive Digital Signal Processing program filtering the data.
- (c) The output of the IDSP program is used to generate Versatec plots.

Tables 3, 4 and 5 show the JCL file created by the COM file, the batch-submit file created by CH and the user file used in this SUBMIT.

Table 3. User Data File Used to Retrieve Central Data and Generate a Batch SUBMIT

```
//U2LJMTNR JOB (U0016,BF2,10) VAX.TO.IBM,MSGCLASS=A,
// TIME=(1,59) 18-OCT-1984 10:30
//*ROUTE PRINT R9.PR1
//*ROUTE PUNCH R9.PU1
//*JOBPARM LINES=10,CARDS=60000
//*COMBDSYSECSIBM5nnn<200,057>DR2:<WEEKLY.YSKET>SITTLER.P10
// EXEC PGM=REBLOCK
//STEPLIB DD DSN=UO#08.COMBOARD,DISP=SHR
//FT06F001 DD SYSOUT=A
//FT05F001 DD *
***COMBDSYSKETKIM5SUB.STWEEKLYDR2:<WEEKLY.YSKET>SITTLER.P10
Sample input for retrieving binary central resident data
'E
'DAY'
128 1L 780250000000 80256000000
0 0
4C,3RH,4C,6H,15RH,4L,17RH,2H,37RH,8E,81RH,34E,183RH,2D,46E,
233RH,64H
//FT10F001 DD SYSOUT=C,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
//DATAIN DD DSN=UO#16.VOYAGER.SUMMARY.M10428,DISP=SHR
// DD DSN=UO#16.VOYAGER.SUMMARY.M10437,DISP=SHR
```

Table 4. Batch-SUBMIT File Generated by COMBOARD-HASP

```
$CHECKOPEN:
$OPEN/READ/ERROR=SLEEP TEST DR2:<WEEKLY.YSKET>SITTLER.P10
$GOTO CONTINUE
$$SLEEP:
$WAIT 00:04
$GOTO CHECKOPEN
$CONTINUE:
$CLOSE TEST
$S<YSKET.COMBD>STWEEKLY DR2:<WEEKLY.YSKET>SITTLER.P10
$PURGE/KEEP=2 DUMMY.LOG
$EXIT
```

Table 5. User COM File Used With COMBOARD-HASP Batch SUBMIT (<YSKET.COMBD>STWEEKLY.COM)

```
$ASSIGN 'P1 FOR010
$$SET DEF DR2:<WEEKLY.YSKET>
$COPY DR1:<YSKET.IDSP>PLOTKET.DAT FOR026.DAT
$PURGE *
$RUN DR1:<YSKET>BINFWO
$RUN DR1:<YSKET.IDSP>INPUTKET
$DELETE/NOCONFIRM FOR009.DAT;1
$RUN DR1:<YSKET.IDSP>PLOTJB
$PHASE2
$EXIT
```

The configuration defined at LEP has remained relatively constant over the past six months. The routing CH software is installation dependent. The LEP routing programs consist of seven FORTRAN based routines with a total of three hundred lines of code.

In summary the LEP CH has been in operation as documented for approximately one year and has met our goals/requirements for data availability and retrievability. The 56kb data line runs at approximately 70% of capacity; an actual data rate of 40kb. The chief uses for the CH facility have been:

1. Retrieval of scientific data stored within the main frame mass store and tape library.
2. Quick turn around of IBM printouts.
3. Sending Dec Standard Runoff files to the main frame for printing on the IBM 3800 laser printer.
4. Code development on the VAX with the debugged code then sent to the IBM 3081 for production use.



R. Friesen, A. Simmons  
Lawrence Livermore National Laboratory  
Livermore, California

J. Helton, R. Schell  
EG&G Las Vegas Area Operations  
Las Vegas, Nevada

### ABSTRACT

We have implemented a distributed network to process and archive digital video images from multiple cameras. The system consists of a high speed dual-ported memory controlled by an LSI-11/23 processor for each camera. The CAMAC serial highway is used to "buss" the LSI-11 front-ends to a single LSI-11 "master." The "master" processor analyzes and archives data gathered by the front-end processors.

### INTRODUCTION

A new data capture system was needed at the Nevada Test Site to record video images from cameras looking at downhole nuclear tests. The system built consists of several CAMAC crates, each with its own LSI-11 local processor. One of the crates is a "master" crate and the others are "acquisition" crates connected to cameras. It was required that this new system be able to capture approximately .5 mb of data per camera, have a battery backed memory, be easy to set up, be self-diagnosing, have data compatibility with LLNL data reduction codes, and be "bullet proof." This paper will cover the development of a CAMAC based LSI-11 network to fulfill these requirements.

### SYSTEM CONFIGURATION

The system configuration was designed to support a three phase fielding sequence. The set-up and testing phase, called "dry-runs," requires up to two months and involves loading test data and "capturing" it. During the "event" (underground nuclear test), the acquisition crates capture data autonomously. The data retrieval phase is almost identical to the dry-run phase except for added precautions to protect the real data.

At event time, the system has several unique and interesting features. The system is unattended, with all operations performed from a control point several miles away. To help insure reliability, no single point failure may prevent all cameras from recording data. The Acquisition crate CPU is a state machine and will remember its state through a power interruption. It will power up to the data capture mode before the event and to a "safe" mode after the event for data retrieval. We plan for a power loss at "zero" time with the crate power supply able to hold up the system for about 50 ms. The camera is destroyed about 5 ms after "zero" time. The data recovery is delayed until the area is safe for workers to re-enter the area.

The Master crate is set up as a standard computer system with a touch screen terminal, 1 mb and 10 mb floppies, color graphics monitor, screen copier, and the CAMAC serial highway interface. The serial highway is a loop passing through each Acquisition crate. Each crate has a crate address and each module in the crate has a module address. Each Acquisition crate is configured with a serial highway crate controller, an LSI-11 CPU, an MRV-11 with ROM, battery backed RAM, high speed battery backed RAM, and a camera interface. The data capture timing signal is generated by an external system and routed to each Acquisition crate.

The physical configuration of the serial highway as a loop would normally imply a token passing, or similar type of network. However, the logical configuration of the network is somewhat different. The serial highway is logically configured as a star, with the Master able to connect to any of the Acquisition crates and the Acquisition crates unable to connect to each other.

At event time, the logical configuration is changed. The Master crate is disabled and the network disappears, so that each Acquisition crate becomes a stand-alone system. The local CPU is used only to monitor the system state, select the appropriate memory bank, and arm the high speed memory interface to be ready to take data. The high speed memory records the camera data under control of external timing signals.

### THE CAMAC NETWORK FEATURES

Before choosing the CAMAC serial highway, we looked at several other networks. DECNET was difficult to implement in the space we had available and required RSX or VMS as the master operating system. We wanted to keep the operating system as simple as possible to promote system reliability and minimize software maintenance. The GPIB require multiple connections which increase the probability of failure. We looked at communication boards like the

-----  
\*Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract number W-7405-ENG-48.

Computrol and Corvus, but they made it difficult to move to fiber optics, a feature we wish to add in the future. We could not find an "off-the-shelf" RS-232 serial network protocol.

The CAMAC serial highway mapped reasonably well into the ISO Network Standard. The application, presentation, and session layers we hard coded into the application code. We wrote the transport layer. The network layer is the CAMAC serial highway protocol and is implemented in hardware. The data link layer is the CAMAC message format with crate and module addressing. The physical layer is the bit serial multidrop loop.

We were able to implement standard network features with the CAMAC network. We can download diagnostic codes and do error logging in the Acquisition crate processor. We can "halt" and "re-boot" the Acquisition crate and collapse the serial highway loop for debugging. The serial highway is a hardware system and powers up to a known state, as does the Acquisition crate processor. The Acquisition crate processor can read the previous state from the battery backed memory and go to the correct next state.

The CAMAC network environment allowed us several operational advantages to enhance the final system. The ability of the master to deactivate the serial highway for the event configuration reduced the potential for erroneous commands to occur at event time. With each Acquisition crate acting autonomously at event time, no single crate failure can prevent the others from capturing data. The serial highway "bypass" prevents a failed crate from interfering with data retrieval from the other crates, and the battery backed memory can be physically removed from a failed crate for possible data recovery. The CAMAC Auxiliary buss allows either the Master or the Acquisition crate processor to directly address any module in the Acquisition crate and the CAMAC serial highway has hardware error detection. The modules needed to construct the system are available "off-the-shelf" and the software is modular and can support a variety of configurations.

Although the system has been in successful operation for several events, there are several things

about the current system we would like to improve. The serial highway requires more supporting modules than we like, and the hardware error detection is not as comprehensive as we would like it to be. Also, the serial highway byte to bit mapping is not fast enough to support future requirements. Possible changes we are currently considering are changing the serial highway to a GPIB bus or embedding a FALCON in a CAMAC crate controller. The number of connections in the GPIB bus are a problem, but there is a large software library to support GPIB and the bus is faster. The most attractive option would be the FALCON in a CAMAC crate, if we could work out the network reliability issues and if the serial port can be used as the network connection.

#### SUMMARY

The CAMAC serial highway is useful in the creation of computer networks even though it has a few weaknesses. It maps well into the ISO standard and is well supported in industry. The supporting software is not difficult to write, and a diversity of configurations are possible. However, the number of modules required to implement it and the speed of data transfer need to be considered when designing a new system.

#### DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade, name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government thereof, and shall not be used for advertising or product endorsement purposes.

# Techniques of Protocol Validation

by

William T.C. Kramer  
University of Delaware  
192 S. Chapel Street  
Newark, DE 19716  
(215) 793-2410

## ABSTRACT

This paper investigates the state of protocol validation and describes a number of methods now in use. The emphasis of the discussion is toward automating these methods so the validation process may be improved. Seven techniques for protocol validation are discussed; finite state machines, and the related methods of directed graphs and petri nets are evaluated first. Then programming and special languages and hybrid techniques are studied before simulation and temporal logic.

This paper was done under the direction of Dr. Dave Farber and Peter von Glahn in conjunction with class work at the University of Delaware.

### 1. Introduction

In recent years, communication protocols have become increasingly complex. It has become necessary to develop formal rules for specifying these protocols. Along with formal specification comes the need to develop methods to validate that a protocol specification will perform the services it claims, and the design itself does not have any flaws. This is particularly important in network protocol design, since there will be distributed and independently developed implementations of the protocol will be used. It becomes very difficult and expensive to correct design errors after the specification is release.

The first section discusses some of the implications of validation and verification. These are closely related terms and are many

times used interchangeably. The next section discusses the attributes of a protocol which may be tested and hopefully validated. These include absence of deadlock, liveness, safeness and other problems which may come up in a communications protocol.

Each type of verification technique is then discussed separately. First the large area of Finite State Machines is broken down into subtopics, including directed graphs and Petri Nets. For each technique, a discussion of the automation of the technique is included along with mention of actual protocols which have been verified. After finite state machines, programming and special languages, hybrid techniques, simulation techniques and temporal logic techniques are discussed in the same mode.



## Protocol Validation

### 2. Validation and Verification

The terms **protocol validation** and **protocol verification** are used interchangeably in some instances. Regardless, they have an overlapping meaning and purpose. It is worth while to start the discussion by explaining the difference between these two terms. It is first necessary to look at what is specified in a protocol specification. First, there is a specification of what services the protocol is to provide to its user. In this case "user" means either a person, a program or another layer of the communications protocol. This specification is called the **Service Specification** [Sunshine79]. There is also the **Interface Specification**, which is a descriptions of how to implement the service specification for a particular computer or system. Verification is the checking of the interface specification against the service specification from which it was derived to show that the implementation does indeed provide the services which were specified.

Validation is insuring that a protocol satisfies the design or Service Specification and thus will satisfy the needs of its users. [Bochmann80A] The validation process determines whether or not a protocol has a sound logical structure. Validation attempts to identify design errors in the protocol which result from incomplete definition of the design. These errors cause unpredictable behavior of implementations of the design [West78A].

The reason these terms are sometimes interchanged is that a protocol verification of an implementation presupposes a properly designed service specification. A protocol may not be validated, so when an implementation of the protocol is verified, it may uncover design errors. Adding to the confusion is the problem that many protocols have not been specified in a formal manner. Thus, to try any validation of the entire protocol, some sort of implementation or interpretation of the service specification is necessary. In some cases, even the validation method is actually verification an implementation of a kind. As protocols are designed in a more formal method in the future, the difference between the two terms should become clear.

### 3. Properties which may be Validated

Validation properties may be divided into general and specific properties. All protocol designs should have the general properties, which are considered an implicit part of the Service Specification. An example of these properties is freedom from deadlock. Specific properties are dependent on the service the protocol is to provide. These could be the proper clearing of a terminal screen in a virtual terminal protocol or the correct copying of a file in a file transfer protocol. These specific properties are generally defined in the Service Specification.

Beyond the specific service specification There are a number of general properties which are important to validate in a protocol specification. In the following discussion, the properties are described in terms of a finite state machine. However, these properties may be validated in any method of protocol specification. In some cases, such as temporal logic, several properties may be stated with one assertion.

The first property is that the specification does not allow any **deadlock** states that may be entered. A deadlock state is one from which an implementation can never emerge. These states may be those which, once entered, do not have any events which would cause them to exit or there may be states whose exit events can never be triggered. It must be recognized that a deadlock state is not necessarily an error, since it may represent an acceptable terminal state of the process [Zafiropulo80].

There is also the problem of distinguishing between **partial correctness** during validation and **progress** or termination [Bochmann80A]. Partial correctness means that, if the protocol preforms any action, it will be in accordance with the service specification. However, there is no guarantee that any progress will be made or that the protocol will do anything at all. Progress or termination means that the protocol will eventually provide the services specified in a finite time, although the service may not be correct.

The second situation which may be validated is that of **liveness**, or progress. This means that from each reachable state, any

## Protocol Validation

other state in the system is reachable. [Merlin79] It should be additionally remarked that for each reachable state, there exists another reachable state from which this transition event can occur. Basically, this property specifies that there will be no long term cyclic behavior specified during which useful activity does not take place. It also sometimes includes the idea that there are no deadlocks.

The general term **correct execution** is used to cover several related properties. Basically, the specification should behave properly in all instances (handling errors, not creating errors and properly maintaining synchronization with other processes.) **Completeness** in the specification requires that any specification handle all possible conditions. A process in a particular state must be prepared to accept any transitions which the other processes could possibly generate. **Unspecified receptions** to a process might cause unpredictable results in implementations [Merlin79].

**Self synchronization** or the absence of state ambiguities is another general property of a protocol. The different communication parts of the network must be able to synchronize their states together. In addition, the processes must not be able to fall out of synchronization permanently, which could happen if transitions are allowed between process states without any messages flowing between them.

Another area under the general heading of correct execution is the need to have a mechanism in the protocol specification which prevents the communication channel (including the physical connection, buffers queues, etc.) from overflowing. This mechanism is also known as providing flow control and may be as simple as a transmit/receive acknowledgment or a hardware signal. In temporal logic, these properties are grouped together under the term **safeness**.

Other basic properties which all protocol specification should have are the ability to **properly terminate**, the idea of **fair scheduling** of resources and no **unnecessary states** or specification. Making sure that all the processes which make up the protocol reach a proper terminal state helps insure the processes synchronize and also that all users will be able to terminate. Fair scheduling is

necessary for good performance of a protocol. If several users are all trying to pass data over one hardware link, the protocol must insure all the users are receiving part of the resource. Of course, it is never desirable to have states which may not be entered under any condition.

## 4. Approaches to Protocol Validation

Seven different approaches to validation of protocol specification will be discussed; Finite State Machines, Directed Graphs, Petri Nets, Programming and Formal Languages, Hybrid Models, Simulation, and Temporal Logic. With the exception of the simulation studies, all these techniques require that the protocol be specified in a formal manner appropriate to the validation method. There has been a great deal of work done to build systems which will allow for a protocol specification and automated validation of the specification. Some of these attempts have proceeded to automatically generate protocol implementations as well. A brief description of each technique will be provided, followed with discussion of advantages and disadvantages. Comments about automating the techniques and attempts to validate actual protocols will finish the discussion.

### 4.1. Finite State Machines

Some of the first attempts to use formal methods to both specify and validate protocol specifications dealt with finite state machines. The finite state machine consists of a diagram or description of each state a process may be in, the input or reasons the state was entered, and all the output produced by the current state and the reasons the state will be exited. A "state" in the machine denotes some independent processing states of the machine. This method was familiar to protocol designers from other areas of computer science. It works well with simple protocols, which can be thought of as a series of relatively simple processing steps, each responding to some input to trigger a transition to a new state [Sunshine79].

There are a number of severe limitations to the finite state machine method. First is the problem of an "explosion" in the number of states. Since finite state machines have no

## Protocol Validation

capacity for memory, each state must be independent. Thus, if a protocol which employs a series of sequence numbers is to be tested, a complete set of states for each sequence number must be specified. Indeed, if there is no limit on the sequence numbers, then the machine will not longer be finite. Second, other common attributes of protocols may be difficult to specify. For example, timeouts are generally considered to be internal events, but in a finite state machine, there would have to be a separate external machine specified to produce timeouts and the actual communicating processes must allow transitions triggered by the timeout input.

A third problem is that there are at least two processes communicating in any protocol specification. Each of these processes are finite state machine, which may have identical specifications. These machines are treated as cooperating processes, not identical ones since one will be transmitting and the other receiving at any given time. For validation, it is difficult to synchronize the two finite state machines.

Even with these problems, finite state machines are common models for protocols. Some extensions to the model have been made to help overcome its limitations. There are also well known procedures which are beyond the scope of this discussion which may be used to reduce the number of states whenever possible.

**4.1.1. Automation of Finite State Machine Validation** Regardless of the limits of the method, a number of automated protocol validation systems are based on the concept of finite state machines. Most systems allow for a finite state machine to be described as a 5-tuple [Danthine80]:

$$\langle X, I, O, N, M \rangle$$

where

- X is a finite set of states
- I is a finite set of inputs
- O is a finite set of outputs
- N is a state transition function -  
    given an input it specifies a  
    state to enter
- M is an output function -

given a state and input it  
specifies an output

From these descriptions, two or more finite state machines may be combined into a global state transition diagram. Essentially, a new machine is constructed, which has all the possible combinations of the states of the two other machines. In addition, a global finite state machine may be produced which combines to two finite state machines.

**Reachability analysis** is an exhaustive exploration of all the possible interactions of two finite state machines [Sunshine79]. A composite system must first be generated, which is the combination of the states of both processes, and the messages transferred between them. machine state is a 2X2 matrix, with the state of the communicating processes on the diagonal and the messages in the other elements [Zafiropulo80]. The matrix would be:

$$\begin{array}{|cc|} \hline \text{P1 state} & \text{P1 to P2 message} \\ \hline \text{P2 to P1 message} & \text{P2 state} \\ \hline \end{array}$$

This method may conceivably be extended to more than two process communication, but the number of states grows (even with only two processes) in an exponential manner.

**Perturbation analysis** is a variation of reachability analysis in which a reachability diagram is constructed by starting from the initial state and generating all possible combinations of transitions to other states. Those states are then perturbed to new states. In the end all the paths should return to the initial state. Reachability analysis or perturbation analysis allow most of the general properties of a protocol to be validated. Deadlock detection is relatively simple, since it is a case where a state has no further transitions and no messages. Unspecified receptions are indicated by states that have no departing transitions to accommodate an input.

### 4.1.2. Actual Protocols Validated with Finite State Machine Methods

#### 4.1.2.1. X.21 Protocol

A number of automated protocol validation systems are based on the finite state machine approach. The call establishment phase of the CCITT X.21 protocol has been

## Protocol Validation

validated using a state transition model and reachability analysis [Bochmann80A].

### 4.1.2.2. IBM's System Network Architecture (SNA) Protocol

IBM's entire SNA protocol was described and validated using a finite state machine method with extensions for the description of data control [Schultz80]. It is worth looking into this particular method, since the approach is typical of protocol validation work. The protocol was described in a programming type language (FAPL), which is an extended PL/I. The extensions to PL/I allow three types of structures to be easily specified; finite state machines, data entities and queues. Finite state machines were specified with a state transition matrix similar to the one described above and was translated into a PL/I procedure. Data entities and list handling facilities were the basic constructs for the data element description. Queues were handled as special case first in first out lists. Combining these added structures, the FAPL language was capable of completely specifying SNA.

Passing the FAPL specification through a pre-processor translated it to standard PL/I. Not only could this translation be used as the basis of a protocol implementation, but it was also be used to automatically validate the design. The validation technique used the state perturbation method. The processes were modeled as finite state machine and the channels connecting them were modeled as queues. The validation detected deadlocks, inconsistent and incomplete design, or the loss of synchronization.

The validation procedure is to run as two half processes, one called the primary which may send requests and initiate the connection. The second half process may only respond to the primary. During the validation procedure, histories of the channel interaction are kept so that particular sequences which lead to improper behavior may be studied. This has the additional benefit of being able to test implementations with the same sequences. Some of the errors detected by the validation process were shown to occur only under peculiar time situations.

### 4.1.2.3. VADILOC and HDLC

The VADILOC system is also based on finite state machines, but with the additional concept of predicates [Rafiq83]. It is an interactive system which describes states in the machine as

<Event From-State Predicate To-State Action>

The validation is done by studying two entities, each as described in the above format, while they communicate over a channel, using reachability analysis. Deadlocks, unspecified errors, and non-executable transitions are detected as errors.

The VADILOC system has been used to validate the ISO transport protocol (classes 0 through 3) including error recovery, the ECMA synchronization/resynchronization phase and an HDLC-like protocol [Rafiq83]. The system has the added feature of being able to produce skeleton programs from the system description in Pascal, ADA and PDIL.

## 4.2. Directed Graphs

Zarifopulo proposed that two finite state machines which interact may be viewed as the interaction between two directed graphs. [Zarifopulo78] The nodes of the graph represent the states of the process and the arcs in and out of the nodes represent input transitions and output respectively. Graphical representation of the interaction is shown using phase diagrams which are maps of the states the machine enters while it is working. They differ from finite state machines because they explicitly show how many times a state executes.

From the directed graphs, segments may be extracted called unilogues which show a complete path leaving and then returning to the initial states. When the unilogues of two communicating processes are combined, they are called a dialogue.

### 4.2.1. Automated Validation with Directed Graphs

A procedure using a dialogue phase diagram held in a matrix is used to automate the evaluation of protocols specified with directed graphs. The rows of the matrix correspond to the states of one process and the columns correspond to the states of the other. The values inside the matrix are used to

## Protocol Validation

indicate the transitions from each state. It is possible to trace the execution of the processes and derive correctness properties from this matrix [West78B].

Another approach to directed graph analysis is to consider a graph model in three domains: control flow, data flow and interpretation. The protocol is modeled in each domain and all three are analyzed simultaneously [Razouk80]. Control flow and data flow are directed graphs. The control flow graph consists of nodes and arcs. There are AND and OR specifications for the transitions arcs to allow for a node being entered when any input for the node is triggered or only when all arcs are triggered. The data flow graph consists of data sets, controlled and uncontrolled processors and data arcs. The purpose of data flow graph is to model the changes which take place in the data. The interpretation is simply a description of all the data formats.

### 4.2.2. Actual Protocol Validation with Directed Graphs

The X.21 specification was also validated using the dialogue phase diagram techniques which correspond to directed graphs. The validation was carried out using a APL program [West78B]. The validation program took advantage of some APL features for optimization, and used less than an hour of CPU time on an IBM 370 model 158.

The X.21 protocol was also validated with other systems, one of which was SARA [Razouk80]. This system allows the control flow and data flow models to be expressed in a pseudo-language used to described graphs. The interpretation is done using PL/I data definition statements.

SARA has been used to validate the CCITT X.21 protocol. The validation showed a number of ambiguities in the specification, which mostly were due to a process reaching a state which had no provisions to receive incoming signals. The X.21 protocol was modified to add states and transitions to remove most of the ambiguities. With these additions, the modified protocol was shown to properly terminate and have the liveness property.

The above analysis of X.21 was carried out with the assumption of an error free communications medium. SARA was modified to take into account certain errors such as lost signals and altered or undefined messages. The number of states grew rapidly with the added complexity, but the time to describe and study the system grew linearly with the number of states.

### 4.3. Petri Nets

A technique which was developed to address some of the limitations of the finite state machine and state transition models discussed above is the Petri Net. Petri Nets were initially used to study interaction between concurrent or parallel processes. Their advantage over finite state machine machines is that Petri nets express the idea that an event in one process will not occur until a particular event or events in the other process occur. Despite this additional capability, Petri Nets suffer from two problems, also seen in finite state machine models [Berthelot82]. First, any moderately sophisticated protocol leads quickly to an extremely complicated and unmanageable graph. Reachability analysis quickly becomes impractical because of the "explosion" in the number of states involved. Second, some common techniques of programming and protocol design, such as sequence numbering of messages leads to complicated Nets.

Nevertheless, Petri Nets are useful in the analysis of protocols. Petri Nets may be thought of as a 4-tuple [Danthine80] in the following form:

$$\langle P, T, I, O \rangle$$

where

- P is a set of places or conditions
- T is a set of transitions or  
events which become enabled
- I is a input function
- O is an output function

The Petri Net consists of places, which are nodes, and conditions or transitions, which are events. Both are connected with directed arcs. Places are connected to a transitions, and transitions are connected to places. A

## Protocol Validation

transition is enabled if all the inputs to that transition have at least one token. The "firing" of transition means that one token is removed from each input to the transition and one token is placed on each output from the transition.

Petri Nets are also extendible. They can include the important concept of timing within a protocol specification. For example, two time variables may be added to each transitions of the Net, to indicate the minimum and maximum time allowed for the transition to fire, once it becomes enabled [Danthine80]. It is also possible to describe relationships between actions and data (the tokens) by adding predicates to the Petri Net description [Diaz82].

Another extension is the predicate/transition Nets, which can be thought of a summations of large ordinary Petri Nets [Berthelot82]. These nets have tokens which are made up of constants and variables. Each transition has predicates which relate the constants and variables and thus allow the transition to fire only with certain combinations of values.

### 4.3.1. Automation of Petri Net Validation

Petri Net theory allows for the validation of properties of a protocol by methods other than reachability. Reduction methods and linear invariant methods may both be used for validation. These methods were developed for general Petri Nets and have been automated [Berthelot82]. Starting with a two separate Petri Net descriptions for the two communicating processes, a global model is obtained. This is done by connecting places and transitions with the same labels in the separate nets. Then reduction rules are applied to the global model, which will maintain the properties of boundedness, liveness, safeness and linear invariants. The reduced net can then be described as a matrix with rows for the places and columns for the transitions. A positive one indicates an output from a place to a transition and a negative one indicates an output from a transition to a place. An automated Petri Net analyzer is reported in [Kujansu82]. Written in PASCAL it uses reachability analysis to analyze Petri

Nets stored in a matrix format.

### 4.3.2. Actual Protocols Validated with Petri Net Methods

Few actual protocols have been validated with Petri Net techniques. The European Computer Manufacturer Association (ECMA) transport protocol, which is between the session and network layers of the ISO model has been validated [Berthelot82]. Petri Nets have also been used to stochastically model Ethernet to investigate the waiting times for network requests for transmissions [Florin83].

## 4.4. Programming and Special Languages

Programming languages are a natural way to specify and validate protocols. Specifying a protocol in a programming language is much like implementing a specification. Indeed some of the protocols specified are almost directly translatable into a program. Protocols may also be specified and validated using formal grammars and with special languages.

Languages, whether common computer languages and their derivatives, or special languages such as executable logic statements are well able to handle many elements of protocol implementation, such as variables, parameters and sequence numbers. However, programming language specification are very dependent on the style and abilities of the programmer. Specifications are also difficult to minimize.

Protocols specified in a programming language are many times validated with simulations. The creation of the process specifications is helped by the program language and then simulation techniques may be used to evaluate them.

### 4.4.1. Automation using Languages

The problems of validating protocols written in programming languages is the same as that for proving correctness in general programs. Much work has gone into this area, with some results. However, all the desired attributes of a protocol cannot yet be automatically validated. When using formal

## Protocol Validation

grammars already established techniques are useful to investigate the correctness of a protocol.

Special languages have also been developed for protocol specification and validation. These languages are specifically designed for the problems they address. One such language is a special calculus which creates a precise mathematical model for concurrent processes [Aggarwal83A]. This language defines functions in terms of states and transitions. The description is then formally reduced to keep the number of states manageable. Another reduction technique, called homomorphism, reduces the description by looking at probabilistic or timing information to eliminate unlikely transitions [Aggarwal83A]. This type of formal approach allows for exact specification and reduction, eliminating many of the problems which are associated with finite state machines and general programming languages.

### 4.4.2. Actual Protocols Validated using Languages

The transport protocol for the French CYCLADES computer network was validated using the programming language approach [Sunshine79]. In a separate investigation, [LeLann78] used simulation to evaluate the Cyclades protocol with respect to performance and efficiency which was specified in a programming type language.

### 4.5. Hybrid Methods

Many useful attempts to combine the above methods have been made in an attempt to eliminate some of the problems with each technique. One of the most common approaches is to combine a finite state machine specification for the control flow and a programming language to describe the data flow of a specification. The final models generally are more closely related to one or the other of the methods.

#### 4.5.1. Actual Protocols Validated using Hybrid Methods

HDLC has been validated using hybrid methods, and parts of the X.25 specification were also evaluated [Bochmann80B]. In the HDLC case, the protocol was first divided into

small sub layers within each layer of the specification. The control flow of the protocol was modeled using finite state machines and the data flow with a type of programming language. Global program variables were also described with a type of programming language [Bochmann80B].

The protocol was verified by using a program assertion technique. An assertion invariant, which is some type of boolean expression, is assigned to the states of the system. The assertion is true when the system is in a given state. This approach served to validate the protocol in terms of partial correctness and liveness.

### 4.6. Validation through Simulation

Validation through simulation of protocol activities differs markedly from the other methods so far discussed. First, there must be actual implementation of the communicating processes to work with the simulation. This means that the protocol specification must in a sense be implemented, which is of varying importance, depending on which method was used to specify the protocol and how detailed the implementation is. The implementation may range from something generated from a formal specification to something which is a complete implementation. Of course, the implementation process increases the chance for error, since there may be

A greater problem is that the correctness of a protocol can not be proven, only disproven. Problems with the specification may be found related to all the areas discussed. There is no proof that all the errors in a protocol specification are found with simulation, because simulation can not be exhaustive. Simulation has also proved to be expensive when carried out to the degree necessary for a reasonable assurance the protocol lacks errors [Remes82].

On the other hand, simulation has several benefits which are not available in other methods. There is a history of the interaction between the processes generated. This history may be used to test implementations as well as specifications. In addition, the work done to study the specification may be carried to testing the actual implementations. The greatest benefit in simulation is that it

## Protocol Validation

can give indications as to the efficiency of the protocol as well as indication of its correctness. During simulations, parameters which indicate the "responsiveness" of the protocol may be measured, either absolutely or relatively. As the specification changes or is updated these response indicators may be compared. In addition, different specifications or implementations can be tried and decisions made as to the best specification not only based on correctness but on performance.

### 4.6.1. Automation of Simulation Validation

Unlike other validation techniques, many of which were developed manually and converted to automated methods, simulation techniques have been primarily automated. The basis for a good simulation study is a thorough understanding of the protocol and some assumptions about the behavior of the network [Remes82]. Most protocols may be thought of processes which deal with queues of data. Requests for services, channels connecting communicating processes, and the output from services are the queues. These queues may have different statistical properties. The simulator's task is to identify the queues, assign the correct properties and then generate the queue interaction with the processes.

Simulation also allows the study of a protocol for properties which are beyond the idea of formal correctness. These are specification dependent properties, instead of general properties, although most specifications will have the same properties with different limits. These attributes include fair network service, guaranteed network service and delay times [Geissler82]. Fair scheduling means that the flow control mechanisms of a protocol will allow all the users of the protocol to receive fair and approximately equal service, within their own level of priority. Some protocol specifications, such as X.25 allow for users to modify the parameters of their process. These parameters include window size, priority class and others. When this is done, the protocol specification must still allow for the requested service to be provided. It can be shown with simulation that such protocol models can dramatically effect the performance of the network with

regard to these parameters. Delays are also important in any protocol because they will cause timeouts and indicate the perceived performance of the network. In packet networks, packet delay is important and is closely related to overall throughput of the network. This has been studied with simulation [Geissler82].

Protocol simulation studies may use special simulation programs which are designed for the specification and simulation of general networks models. Special simulation packages which allow for the simulation of parts of a network exist as well [Remes82]. There are also special purpose simulation languages such as SIMSCRIPT and GPSS. These languages provide convenient facilities to model queues and other items generally found in a simulation model. However, use of these languages usually involves the translation of the specification into the special language, thus introducing room for error. Lastly there is the use of programming languages to implement a specification and simulate its activities. This is generally tedious work and worthwhile only if exhaustive simulations will be done.

Simulation also allows the study of a protocol for properties which are beyond the idea of formal correctness. These are specification dependent properties, instead of general properties, although most specifications will have the same properties with different limits. These attributes include fair network service, guaranteed network service and delay times [Geissler82]. Fair scheduling means that the flow control mechanisms of a protocol will allow all the users of the protocol to receive fair and approximately equal service, within their own level of priority. Some protocol specifications, such as X.25 allow for users to modify the parameters of their process. These parameters include window size, priority class and others. When this is done, the protocol specification must still allow for the requested service to be provided. It can be shown with simulation that such protocol models can dramatically effect the performance of the network with regard to these parameters. Delays are also important in any protocol because they will cause timeouts and indicate the perceived performance of the network. In packet



## Protocol Validation

networks, packet delay is important and is closely related to overall throughput of the network. This has been studied with simulation [Geissler82].

### 4.6.2. Actual Protocol Validation with Simulation

A number of protocol specifications have been studied with simulation to check the specification and to give some idea of the performance of the model. Ethernet has been studied to evaluate its performance for industrial process control [Florin83]. The first step in this study was to collect statistics from typical industrial application as to message traffic load. Then a model of Ethernet written in the special simulation language SIMULA was used to study the network response to the modeled conditions, which were asserted in terms of probability models. The results of the simulation revealed the upper limits on the response time for a request and the number of maximum request per second.

The Cyclades network was also studied with simulation techniques. The studies showed a number of features about the protocol including the fact that there is no absolutely reliable technique for opening or closing connections because synchronization may be lost due to transmission uncertainties [LeLann78]. In addition, other aspects of the protocol specification were studied to ensure fair service. This included investigating the mixture of short and long packet sizes, and the effect of "channel stealing protocol" which allows empty channels from one category to be used by another, and a different channel control scheme which used no pre-allocation.

### 4.7. Temporal Logic

The use of temporal logic to specify and validate protocols is relatively new. Temporal logic is a formal language which is designed to express time dependent operation. It grew out of the problems associated with using general languages to describe protocol specifications, and the inability to clearly separate the necessary features of the protocol from the incidental needs of the language chosen to implement it.

Temporal logic contains two basic control operators - henceforth and eventually -

combined with the notation that control may be at, in, or before a statement. Temporal logic is used to specify a system's actions over time [Sunshine82]. This makes validation of a specification with respect to the properties of progress and safeness easier than with other methods. Additional operators have been added to temporal logic systems to make the specification of common protocol techniques easier. These include the **until** and **latch until** operators. With these, it is relatively simple to specify concepts such as having sequence numbers on messages increase, while not specifying the actual starting number. This type of concept is difficult to describe with other methods such as finite state machines [Schwartz81].

State Deltas is a validation approach which makes implicit use of temporal logic [Sunshine82B]. The basic unit of the specification is the state delta, which contains a precondition, a modification list and a postcondition. The theory says that if the precondition becomes true, then at some later time, the postcondition will also become true. In the interval, the variables listed in the modification list may change. [Sunshine82B].

### 4.7.1. Automation of Temporal Logic Validation

To prove the correctness of protocol specifications using temporal logic, proving of program assertions is needed. Assertion proving has been used to validate protocols and other systems specified with of languages. Special systems have been developed for the proving of program assertions using temporal logic. Generally three steps are needed for this type of approach:

- 1) Formal specification of the protocol
- 2) Choosing the assertions needed to prove correctness
- 3) Proving the assertions are true

Most automated systems require the first and second steps to be done manually and the automated system does the last step. Generally two properties are asserted for correctness of a protocol - the Liveness and Safety Properties. (Safety corresponds to the correct execution property [Sabnani82].)

## Protocol Validation

### 4.7.2. Actual Protocols Validated with Temporal Logic

No major protocols, on the order of SNA or X.25, have been validated using temporal logic. However, automated systems have been used to validate theoretical protocols. A multi-destination protocol was specified and then validated using an ALGOL-like language for the specification [Sabnani82]. The protocol was validated, using the assumption that there will never be two packets being transmitted at the same time which have the same sequence number. It was also shown that specification will loop forever if there is a complete breakdown of the communication channels. This problem is corrected by adding provisions for timeouts of the sender and receiver.

Using temporal logic, Kurose showed the correctness of a three way connection establishment protocol which was described using an extended PASCAL [Kurose82]. The total correctness of the protocol described is concisely stated as:

*"Eventually, both processes will enter and remain in the connection established state and will have agreed on the sequence numbers of the messages to be transmitted."* ([Kurose82] p 48.)

This was verified in the described protocol using semi-automatic methods.

There is an automated system, the Concurrent State Delta (CSD) system, which has been used to specify and validate protocols using the state delta method [Sunshine82A]. Each state had a precondition, postcondition read list, modify list and time bounds. If the precondition becomes true, then the postcondition will also become true with the time limits. During the time the postcondition takes to become true, the variables in the read list will be referenced and those in the modify list will be changed. In theory, this system will automatically generate proofs, with no user aid. In practice however, the system quickly becomes overtaxed, and user help is needed in helping the system move through the proof in a reasonable way. Since time bounds are included, the system can simultaneously study progress and safety. This system is relatively new, and rated difficult to use, but is

undergoing continued development.

## 5. Evaluation of Methods

Each of the methods discussed above presents certain problems along with the capabilities. Finite state machines, and other associated techniques such as directed graphs, abstract machines and Petri Nets have limitations in handling "real life" protocol specification. At the same time they have provided a number of useful validations, by specifying simplified protocol descriptions which either assume error free transmission, assume no loss or misordering of messages, or disregard error correction. Other common protocol features such as timeouts and sequence numbers on messages difficult or impossible to express. Without simplification, state models quickly become unmanageable and the number of states grows extremely rapidly.

It seems that analysis of protocols with 10,000 to 100,000 states is now possible [West82]. In addition, it is common to break protocols down into layers to simplify the specification and validation. With automated state reduction methods surprisingly complex protocol specifications may be analyzed. Extensions to the state machine methods allow for real protocols to be analyzed in meaningful ways.

Specification validation using programming or special languages is also faulted. The languages allow for specification of looping, sequence and other methods, but the specification becomes obscured with the details of the language used. In addition, the methods of proving program correctness are not completely understood, and certainly not automated completely.

Temporal logic is an attempt to mathematically formalize protocol specification and provide rules for correctness proving. This new technique shows promise and will continue to improve. Proving program assertions is not currently a totally automated process. It requires a great deal in ingenuity on the part of the protocol specifier, both in the specification and in providing the correct assertions to prove the correctness of the protocol.

## Protocol Validation

Simulation is not a solution to proving protocol correctness. However, it has been very useful in developing protocol specifications. It has the ability to provide quantitative information about protocol performance as well. In many cases this is just as important as proving the absolute correctness of a specification.

The development of a protocol is an interactive process. A specification is made and then analyzed for correctness. Problems are discovered and then eliminated in the specification. In addition, protocols are seldom specified completely, but rather in steps. This approach to developing protocols requires a close relationship between the protocol specification method and the validation method.

### 6. Conclusion

The methods of protocol validation have improved as the methods of specification have improved. Major protocols have been validated with a variety of techniques, but always with difficulty. There is yet no systems which can automatically validate numerous large protocols. None of the techniques discussed here are clearly superior to the others.

The methods outlined above are not an exhaustive listing, but rather the methods which seemed to be the most used to do validation. They have been automated in various ways, and have been used with success to automatically validate protocols. There are limitations with each system, both in the methods themselves and in the automation techniques. Some of the techniques, such as finite state machines, seem to be limited by the ability of the computer resources used for the validation. Other methods such as temporal logic show some promise that they may be able to overcome the limitations. Regardless, there remains a great deal of work left to fully understand and realize the promise of easy and automated protocol validation.

K. Sabnani, *A Calculus for Protocol Specification and Validation*, pp 19-34 in **Protocol Specification, Testing and Verification III**, H. Rudin and C.H. West editors, North-Holland Publishing Company, Amsterdam, 1983. Proceedings of the IFIP WG6.1 Third International Workshop on Protocol Specification, Testing and Verification.

[Aggarwal83B] S. Aggarwal, R. Kurshan and K. Sharma, *A Language for the Specification and Analysis of Protocols*, pp 35-50 in **Protocol Specification, Testing and Verification III**, H. Rudin and C.H. West editors, North-Holland Publishing Company, Amsterdam, 1983. Proceedings of the IFIP WG6.1 Third International Workshop on Protocol Specification, Testing and Verification.

[Aggarwal83C] S. Aggarwal and R. Kurshan, *Modeling Elapsed Time in Protocol Specification*, pp 51-62 in **Protocol Specification, Testing and Verification III**, H. Rudin and C.H. West editors, North-Holland Publishing Company, Amsterdam, 1983. Proceedings of the IFIP WG6.1 Third International Workshop on Protocol Specification, Testing and Verification.

[Berthelot82] G. Berthelot and R. Terrant, *Petri Net Theory for Correctness of Protocols*, **IEEE Transactions on Communications**, Com30:12, (Dec 1982), pp 2497-2505. *A good discussion of basic Petri Net techniques.*

[Bochmann80A] G. Bochmann and C. Sunshine *Formal Methods of Communications Protocol Design*, **IEEE Transactions on Communications**, Com28:4 (Apr 1980), pp 624-631. *A general discussion of protocol design, with a fair section on validation techniques.*

[Aggarwal83A] S. Aggarwal, R. Kurshan and

[Bochmann80B] G. Bochmann, *A General*

## Protocol Validation

- Transition Model for Protocols and Communication Services*, **IEEE Transactions on Communications**, Com28:4 (Apr 1980), pp 643-650. *A good paper on a particular use of hybrid techniques for HDLC class protocols.*
- [Danthine80] A. Danthine, *Protocol Representation with Finite State Models*, **IEEE Transactions on Communications**, Com28:4 (Apr 1980), pp 632-642. *A survey of several variations of finite state machines. It is of general interest.*
- [Diaz82] M. Diaz, *Modelling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models*, pp 465-511 in **Protocol Specification, Testing and Verification**, C. Sunshine editor, North-Holland Publishing Company, Amsterdam, 1982. Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification. *A detail explanation of Petri Nets.*
- [Florin83] G. Florin, S. Natkin and J. Attal, *Quantitative Validation for Industrial Ethernet Local Networks*, pp 251-256 in **Protocol Specification, Testing and Verification**, C. Sunshine editor, North-Holland Publishing Company, Amsterdam, 1982. Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification. *A specific discussion of one type of protocol. It is not of general interest.*
- [Giessler82] A. Giessler and J. Hanle, *Simulation of Packet Switched Data Communications Networks*, pp 119-140 in **Computer Networks and Simulation II**, S. Schoemaker editor, North-Holland Publishing Company, Amsterdam, 1982. *A good paper which gives a feeling of what simulation does well.*
- [Kurose82] J. Kurose, *The Specification and Verification of a Connection Establishment Protocol using Temporal Logic*, pp 43-62 in **Protocol Specification, Testing and Verification**, C. Sunshine editor, North-Holland Publishing Company, Amsterdam, 1982. Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification. *This is an interesting article which explains temporal logic in an understandable way.*
- [LeLann79] G. LeLann and H. LeGoeff, *Verification and Evaluation of Communication Protocols*, **Computer Networks**, 2:1 (Feb 1978), pp 50-69. *A detailed presentation of one network. It is of interest since the network was specified using a special language.*
- [Merlin79] P. Merlin, *Specification and Validation of Protocols*, **IEEE Transactions on Communications**, Com27:11 (Nov 1979), pp 1671-1680. *A comprehensive presentation of a number of protocol validation methods. It is a good place to start reading.*
- [Rafiq83] O. Rafiq and J. Ansart, *VADILOC - A Protocol Validator and its Applications*, pp 189-197 in **Protocol Specification, Testing and Verification III**, H. Rudin and C.H. West editors, North-Holland Publishing Company, Amsterdam, 1983. Proceedings of the IFIP WG6.1 Third International Workshop on Protocol Specification, Testing and Verification. *This is a good paper on a particular system.*
- [Razouk80] R. Razouk and G. Estrin, *Modeling and Validation of Communication protocols in SARA: The X.21 Interface*, **IEEE Transactions on Communications**, Com28:12 (Dec 1980), pp 1038-1051. *This paper presents the SARA in detail. It is worth reading.*

## Protocol Validation

- [Remes82] A. Remes, *Simulation techniques in Network Design*, pp 101-118 in **Computer Networks and Simulation II**, S. Schoemaker editor, North-Holland Publishing Company, Amsterdam, 1982. *A general discussion of simulation. It is not necessary for someone who has done a little simulation work.*
- [Sabnani82] K. Sabnani and M. Schwartz, *Verification of a Multidestination Protocol using Temporal Logic*, pp 21-42 in **Protocol Specification, Testing and Verification**, C. Sunshine editor, North-Holland Publishing Company, Amsterdam, 1982. Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification. *An average presentation on temporal logic.*
- [Schneider79] G. Schneider, *Computer Network Protocols: A Hierarchical Viewpoint*, **Computer**, 12:9 (Sep 1979), pp 8-10. *A brief overview of techniques.*
- [Schoemaker82] S. Schoemaker, *A Review of Simulation*, pp 79-100 in **Computer Networks and Simulation II**, S. Schoemaker editor, North-Holland Publishing Company, Amsterdam, 1982. *A general discussion of simulation abilities and methods.*
- [Schultz80] G. Schultz, D. Rose, C.H. West and J. Gray, *Executable Description and Validation of SNA*, **IEEE Transactions on Communications**, Com28:4 (Apr 1980), pp 661-677. *This is an interesting and full view of what it takes to validation a real protocol.*
- [Schwartz82] R. Schwartz and R. Mellier-Smith, *From State Machines to Temporal Logic: Specification Methods for Protocol Standards*, pp 3-20 in **Protocol Specification, Testing and Verification**, C. Sunshine editor, North-Holland Publishing Company, Amsterdam, 1982. Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification. *This is an introduction to temporal logic. It does not quite relate finite state machines to temporal logic.*
- [Schwartz83] R. Schwartz, R. Mellier-Smith and F. Vogt, *Interval Logic: A Higher-Level Temporal Logic for Protocol Specification*, pp 3-18 in **Protocol Specification, Testing and Verification III**, H. Rudin and C.H. West editors, North-Holland Publishing Company, Amsterdam, 1983. Proceedings of the IFIP WG6.1 Third International Workshop on Protocol Specification, Testing and Verification
- [Sidhu83] D. Sidhu, *Protocol Verification via Executable Logic Specifications*, pp 237-248 in **Protocol Specification, Testing and Verification III**, H. Rudin and C.H. West editors, North-Holland Publishing Company, Amsterdam, 1983. Proceedings of the IFIP WG6.1 Third International Workshop on Protocol Specification, Testing and Verification
- [Sunshine79] C. Sunshine, *Formal Techniques for Protocol Specification and Verification*, **Computer**, 12:9 (Sep 1979), pp 20-27. *A complete general overview.*
- [Sunshine82A] C. Sunshine, *Four Automated Verification Systems*, pp 373-379 in **Protocol Specification, Testing and Verification**, C. Sunshine editor, North-Holland Publishing Company, Amsterdam, 1982. Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification. *An interesting discussion of automated system, comparing the ease of use in specifying the alternating bit protocol. It shows there is still work to be done.*
- [Sunshine82B] C. Sunshine, *Formal Modeling of Communications Protocols*, pp 53-

## Protocol Validation

- 75 in **Computer Networks and Simulation II**, S. Schoemaker editor, North-Holland Publishing Company, Amsterdam, 1982. *Another overview of methods. The current state delta discussion is worthwhile.*
- [Sunshine83] C. Sunshine, *Experience with Automated Protocol Verification*, pp 229-236 in **Protocol Specification, Testing and Verification III**, H. Rudin and C.H. West editors, North-Holland Publishing Company, Amsterdam, 1983. Proceedings of the IFIP WG6.1 Third International Workshop on Protocol Specification, Testing and Verification. *A duplicate of the 1982 paper above.*
- [Walden79] D. Walden and A. McKenzie, *The Evaluation of Host to Host Protocol Technology*, **Computer**, 12:9 (Sep 1979), pp 29-38.
- [West78A] C.H. West, *General Techniques for Communications Protocol Validation*, **IBM Journal of Research and Development**, 41:4 (Jul 1978), pp 393-404. *A good presentation of the perturbation method.*
- [West78B] C.H. West. *An Automated Technique for Communications Protocol Validation*, **IEEE Transactions on Communications**, Com26:8 (Aug 1978), pp 1271-1275. *A discussion of Zarifopulo's Duologue method, with additional suggestions.*
- [West82] C.H. West, *Applications and Limitations of Automated Protocol Validation*, pp 361-372 in **Protocol Specification, Testing and Verification**, C. Sunshine editor, North-Holland Publishing Company, Amsterdam, 1982. Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification
- or, *Is the Alternate Bit Protocol Really Correct?*, pp 189-197 in **Protocol Specification, Testing and Verification**, C. Sunshine editor, North-Holland Publishing Company, Amsterdam, 1982. Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification
- [Zarifopulo78] P. Zarifopulo, *Protocol Validation by Duologue Matrix Analysis*, **IEEE Transactions on Communications**, Com26:8 (Aug 1978), pp 1187-1194. *A good presentation of the method for directed graphs.*
- [Zarifopulo80] P. Zarifopulo, C.H. West, H. Rudin, D. Cowan and D. Brand, *Toward Analyzing and Synthesizing Protocols*, **IEEE Transactions on Communications**, Com28:4 (Apr 1980), pp 651-660. *This was a very useful paper on combining finite state machines and performing reachability analysis.*
- [Yemini82] Y. Yemini and J. Kurose, *Towards the Unification of the Functional and Performance Analysis of Protocols*,



# PERSONAL COMPUTER SIG



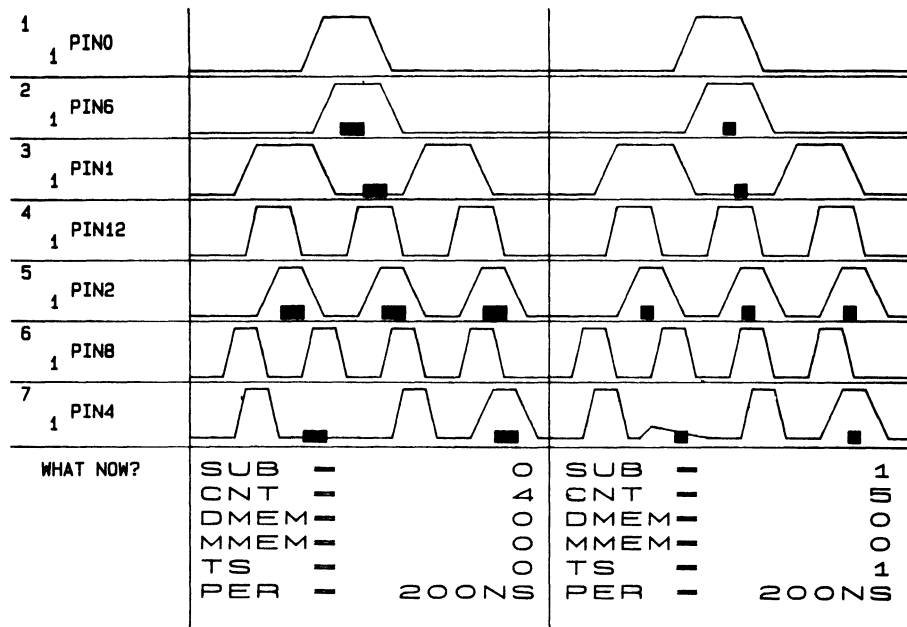


Figure 1. WAVEFORM Display

### Interfaces

The block diagram (Figure 2) shows various interfaces between tasks and peripherals. When WAVEFORM invokes GAM terminal emulation in the PC is suspended, and the RS-232 ports and line which connect the PC to the VAX are used by the two tasks as a communications port for messages which are interchanged between the two tasks. Specifically, WAVEFORM issues SYS\$QIOW calls to SYS\$COMMAND:, and GAM issues calls to the POS (Pro Operating System) services CCTXD and CCRXD.

At all times when it is active GAM controls both the keyboard and screen. All output (including echoing of keystrokes) to the screen is

via calls to the GIDIS interpreter. All input from the keyboard is via the GETKEY service.

The left circle in the lower block represents a disk file named WAVPRM.DAT, which is used to initialize color maps, command syntax, and other operational parameters whenever GAM is first activated. The right circle, WAVHLP.HLG, represents a special graphics HELP file which contains informational text which is graphically displayed whenever the user presses the HELP key.

The names of these files are passed to GAM by the application which invokes GAM. This method of resolving parameter and HELP file names at startup time enables different VAX-resident applications to invoke the single GAM task image.

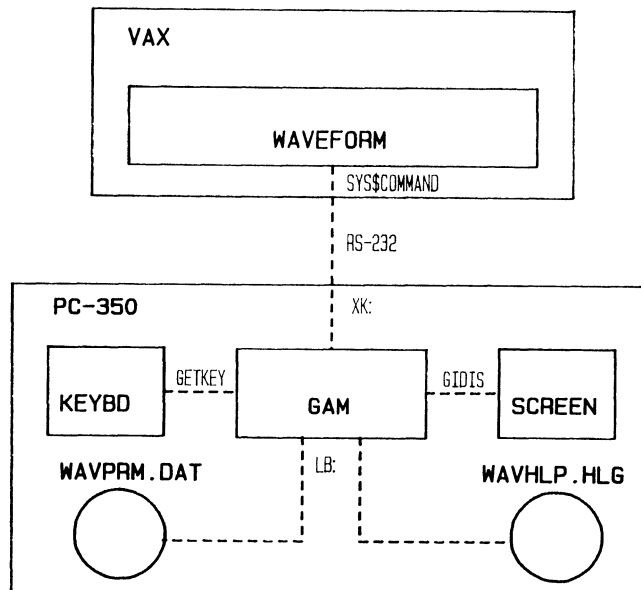


Figure 2. WAVEFORM Interfaces

## The Hierarchical View

The layered model (Figure 3) illustrates the hierarchy of modules used for intertask communication between WAVEFORM and GAM. From the standpoint of the outermost layers (1) WAVEFORM and GAM interchange application-dependent messages which define data to be stored on the PC's hard disk, tell WAVEFORM which vectors the operator wishes to see next, and otherwise synchronize the two tasks.

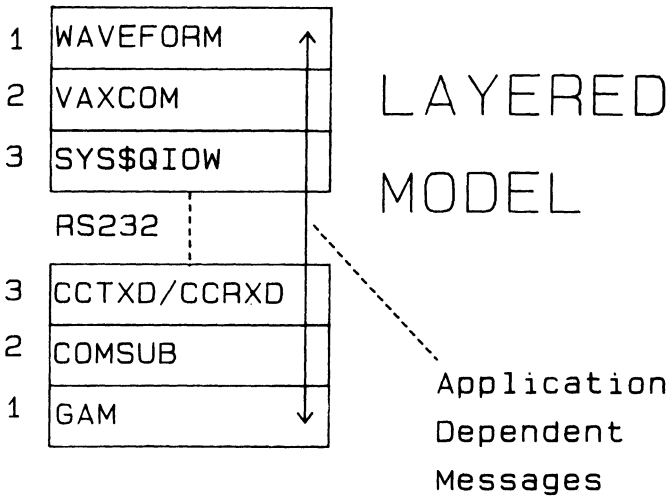


Figure 3.

The two modules in layer 2, VAXCOM and COMSUB, are also functional counterparts. Both of these modules implement high level communication services which in turn call low level communication primitives, such as CCTXD. These layers are totally independent of the applications which call them, and in no way know the "meaning" of the contents of messages which are passed between the WAVEFORM and GAM. In practice, the software which implements these communications services is used by all Sentry applications which conform to the layered model.

In addition to making software in layer 1 independent of the communication primitives in layer 3, VAXCOM and COMSUB perform another more crucial function. The application-dependent messages are regarded by the layer 2 services to be a varying length string of binary, eight bit bytes. Limitations imposed by the communication primitives in layer 3 prohibit use of the low level services for sending certain binary characters, for example XON, which are used by layer 3 services for flow control and other purposes.

Software within VAXCOM and COMSUB encode each message prior to transmission by replacing each prohibited byte with two bytes, both of which have acceptable values. Equations 1 through 4 define the transformation used for each character, C, in the message. Equations 1 through 3 result in replacement of C by a two byte pair of characters, and Equation 4 leaves C unaltered.

- (1)  $C' = 32, 32+C$  if  $C$  in  $[0,32]$
- (2)  $C' = 126, 32+(C-126)$  if  $C$  in  $[126,160]$
- (3)  $C' = 254, 32+(C-254)$  if  $C$  in  $[254,255]$
- (4)  $C' = C$  otherwise

Once the bytes in the message have been encoded the count of encoded bytes N is transformed by Equations 5 and 6 into a two byte pair, N1 and N2.

- (5)  $N1 = (N \text{ div } 64) + 32$
- (6)  $N2 = (N \text{ mod } 64) + 32$

When all transformations are complete the layer 2 software transmits N1 and N2, followed by all of the encoded C' characters. The layer 2 routine which receives the message first reads N1 and N2, computes N, reads the remaining encoded characters in the message, decodes the message, and returns the decoded buffer to its caller in layer 1.

Software within layer 3 consists solely standard operating system software. These services receive encoded (no control characters) buffers from their level 2 callers and perhaps add flow control characters to the data which is being sent across the RS-232 line which connects the processors. Layer 3 software on the other end of the line processes any flow control characters and returns an encoded data buffer to its layer 2 caller.

## Overlay Structure

GAM consists of a root segment and eight overlays. Figure 4 shows the relationship between the root segment routines, the overlays, and the various modules which make up each overlay.

GAM is also the name of the main program module within the root segment. It serves to initialize the task by calling SETUP, and dispatches control between the BUILD overlay and LOCAL.

It is within the BUILD overlay that all communications (with the VAX) take place. BUILD is the module which receives data from WAVEFORM and adds the data to that already on the PC disk.

The GAM mainline calls the module LOCAL whenever it is time for the PC to interact with the user. LOCAL in turn calls PARSE to prompt the user for a response. Once a response is read from the keyboard, LOCAL may return the response to GAM (for transmission to WAVEFORM) or process it locally at the PC by calling TAGA (zoom, pan, . . .), DISPLY (draws graphics), HELP, SELX (special subwindow), or AUX (displays graphics parameters).

The GETCHR module is included in the root segment as a common interface to the POS GETKEY routine, and is called by PARSE, HELP, SELX, and AUX. (Initial attempts to call GETKEY directly from different overlays resulted in errors at task build time.)

Similar in concept to GETCHR, GIDOUT is the module which contains the only calls to the GIDIS interpreter. GIDOUT can run in an unbuffered or

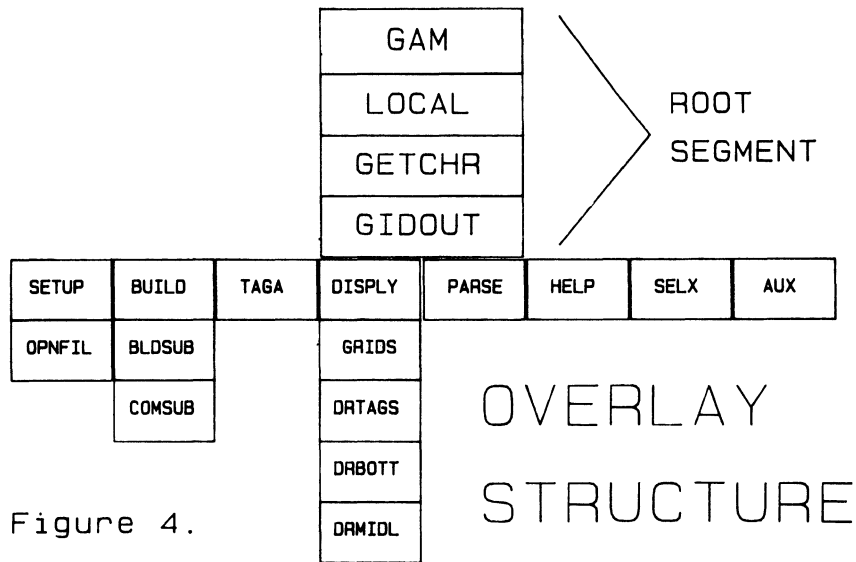


Figure 4.

buffered mode. When unbuffered, a call to GIDOUT results in an immediate display of graphics on the screen. When buffered, GIDOUT calls the GIDIS interpreter only when its buffer is full. GIDOUT is called from SETUP, DISPLY, PARSE, HELP, SELX, and AUX.

#### IMPLEMENTATION ISSUES

In order to produce a functioning WAVEFORM application which conformed to the architecture which was previously discussed, and in order to provide high performance it was necessary to develop a number of essential capabilities. This section details implementation of cold start into terminal emulation, automatically suspending terminal emulation and invoking a PC-resident task from a VAX, use of the hard disk for graphics storage, and how to achieve an orderly return to terminal emulation.

#### Cold Start to Terminal Emulation

A major design objective in the use of the PC as a smart graphics engine was that it be turnkey in the strictest sense. The reason for this

requirement was that most of the time the PC would be used for terminal emulation, usually by persons who knew nothing about the PC-350. Forcing users to learn to use POS menus would only interfere with the real task they were trying to accomplish.

The capability to cold start into terminal emulation is available if PRO DCL or PRO TOOL KIT has been installed, and requires only minor changes to implement.

For WAVEFORM some added protection was desirable: A "safety net" program, (named SENTRY) was designed so that if the user inadvertently pressed EXIT or MAIN SCREEN while in terminal emulation he would be warned of the fact, and would automatically return to terminal emulation if the next keystroke was any key other than "+".

Figure 5 illustrates the automatic entry into terminal emulation, and the manual steps which are required in order to return to the POS main menu. When the PC is cold started a self test is automatically run and control is passed to POS (1).

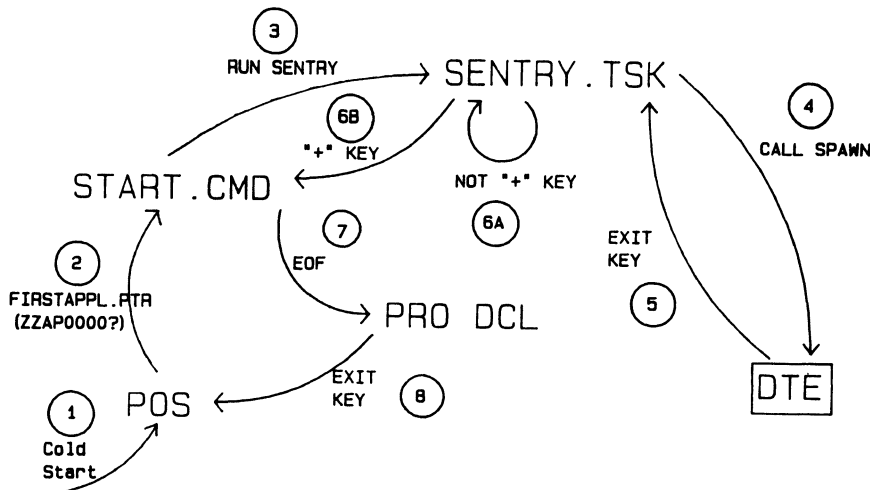


Figure 5. Cold Start and Exit

Ordinarily the main menu would appear at this point. However, if there is a file on hard disk named [ZZSYS]FIRSTAPPL.PTR, then POS first reads the contents of this file looking for the "ZZAP" name of the directory in which an application resides (2). For the WAVEFORM, application software is installed in such a manner that FIRSTAPPL.DIR contains the text ZZAPO0001 -- the application directory for PRO DCL.

The POS main menu is bypassed and [ZZAPO0001]START.COMD becomes active. This is the startup file for PRO DCL. A modified version of this startup file has been configured such that WAVEFORM, SENTRY, and other applications are automatically installed. The final line in START.COMD is the command RUN SENTRY (3).

The SENTRY program takes control of the PC and calls SPAWN (4) to activate and wait on the DTE (Digital Terminal Emulator) program. Once DTE becomes active, terminal emulation is achieved.

Any time the EXIT key is pressed (5) while in terminal emulation, DTE terminates. SENTRY, which has been waiting on DTE, continues execution, calls GIDIS to issue a warning to the user that terminal emulation is about to be exited, and calls GETKEY to read a keystroke.

If the user presses any key but "+" (6A), SENTRY repeats step 4 to reestablish terminal emulation. If "+" is pressed (6B), SENTRY terminates, the DCL startup resumes and encounters an EOF in START.COMD (7), and the PRO DCL prompting character appears on the screen.

At this point the user can press the EXIT key (8) a second time to return to POS, issue DCL commands, or manually RUN SENTRY to reestablish terminal emulation.

### Invoking The PC-resident Task

For a task at the VAX to cause suspension of terminal emulation and activation of a PC-resident task is a simple matter, and is detailed in the documentation for PRO/Communications Rel 2.0. For a task at the VAX to invoke a PC-resident task and then reliably establish intertask communication is entirely a different matter.

WAVEFORM and GAM execute a seven step sequence in order to synchronize themselves and establish intertask communication:

1. WAVEFORM, via a call to an entry point within the VAXCOM module, issues a QIO which captures current SYS\$COMMAND characteristics and returns them in a buffer. This is so that they may be restored when WAVEFORM terminates.

2. WAVEFORM issues another QIO which is equivalent to the VMS command "SET TERMINAL/NOBROADCAST/EIGHTBIT/NOWRAP". BROADCAST and WRAP are suppressed because SYS\$COMMAND is about to become a communications port. Broadcast messages or wrap characters from VMS must be inhibited in order to prevent garbled intertask communications.

3. WAVEFORM issues a third QIO which writes the characters <ESC>\_aGAM!<ESC>\ to SYS\$COMMAND.

("<ESC>" represents the single character, ESCAPE). This escape sequence causes suspension of terminal emulation at the PC and activates GAM.

4. When GAM becomes active it calls CCTXD to write the two character sequence <169><215> to the communications port.

5. WAVEFORM issues QIO calls which read characters one at a time until the sequence <169><215> is encountered. A time out of five seconds is specified for each character, so that if WAVEFORM is invoked from other than a PC-350 with GAM properly installed, a time out will occur and WAVEFORM will perform an orderly recovery. Reading characters until the special sequence is encountered flushes the type-ahead buffer at the VAX and synchronizes WAVEFORM and GAM. That particular character sequence was chosen because the characters are advanced graphics characters which would not be likely to be found in a type-ahead buffer.

6. GAM writes an encoded application-specific message which tells WAVEFORM that it was GAM which sent the two character sequence.

7. WAVEFORM reads (and decodes) the message.

### Storage of Graphics Data on Disk

Because GAM must access more data than will fit in memory or on the screen it is necessary to store graphics data on the pc-350 hard disk. Figure 6 shows the four major windows into which the screen is divided and associates with each window a filespec.

For a particular WAVEFORM run the XYZ portion of the filespec refers to name which is user-specified or defaulted. This name is passed from WAVEFORM down to GAM at initialization time. The last character of the extent name of the filespec is used to differentiate the names of files which store data for the various windows. All four files are opened with the STATUS='UNKNOWN' option to avoid filling up the disk with old versions of graphics files.

The .ART file stores pin group info, one record per pin group, and consists of the ASCII strings which are displayed on the screen. The records are fixed length, unformatted, and are randomly accessed.

Similarly, data within the .ARV file contains the text which is displayed at the bottom of the screen and a real number which defines the period (in time) of each vector. In Figure 1 all vectors had equal, periods of 200 nanoseconds, but often vectors have different periods.

Data within the .ARG file consists of 4096 byte fixed length records which are accessed sequentially when the screen is drawn. The large record size was chosen to minimize time consuming disk accesses. Data within each record is organized logically into subrecords of one or more bytes which define each entity (shape or strobe) to be displayed, and into which row/column intersection the entities should be displayed.

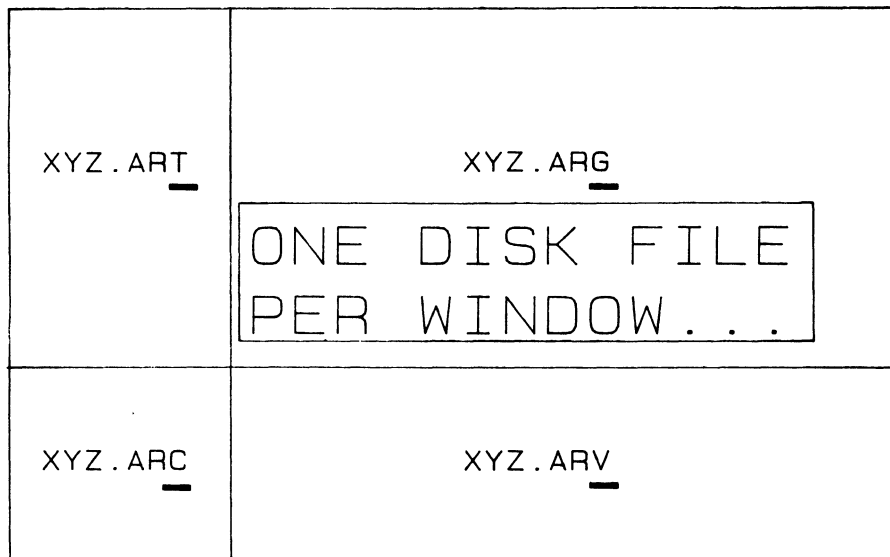


Figure 6. Windows and Disk Files

Byte-oriented storage was chosen to minimize disk and communication overhead. A message which defines a wave shape, for example, is stored into the .ARG file exactly as it is received from the VAX.

Computational overhead in conversion to GIDIS coordinates is avoided by prescaling all possible GIDIS X and Y values and storing them in a static array. When it is time to draw a shape the byte which defines an X or Y coordinate in the .ARG file is used to index the static array of INTEGER\*2 values, which are then passed to the GIDIS interpreter.

The final file, XYZ.ARC, contains copies of GAM's state variables. Included within this file is the number of rows (1-20) on the screen, scale along the X axis, first row on the screen, and first vector. This file is updated whenever GAM terminates.

A boolean which is passed to GAM at initialization time tells GAM whether to restore the state from the previous run by reading in the data from the .ARC file, or whether to reinitialize it's state variables.

This feature provides a powerful capability for the user. It is possible to terminate WAVEFORM and return to terminal emulation, run other utilities at the VAX, and later invoke WAVEFORM with a "/RESUME" qualifier. When /RESUME is specified graphics data from the previous run is retained from disk -- WAVEFORM returns to its previous state in a matter of seconds by not having to regenerate the graphics data.

#### Orderly Return to Terminal Emulation

To return from GAM to terminal emulation is a seven step process:

1. When the EXIT key is pressed GAM closes all of its disk files.

2. GAM clears the screen and reinitializes GIDIS to restore the color map entries to their default state.

3. GAM calls COMSUB to write a message which tells WAVEFORM to exit.

4. GAM terminates via a CALL EXIT. Terminal emulation will resume in a second or less.

5. WAVEFORM reads the exit message from GAM, and waits two seconds to insure that GAM has time to terminate.

6. WAVEFORM closes it's disk files and issues a QIO which restores the terminal characteristics which had been saved prior to invoking GAM.

7. WAVEFORM terminates.

#### GIDIS TOPICS

The initial version of GAM had used calls to CGL (Core Graphics Library) to produce it's screen displays. Full functionality was achieved using CGL, but the speed of this version was found to be marginal -- a dense screen (20 pin groups by 10 vectors) took more than two minutes to display.

Throughput studies of GAM and experimental benchmarks with GIDIS indicated that a throughput increase in the range of 3X to 5X could be achieved if GAM were modified to use the GIDIS interpreter for all of it's graphics. Conversion of GAM to GIDIS was surprisingly easy, due to the presence of various tools which had been developed during the evaluation of GIDIS.

Experience in development of the CGL version of GAM had shown that the time consuming edit/compile/task build/debug cycle for PC resident applications was the pacing item when developing or modifying software on the PC. It was thought that conversion of GAM to GIDIS could be speeded up by first developing software tools which would reduce the number of these cycles.

```

1 1 INIT
275 maps, text, cursor
2 29 SET POSITION
50 x
450 y
255 35 DRAW CHARACTERS
104 h
105 i
-32768 END LIST
0 24 END PICTURE

```

Figure 7. A Simple GID2 Source Program

### The GID2 Graphics Assembler

The first such tool to be implemented was a simple one-pass graphics assembler named GID2. The purpose of GID2 was to provide a mechanism for rapid experimentation with GIDIS, as an alternative to using FORTRAN 77 for this purpose.

The GID2 assembler reads in ASCII text which specifies various GIDIS commands, converts the text into a binary GIDIS buffer, and passes the buffer to the GIDIS interpreter which draws the graphics on the screen. The edit...debug cycle is replaced by a much faster edit/run cycle. The idea here is for the user to be able to try out various sequences of GIDIS commands before implementing them in FORTRAN.

Figure 7 shows a simple GIDIS sequence in GID2 source format. This sequence of GIDIS commands will draw the characters "hi" at coordinates 50, 450 of the screen.

The leftmost column contains operand counts and the operands themselves. For example, the left column of line 3 is an operand count of 2, and the left column of lines 4 and 5 are two operands which are the X and Y coordinates for the "SET POSITION" directive.

The center column is either a GIDIS opcode (29 in the case of SET POSITION) or blank if the left column is an operand. The rightmost column is commentary, and is ignored by GID2. The GID2 assembler detects the end of a GIDIS sequence by detecting and end of file.

The result of using GID2 was that an entire category of bugs -- those in which incorrect sequences of GIDIS directives are issued -- were eliminated prior to coding. Although quantitative data is absent, it is believed that this tool has increased productivity by making it easier to develop GIDIS applications.

### Data Statement Generator

Immediately prior to conversion of GAM-CGL to GAM-GIDIS a simple application (a wafer map

display) was coded and debugged so that more could be learned about the realities of GIDIS-based applications. A major source of coding errors was found to be in manual entry of opcode/operand count numbers and the operands themselves.

A second program, named DATA, was created to automate this process. DATA reads in the same source file which is read by GID2. Rather than writing graphics to the screen, DATA produces DATA statements and array declarations (in FORTRAN 77).

The coding cycle which was used for generation of GIDIS sequences for GAM was a four step process:

1. Write and debug a GID2 program by watching GID2 display the graphics on the screen.
2. When convinced that the GIDIS sequences are correct, run DATA to generate a FORTRAN data statement.
3. Use the editor to rename the array. Add EQUIVALENCE statements for rapid run time access of those array locations which are variable. The constant locations will remain unaltered.
4. Insert the declarations from step 3 into GAM. Add appropriate code to set the variable locations. Use calls to GIDOUT to pass the entire buffer to the GIDIS interpreter.

The results of generating code in this manner were found to be two fold: code tended to be relatively bug free, and was efficient at run time due to the predeclaration of constants and buffering of multiple GIDIS commands.

### Other GIDIS Utilities

Figure 8 shows the relationship between GID2, DATA, and a number of other GIDIS-related utilities which emerged during the conversion of GAM, and have proven useful in development of other applications. Names shown within boxes are the names of devices or disk file formats. The other names are the names of various utilities which read in data in either GID2 or .GID format and output data in another format or to a device.

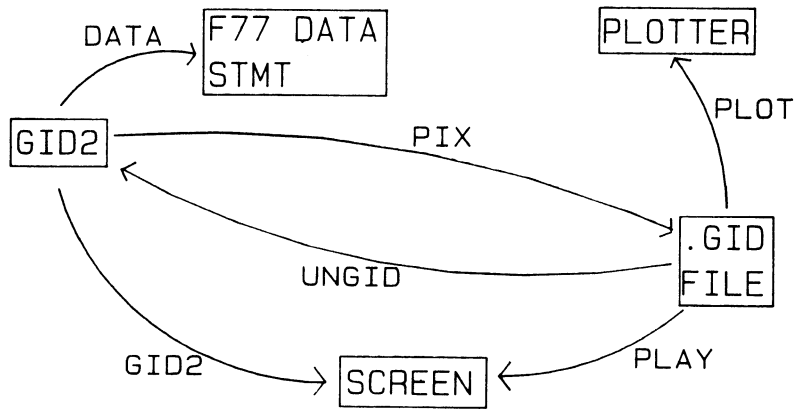


Figure 8. GIDIS Utility Programs

The PLOT and PLAY programs are only a few lines in length, because they merely prompt the user for a filespec and then call Core Graphics to convert the data. With the exception of UNGID, which reverse assembles a .GID file into commented GID2 format, the other programs are only a page or two in length.

#### Color Hard Copy from GIDIS-based Applications

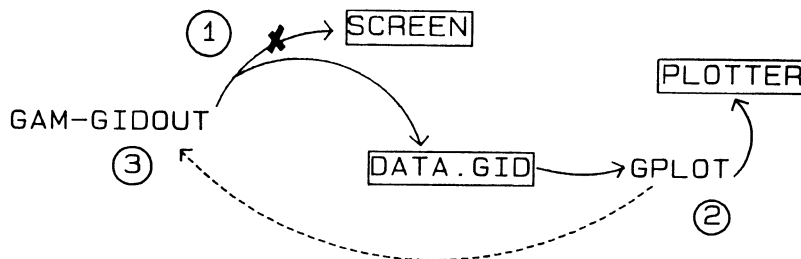
At the same time GAM-CGL was converted to GAM-GIDIS, the capability of producing color hard copy on an LVP-16 pen plotter was added. As shown in Figure 9, generation of a color plot is a simple three step process:

1. In response to a keyboard command, GAM sets a boolean which tells the GIDOUT subroutine to write GIDIS sequences into a file named DATA.GID instead of calling the GIDIS interpreter to write them to the screen. GAM then calls a subroutine which refreshes the screen. Because the boolean is set, all of the GIDIS command sequences are stored into the disk file.

2. When DATA.GID is complete, GAM closes the file and calls SPAWN to invoke a task named GPLOT. GAM becomes inactive until GPLOT terminates. GPLOT (a small FORTRAN program) calls various Core Graphics routines to cause DATA.GID to be "played back" to the plotter.

3. When GPLOT terminates GAM again becomes active. In general, use of the Core Graphics Library is incompatible with direct calls to GIDIS because CGL also calls GIDIS and thus alters various GIDIS state variables. Because these variables are in an unknown (to GAM) state, GAM reinitializes GIDIS and causes the entire screen to be redrawn.

In general, addition of the ability to generate (LVP-16) color hard copy from a GIDIS application can be easy or quite difficult, depending on the structure of the application in question. In the case of GAM, diverting the output from GIDIS to DATA.GID was trivially simple because the GIDIS interpreter was called at only one place within the entire application -- a



1. GAM diverts I/O from screen to disk.
2. GAM spawns GPLOT (CGL) to plot file.
3. When done, GPLOT handshakes with GAM, which reinitializes GIDIS and redraws.

Figure 9. Plotting from GAM-GIDIS

single WTQIO call within the GIDOUT subroutine. Addition of a single IF/THEN/ELSE structure and addition of a (FORTRAN) WRITE statement was all that was required. Had GAM been written with many GIDIS calls scattered about in many different places, addition of color hard copy would have been more difficult.

### RESULTS

Conversion of GAM to use the GIDIS interpreter required major modifications to virtually every one of GAM's subroutines, which totalled roughly 6000 lines of FORTRAN. Conversion and optimization was assigned to a summer coop student who was unfamiliar with the PC-350, and was completed in only six weeks.

When conversion to GIDIS was complete, GAM was again benchmarked and found to execute 4 times faster than GAM-CGL. Times required to draw the

screen ranged from less than 30 seconds (dense screen) down to 2 seconds (sparse screen).

Performance of several applications which utilize the PC-350 as an intelligent graphics engine have been compared to the performance of similar applications which use "dumb" terminals. The speed improvements ranged from 3X to an astonishing 30X -- all in favor of the PC.

### CONCLUSIONS

Development of the WAVEFORM application has proven that it is possible to use the PC-350 as a higher performance alternative to conventional color graphics terminals by implementing the graphics presentation portion of the software locally on the PC. GIDIS is the method of choice for implementing such local graphics presentation software, both in terms of performance and ease of development.





# **SITE MANAGEMENT AND TRAINING SIG**



# DON'T GET BURNED! COMPUTER ROOM FIRE PROTECTION

Terry C. Shannon

THE DEC\* PROFESSIONAL Magazine  
Springhouse, Pennsylvania

## ABSTRACT

*This paper addresses the fire problem in an EDP environment and details some of the measures you can take to minimize the possibility of a fire-related computer room disaster. It presents an overview of the fire protection systems most commonly used in computer room applications, as well as the relative advantages and disadvantages of each type of system.*

Numerous articles have been written about different aspects of computer security and computer room design, but the topic of fire prevention and suppression in data processing installations has been sorely neglected. Often this neglect has had catastrophic consequences, such as the total loss of thousands of personnel records in a fire at a St. Louis military computer installation, or the loss of millions of dollars in revenue which resulted from the New York Telephone Company computer fire in New York City.

If you are a system manager or site manager, a basic knowledge of fire protection systems and techniques can help you make an informed decision should you be called upon to evaluate a proposed system for your own installation. Such knowledge may also save you money — by being aware of the capabilities, limitations and usual applications of the various fire protection systems, you should be able to select the most cost-effective system for your facility. Even if you don't have managerial duties, you still owe it to yourself to be aware of fire protection principles and to know what fire protection systems, if any, are in use at your facility.

Often, little consideration is given to the topic of fire protection beyond meeting the express requirements of insurance underwriters or the local building code. Particularly in a small installation, fire prevention and suppression measures may consist of nothing more than posting "no smoking" signs in the computer room and hanging a twenty dollar fire extinguisher on the wall.

Two reasons help foster this attitude: nobody really expects to have a fire and, perhaps more significantly, fire protection costs money. As a former system manager, I am well aware of budget constraints and cost justification. And as a former fire protection engineer and firefighter, I am equally aware of the costs of inadequate or nonexistent fire protection measures.

The costs of a major fire, both tangible and intangible, can be very substantial. Property damage, equipment loss and the restoration of a building or facility to its pre-fire condition are all costly. In our line of business, the costs of system downtime and file restoration must also be considered. It's impossible to place a dollar value on the intangible costs of injuries, fatalities and the unemployment of workers who have been burned out of a facility, but these so-called "intangible" costs are very real and very significant.

Finally, a high proportion of businesses which suffer large scale fires close down and never reopen. For these reasons, a basic knowledge of fire protection systems and techniques can be very valuable to you.

## THE CHEMISTRY OF FIRE

To get an idea of how to put out a fire, you should be aware of the elements that must be present for fire, or continuous sustained combustion, to take place. In order to sustain itself, a fire requires the presence of four factors or elements in the proper

proportions. The first three of these elements are fuel, oxygen and heat — the components of the "fire triangle" that you probably learned about in a high school science or chemistry class. The fourth element is the combustion chain reaction—a continuous chemical reaction which permits the intermolecular collision of fuel and oxygen molecules and is essential to the continued life of a fire. This chain reaction converts vaporized fuel into "free radicals," or forms of carbon and hydrogen. It is these free radicals that combine with oxygen and do the actual burning in a fire.

## METHODS OF FIRE EXTINGUISHMENT

Traditionally, there have been three ways to put out a fire, one for each leg of the fire triangle. Briefly, these are:

- Removal of Fuel — by pumping, diluting, covering or coating it.
- Removal of Oxygen — by inerting, smothering or blanketing the fire.
- Removal of Heat — by application of water which vaporizes and absorbs heat.

The concept of the combustion chain reaction, the fourth leg of the fire tetrahedron leads to another method of fire extinguishment. This is referred to as "chainbreaking," or inhibiting the combustion chain reaction. This may be accomplished by the introduction of a chemical which inhibits the production of the free radicals discussed above.

Several chemicals that inhibit the combustion chain reaction are in use today as extinguishing agents. The most common of these are dry chemical agents like monoammonium phosphate and purple-K. These agents are generally referred to as dry powder. If you have a small fire extinguisher hanging in your kitchen or elsewhere in your home, chances are it's filled with one of these chemicals.

The other group of chemicals that inhibit the combustion chain reaction are the gaseous halogenated hydrocarbons, more commonly known as the Halons. Both classes of chainbreaking agents are covered in this paper, but for now it's sufficient to note that these agents don't absorb heat, remove fuel or remove oxygen in the course of putting out a fire.

## FIRE PROTECTION EQUIPMENT

Two forms of fire protection equipment are available for protecting a structure or a room within a structure. These categories consist of manually operated fire extinguishers and built-in automatic fire suppression systems. Extinguishers are intended for local application of extinguishant by the occupants of a protected

area, while fire suppression systems are designed to flood an entire protected area with large quantities of an extinguishing agent.

## FIRE EXTINGUISHERS

Not counting the esoteric, special purpose extinguishers that only a firefighter would have knowledge of, there are five types of fire extinguishers that you are likely to come across. Each extinguisher is categorized by the extinguishing agent that it's charged with, namely carbon dioxide, dry powder, foam, Halon and water.

For data processing applications, you should limit your choices to carbon dioxide and Halon extinguishers. Dry powder is an excellent extinguishing agent which acts by breaking the combustion chain reaction. It's discharged as a stream of very finely divided particles to increase its effectiveness. After you use one of these extinguishers, you have a major cleanup problem — the powder residue tends to cling to every surface that it touches. You can imagine what this residue would do to the memory boards, controllers, wiring and circuitry in your CPU cabinet, particularly if they had an opportunity to get good and hot. You don't even want to imagine what the powder would do if it was sucked into one of your disk drives. In summation, dry chemical extinguishers are fine for your automobile, home and office—but keep them out of your computer room.

Foam and water extinguishers are not recommended for data processing or electrical applications for more obvious reasons — potential water damage to equipment, and the very real possibility of electrocution. Both foam and water are excellent conductors of electricity, and it is very likely that the stream of extinguishing agent would contact energized electrical circuits in the event of an equipment fire in a computer room.

Carbon dioxide and Halon extinguishers discharge their contents in gaseous form, leave no residue, do not conduct electricity, and are very effective. Both extinguishers are good candidates for data processing applications.

Carbon dioxide has the advantage of being far less expensive than Halon, but Halon is more than twice as effective on a weight basis as carbon dioxide. It's also far less likely to cause thermal shock to computer chips than CO<sub>2</sub>. In my opinion, Halon is the superior fire extinguishing agent, and I would choose it over CO<sub>2</sub> virtually without exception. However, carbon dioxide is a perfectly acceptable alternative if cost is your primary consideration.

No overview of fire extinguishers would be complete without addressing some of the drawbacks associated with them. First, remember that a fire extinguisher is an effective fire suppression tool only in an occupied area and only after a fire has been detected by personnel in the area. An extinguisher is utterly worthless if there's nobody around to use it.

Secondly, fire extinguishers are portable devices that are intended to put out small fires. By virtue of being portable, extinguishers contain limited quantities of extinguishing agent and cannot be relied upon to combat large or severe fires.

Third, in order to use an extinguisher effectively, the operator must have had some training in the proper use of the device. Most portable extinguishers discharge in a matter of a very few seconds, and if an understandably distraught individual doesn't aim the extinguisher properly, he or she can run out of ammunition in no time at all.

Finally, consider the three foregoing points and don't let the presence of fire extinguishers inspire overconfidence. Remember that extinguishers are not a panacea — they are local application, "first aid" tools. The effectiveness of an extinguisher is contingent upon the size and type of fire, its severity, and the proper behavior of the person operating the extinguisher.

## FIRE PROTECTION SYSTEMS

Unlike local application fire extinguishers designed for "first aid" use on a fire of limited size and intensity by area occupants, a fire protection system is a group of components that act together to:

- detect a fire,
- sound an alarm,
- and discharge an extinguishing agent throughout a protected area to suppress the fire.

The components of a fire protection system include:

- a storage container for the extinguishing agent,
- a discharge nozzle or distribution piping attached to the container,
- an automatic detection and release mechanism with manual override,
- and pressure switches to sound alarms, shut off equipment and close doors and ventilation ducts.

Several types of fire protection systems are available for computer room applications. Each has its relative advantages and disadvantages, all of which should be weighed before choosing a system best suited to your needs. We'll look briefly at four kinds of systems — carbon dioxide, Halon, automatic sprinkler and combination.

### Carbon Dioxide Systems

Carbon dioxide is a very effective extinguishing agent for electrical fires and has had widespread use in computer rooms and other facilities with electronic equipment for this reason. It is relatively inexpensive, readily obtainable and leaves no residue to foul chips, memory boards or peripheral equipment. However, a CO<sub>2</sub> fire protection system does have some shortcomings in data processing applications.

The primary disadvantage involves personnel safety. When a carbon dioxide system discharges, it floods the area that it is designed to protect with CO<sub>2</sub> gas, thereby displacing the normal atmosphere and reducing the oxygen concentration below the 15% required to support combustion.

This 15% threshold also applies to respiration. A CO<sub>2</sub> system is usually designed to provide a 30% to 60% concentration of carbon dioxide gas. This concentration is more than adequate to suffocate people who are unfortunate enough to be stranded in a protected area at the time of system discharge. Because a CO<sub>2</sub> system discharges very rapidly, the air in the protected area is rendered unbreathable within just a few seconds. For this reason, CO<sub>2</sub> systems generally have a delay of at least thirty seconds between sensing a fire and discharging. When the system detects a fire, it will sound an alarm to warn people to evacuate the area prior to actually dumping its supply of agent. A manual override switch is provided to abort system discharge. Some systems are semiautomatic — they employ automatic fire detection and alarm devices but must be manually discharged. Manual discharge eliminates the possibility of personnel being trapped in the protected area when the system floods the area with extinguishant.

While both the automatic time delay and manual system discharge switches are life safety features, they have their disadvantages. If a protected area is not occupied twenty four hours per day, a system that must be manually discharged is useless during nonworking hours. And while the built in time delay between alarm activation and agent discharge is critical to personnel safety, it may allow a fire to intensify. Thirty seconds or so may not seem like a long time, but it can mean a great deal in the development of a fire.

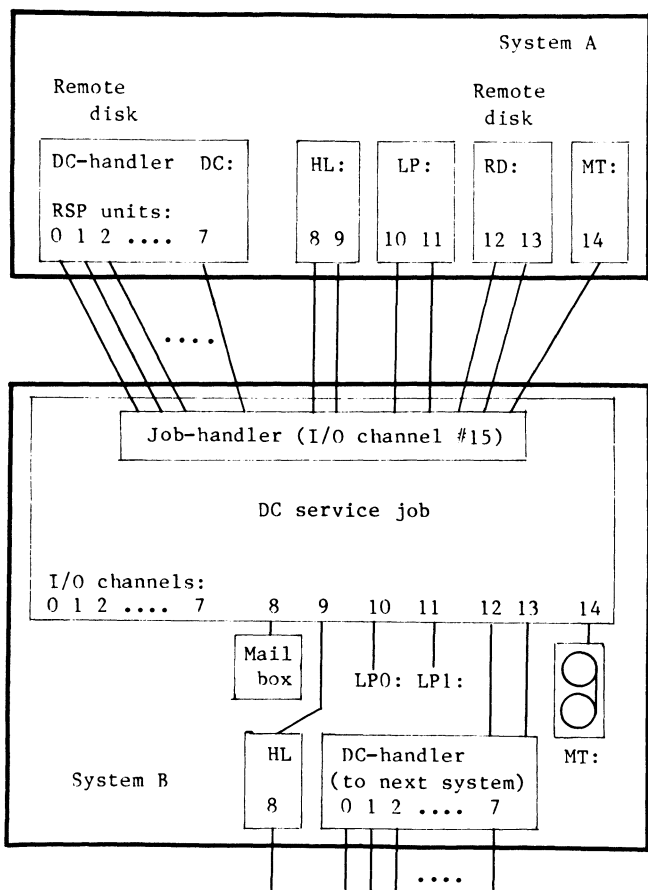


Figure 3. DC Data & I/O channels

systems may be accessed, even when there are intermediate systems.

Currently the following hardware is implemented as the physical data link:

- LSI-11: DRV-11 (DEC)
- WBV-11 (Buchholz / Hammond, Germany)
- Qnector (Westvries Systems, Holland)
- (DMA-interface)
- PDP-11: DR-11C (DEC)
- DR-11K (DEC)
- WB-11 (Buchholz / Hammond, Germany)

However, as all hardware dependent code is in the DC-handler and job-handler, implementation of new hardware only requires that these handlers are adapted to drive that hardware. The general purpose parallel interfaces DRV-11, DR-11C, DR-11K require simple electronic circuitry in order to form a handshake connection, which can be used on interrupt bases. Only one FLIP-FLOP and one OR-gate are the needed components [1]. In the first version also DL(V)-11 type hardware was supported. However, this was dropped as transfer rates are low and DEC supports these data links within the RT-11 package (programs VTCOM and TRANSF, handlers XL, XC).

The size of the DC service job is about 1.5 Kw

including it's 256 word default buffer and the job-handler. The size of the DC handlers is about 700 words and that of the pseudo handlers ca. 50 words. Default the DC job handles read, write, special function (.SPFUN) and boot requests on all channels. However, one channel the message channel, is special purpose and used for message transfer to the user on the other system. These messages and news are also put in a mailbox, a file which is present on each system that runs at least one DC job. The message channel is also used for transfer of date&time from system to system and DC job configuration data (list of remote devices, read/write access).

Normally the job tries to open an I/O channel to all devices in a list, the job device list. This requires that these devices are loaded. One exception on this rule are Special Directory devices such as Magtape. In fact, these devices have no directory and require special operations to open a file. Therefore when an "open file" request (.LOOKUP, .ENTER) is received, this is forwarded to the device itself and not processed by the local USR as for disk devices. Handling Special Directory devices requires additional code in the DC job. This code, including handling "asynchronously directory" operations for Magtape (see RT-11 Software Support Manual), is about 200 words in size. This code may be selected at assembly time by setting a conditional in the DC job source. So two versions may be kept at hand: one that supports all devices (DCJOB.SPD) and one that supports all but Special Directory devices (DCJOB.REL). At the other side of the link a pseudo Magtape handler is available which behaves like a normal Magtape handler but is much smaller in size (ca. 180 words).

#### USING THE DATA LINK

As already pointed out: remote devices are used in the same way as local devices are used! However, pseudo-handlers use the DC-handler and this requires that the DC-handler must be loaded when using a pseudo handler! When it's DC-handler is not loaded a pseudo-handler immediately returns a hard I/O error.

A utility HELLO can be used to send a message to the user at the remote system. It also checks whether there is a difference between remote and local date&time. Further it prints which remote devices are available, the device's characteristics such as size, identifier (helps you to "see through" logical assignments at the DC jobs site), etc. and read/write access to the remote devices. The JBDAT utility is very usefull in the startup command file as it copies remote date&time and sets them locally. When at a site one or more DC jobs have been started, the JSHOW utility should be run. It displays the following data:

```
.JSHOW
JOB Hndlr Nr. I/O Checks Protoc Buffer
--- ----- requests errors errors size
DCJOB0 (0,QJ) 9K 950 0 0 1024 No SPDIR
DCJOB1 (1,DJ) 12K 372 0 0 256 No SPDIR
DCJOB2 (2,DI) not running!
```

SHOW ALL / r&w / change r&w / exit :

Another fact to consider is the possibility of thermal shock to computer chips and circuit boards caused by the extreme low temperature of the carbon dioxide gas. For this reason and the safety factors mentioned above, it is suggested that total flooding carbon dioxide systems be employed only in areas which are not generally occupied by personnel, such as tape libraries and electrical rooms.

If you decide that carbon dioxide provides the optimal solution to your fire protection needs, you should be aware that there are two basic types of carbon dioxide systems, referred to as low pressure and high pressure storage systems. The storage method you will use is almost entirely dependent upon the size of the area you intend to protect. Relatively small areas are candidates for high pressure systems, while large areas are better served by low pressure systems.

A high pressure carbon dioxide storage system consists of high pressure cylinders which are manifolded together with flexible connections to provide the carbon dioxide to the piping and distribution system. These cylinders maintain an average pressure of 850 PSI at normal room temperature, thus keeping the carbon dioxide liquefied until it is discharged. This type of system normally contains two banks of cylinders. The main bank of cylinders provides primary fire protection while the reserve bank ensures that the system will continue to provide protection immediately after the main bank is discharged in response to a fire. This backup protection is useful in situations where expended cylinders cannot be replaced immediately but continuous fire protection is required.

In situations that require a total carbon dioxide capacity in excess of one ton, a low pressure carbon dioxide storage system is often provided. This type of system utilizes an insulated, electrically refrigerated storage tank which maintains the liquefied carbon dioxide at approximately zero degrees Fahrenheit and about 300 PSI. Refrigerating and liquefying the gas allows the storage tank of a low pressure system to be filled to a greater density than equivalent tanks in a high pressure system where the gas is stored at the ambient temperature. In effect, you can store considerably more CO<sub>2</sub> per unit of volume in a low pressure system than in a high pressure system.

However, this density advantage, which translates into a volume advantage, is offset by the fact that the low pressure system requires an uninterrupted power supply to maintain the agent at the optimum temperature. In the event of a prolonged power outage, the CO<sub>2</sub> will gradually warm to the ambient temperature. In doing so, the pressure inside the tank will increase and the gas will be vented to the atmosphere via a safety valve. This, of course, renders the system inoperative.

## Halon Systems

There is a gaseous alternative to both high and low pressure carbon dioxide systems. This is Halon, an acronym for **HA**logenated hydrocarb**ON**. This agent, a relative newcomer to the field of fire suppression, possesses some unique properties and advantages. Unlike conventional extinguishing agents that act by removing heat, fuel or oxygen from the fire equation, Halon breaks the chemical chain reaction of combustion, the fourth leg of the fire tetrahedron. This chainbreaking property of halogenated hydrocarbons was discovered early in the century and was put to use as early as 1907 in the form of carbon tetrachloride fire extinguishers.

Unfortunately, as was learned over a period of years, carbon tetrachloride gives off highly toxic decomposition products when it is applied to a fire and thus subjected to heat. The most notable of these toxic thermal decomposition products is phosgene, which was used extensively as an asphyxiating gas during the first World War. In the 1930's a number of deaths and serious injuries were

attributed to the use of carbon tetrachloride as a fire extinguishing agent, and its use in this capacity was subsequently outlawed.

The modern Halons, notably Halon 1211 and Halon 1301, do not share this toxicity hazard to any great extent. Developed in the past thirty years, these distant cousins to Freon refrigerant are colorless, odorless and tasteless gases which are effective extinguishants even in very low concentrations. In the concentrations commonly used for fire suppression, the Halons pose virtually no threat to the occupants of protected areas.

According to the DuPont Company, a manufacturer of halogenated hydrocarbons, "In its manufactured state, Halon 1301 presents little hazard to individuals exposed to concentrations of 10% or less for up to 10 minutes." A 3% to 4% concentration of Halon is sufficient to extinguish most common fires, and the normal design concentration for a total flooding Halon system is 6% to 7%.

Recently, some questions about the long-term toxic or mutagenic effects of chronic, or repeated and continuous, exposure to the Halons have been posed. This isn't to say that sniffing Halon causes cancer, for there is no evidence available to support such a theory. Over the past thirty years, numerous exhaustive tests on the toxicity of Halon have again and again borne out the contention that it is a safe, "user-friendly" fire extinguishing agent.

The decomposition products of Halon can be toxic in sufficient concentrations. However, any fire of sufficient size and duration to cause a large percentage of the Halon in a protected area to decompose would undoubtedly result in evacuation of the area before the decomposition products presented a toxicity hazard. Because Halon inerts flames so rapidly, it is highly unlikely that a fire in a protected area would be able to decompose a significant amount of the agent before being extinguished by the fire suppression system.

As it extinguishes a fire, Halon produces acid halides, notably hydrogen bromide and hydrogen fluoride gases. In most fire situations, the concentrations of these acid halides will be below 20 parts per million, barely detectable to the nose. Brief exposure to concentrations of this magnitude is not considered to be a major health hazard. In the event of an extremely hot, intense fire, these concentrations may reach levels of as much as 200 to 300 parts per million. Prolonged exposure to these levels of hydrogen fluoride and hydrogen bromide can be harmful, but such exposure would be noxious, irritating and immediately recognizable. The irritating nature of these decomposition products provides a built-in alarm which gives ample warning to occupants of a protected area before toxic concentrations are reached.

Generally speaking, intense heat, smoke, carbon monoxide and other products of combustion from the fire itself present a greater health hazard than do the decomposition products of Halon.

In addition to enjoying widespread use in data processing installations, Halon systems are also used in museums and libraries where prevention of damage to rare books and collections is critical. It is also used for such diverse applications as the protection of jet aircraft engines and racing cars. During the time that I was employed as a fire protection system specialist, I witnessed a number of tests of Halon systems and have actually been in computer rooms during these tests when total flooding systems have discharged. I am convinced that Halon is the safest and most effective extinguishing agent available for total flooding applications.

The major drawback to a Halon fire suppression system is its expense. Halon costs about five dollars a pound, significantly more than carbon dioxide. However, if you are more concerned with occupant safety and minimal interruption of activity than expense, Halon is your best choice for fire protection. The relatively high initial cost of a Halon system is often outweighed by the minimal downtime imposed by the discharge of such a system in response to a fire.

A typical Halon fire suppression system consists of the same

components used in a carbon dioxide total flooding system, with several modifications. Manual discharge of the agent is not normally required, and the time delay between alarm and agent discharge is reduced and in some cases eliminated. Because of the expense of Halon, system actuation is often based on the use of "cross-zoned" detectors. In this scheme, a protected area is divided up into zones, each of which is protected by photoelectric or ionization smoke detectors and ultraviolet flame detectors. When one detector in one zone is activated, an alarm will be given, but the system will not discharge its extinguishing agent. Actual discharge comes only when a second detector in a different zone is activated. In the case of a minor fire, cross-zoning gives personnel time to eliminate the problem and deactivate the system before it floods the protected area with \$5.00 per pound Halon agent. In the event of a wastebasket fire or the false activation of a single smoke detector, this feature can save a great deal of money, aggravation and disruption.

### Automatic Sprinkler Systems

We can now look at another kind of fire protection system — the automatic sprinkler. Automatic sprinkler systems have been in use in one form or another for over 100 years. While the concept of automatic sprinklers is somewhat antiquated, the sprinkler system is still the most popular and frequently specified fire suppression measure. Automatic sprinkler systems are far less expensive on a square foot basis than systems employing gaseous extinguishing agents, and have historically been more than 96% effective in fire suppression.

It may seem heretical to suggest that you even consider employing a water-based fire suppression system in a computer room, but there are a number of circumstances which can dictate its use. First, water is generally a cheap and abundant commodity. Second, it is extremely effective as a cooling and smothering agent. As it flashes to steam, one pound of water absorbs 970 BTUs of heat — making it more effective than virtually any other heat absorbing media. (Granted, metallic mercury is more effective, but it's expensive, toxic and extremely heavy besides). And third, an automatic sprinkler system provides insurance against catastrophic structural damage should a Halon or carbon dioxide fire protection system malfunction.

Two types of sprinkler systems are of interest from the standpoint of computer room fire protection. The first and most common variety employs sprinkler heads with fusible metal links that melt when they reach a given temperature. Once the links melt, the sprinkler heads are free to discharge water. This type of system will continue to discharge water until it is shut off by the fire department or the building owner.

As computers and electronic equipment are allergic to water, a fusible link sprinkler system is not recommended for electronic data processing applications. However, there's another kind of automatic sprinkler that doesn't share this drawback. This second type of sprinkler system utilizes on-off sprinkler heads that are actuated by a thermal valve — a bimetallic element that works like the coil in your heating system thermostat. In an on-off system, the valve on each sprinkler head will open and discharge water when a preset temperature (usually 165 degrees Fahrenheit) is reached. Unlike fusible link sprinklers, these sprinklers will shut themselves down when the temperature in the protected area falls below the preselected actuation temperature.

The main problem associated with sprinkler systems is damage from excess water — not the water which vaporizes into steam and actually extinguishes the fire. The on-off sprinkler system minimizes the amount of water used to suppress a fire and applies the water only where it is needed. Therefore, this kind of system is less likely to cause damage from excessive water than an ordinary sprinkler system. Furthermore, the bimetallic element in an on-

off sprinkler head is designed to actuate in response to a rise in temperature more rapidly than the fusible link in a standard sprinkler head can melt. These features make on-off sprinklers suitable for data processing installations, particularly very large sites that would be prohibitively expensive to protect with total flooding carbon dioxide or Halon systems.

### Combination Systems

The last type of fire protection system that I want to touch on is one that combines the best features of total flooding and automatic sprinkler systems. In very large data processing installations which have raised floors, a combination fire protection system is often employed. This system will protect the subfloor area where the bulk of the electrical wiring is located with Halon while on-off automatic sprinklers are installed overhead. In such an installation, a fire is most likely to start in the underfloor wiring and cables. The Halon system will deal with this fire very efficiently and with virtually no interruption to data processing activities. Should a fire get out of hand and penetrate the floor or originate above the floor, the on-off sprinklers will quench the fire with minimal damage to the facility and its equipment. In a large installation, a combination system offers effective fire protection at far less than expense than an equivalent total flooding Halon system.

### SELECTING A SYSTEM

When you do choose a fire protection system for your installation, you must take several factors into consideration. The most important factor is life safety: your system must first protect people. Data processing equipment is replaceable — human life is not. Choose your system with this thought in mind.

It goes without being said that the cost of a fire protection system is a critical factor in system selection. Your goal should be to select a system that will give you maximum desired protection at minimal expense. If you have a choice between an automatic sprinkler system and a total flooding Halon system and are interested mainly in extinguishing a fire, the automatic sprinkler system should give you the protection you need at less cost than a Halon system. If you are also concerned with potential water damage and service interruption, the additional expense of the Halon system may well be justifiable.

The expense of disaster recovery should also be taken into consideration. Often, this factor is not weighed during the evaluation of a fire protection system. While a fire protection system is an expensive item, recovery from a computer room fire can be far more expensive, particularly when you add up the costs of system downtime, cleanup and repair or rebuilding of the computer center, replacement equipment, and the intangible cost of lost business.

Check with your insurance company to determine what steps you must take to protect your computer facility. Your insurance policy may well require that you adopt a minimum level of protection. If you fail to provide this minimum degree of protection, a hold-harmless clause in your policy may absolve the insurance company from any liability resulting from a fire. If you lease your EDP equipment, read the terms of the lease very carefully. Most lessors require that you protect your leased equipment even though you do not own it. In any event, it is more than likely that the provision of more than the minimum required level of fire protection will reduce your overall insurance premiums.

### FOR MORE INFORMATION

In this paper I have covered some of the fundamentals of com-



puter room fire protection and fire protection systems. Please don't regard this paper as an in-depth study of fire protection technology. Should you be considering the purchase of a fire protection system, some of the best sources of further information are fire protection system vendors, major fire protection equipment manufacturers and independent fire protection engineers and consultants.

The NFPA, or National Fire Protection Association, which promulgates fire and life safety codes, is another excellent source of further information. Check the reference section of your public library for further information on the NFPA.

Your local fire department and building code commission as well as your insurance carrier may also be of assistance. Finally, your state fire marshal's office may be able to provide you with information or answer some of your questions.

It is hoped that you will find some of the information presented in this paper to be useful. My goal in researching and writing this paper is to make you more conscious of the importance of fire protection and suppression in a data processing environment, and provide you with some ideas for increasing the level of fire protection, safety and fire awareness at your installation.

# HOW TO WRITE USER-FRIENDLY DOCUMENTATION

Terry C. Shannon  
THE DEC\* PROFESSIONAL Magazine  
Springhouse, PA 19477

## ABSTRACT

*One of the most significant problems confronted by software developers is the production of high quality product documentation. This paper presents some guidelines and suggestions that you can implement to increase the quality and useability of your reference manuals, users guides, and other forms of product documentation.*

“User-Friendly,” one of the computer catchphrases of the 1980’s, is usually equated with programs and systems, not with the equally important documentation that supports them. In all too many cases, excellent software packages are supplemented with user’s guides that are as nebulous and frustrating as the instruction sheets that accompany products contained in cartons bearing the warning caveat “SOME ASSEMBLY REQUIRED.”

It’s bad enough that we have to deal with instruction sheets like this when we are confronted with something that has to be put together. Its far worse when we, as writers, foist this sort of documentation on hapless computer users. At best, hostile documentation confuses and infuriates computer users. At worst, it turns hesitant novices into outright computerphobes.

This paper is not intended to teach you how to write, but to offer suggestions on what to write and where to locate it in your documentation. Its objective is to assist you in writing documentation that is as “user-friendly” as your software. While the focus of this discussion is on software documentation, most of the material presented here is equally applicable to other aspects of technical writing for the computer industry.

## THE IMPORTANCE OF GOOD DOCUMENTATION

The satisfaction and the success of your software customers depends on many factors, not the least of which is the quality and consistency of your documentation. Prospective clients may visit existing sites as they evaluate your products. During such visits, one of the things that they will usually see is your documentation. Alternatively, a prospective customer may request samples of your system documentation to use in a case study or product evaluation. In this case, your documentation is the vehicle that will provide a potential buyer with that all-important first impression of your product. In any event, your documentation projects a corporate attitude to your readers — it mirrors your company and its concern for the people who use its products.

Take a moment and contrast your software documentation with an advertisement for the same software package. In one regard, they are synonymous — your document can be considered to be an advertisement for your firm and the software that it markets. Compared to an advertisement, a user manual has a long half-life: the manual will be around long after the conclusion of a software advertising campaign. A lucid, well written manual provides readers with a favorable impression of your firm and its product each time the manual is read. Conversely, a poor quality manual projects an unfavorable image of your company whenever it is used.

For these reasons, it is important to all of us who write documentation to contribute to the success of our users, employers, and ourselves by striving to write the best documentation possible. If you do not actually write documentation yourself, you can read and critique documents written by others, and make suggestions to these writers to help improve the quality and readability of their work.

## WHAT MAKES DOCUMENTATION “GOOD”

An analogy can be made between a college textbook and a documentation manual. Both are intended to present information and facilitate learning, and both can be of good or poor quality. The reason for this is not so much the nature of the information being presented in the textbook or manual, but the style or method in which the information is presented.

As a technical writer or documentation specialist, your primary objective should be to convey information to your audience. To achieve this goal, you must first get and maintain the interest and attention of your readers. By doing so, you create an atmosphere which motivates your readers and facilitates learning.

Many factors contribute to the quality of documentation, and there are many criteria by which documentation can be judged. If a document is readable, understandable and usable, it can be considered to be adequate. If the manual conveys all the information a reader needs to use a particular software package, and conveys the information so clearly, coherently and completely that the reader doesn’t have to turn to another manual or another person to resolve unanswered questions, then the manual is an example of “good” documentation.

## CHARACTERISTICS OF GOOD DOCUMENTATION

Your main goal should be to write “good” documentation, using the criteria of clarity, coherence and completeness. These criteria are summarized as follows:

- CLARITY — Convey essential information in such a manner that it will be understood, not misinterpreted.
- COHERENCE — Write so there is a natural progression between each section of your document. Confine your discussion of each major topic to its appropriate section within the document.
- COMPLETENESS — Provide all the information a user might need to use your system or program. Users want to refer to a single source — not a shelf full of manuals — to get your system to work.

## THE DUAL PURPOSE STRATEGY

Much documentation from software houses and independent software writers is in the form of a dual purpose manual, with a title such as “Reference Manual and User’s Guide”. A title like this implies that you should have dual objectives in mind as you organize and write. A dual purpose manual is often written to save documentation development and publication expenses (why write two manuals when one will suffice?) However, writing a dual purpose manual is not an easy undertaking. You must provide your reading audience

with a document that is both tutorial and informational in nature, and you must keep these two concepts separate from each other to prevent confusion.

A reference manual is the place where the “technical user”, such as a programmer or systems analyst, looks for answers to questions about a system. Many of these answers may need detailed information, which your manual must provide. A reference manual also demands a logical organization so that information is easy to locate, supported by a good table of contents and index. The user of a reference manual wants detailed answers or needs to know where to find these answers.

A user’s guide, on the other hand, is where the “average user” turns for instructions and information on how to make the system work to accomplish his or her routine processing needs. Often, this person does not want much detail. He or she may want a discussion of the work flow and some suggestions for the preparation and processing of data. Do not intimidate your average user with highly technical and usually irrelevant information.

Both types of users need an overview of the system. They need to have in mind a conceptual understanding of the structure of the system. From this point forward, their needs are quite different. This is why the structure of your manual is so critical to its success.

### Writing for the Average User

Remember that your average user is not likely to have much experience with your particular system, or computer systems in general. Your documentation should ease this reader into your system, explaining the system in layman’s terminology wherever possible. Introduce your reader to your system through the use of easily read, small sections of text, each of which covers one main point. Avoid the use of complex acronyms where possible.

Obviously this isn’t possible all the time—acronyms are an inescapable fact of life in the computer industry. Whenever you do make use of an acronym (such as VMS), spell out the definition of the acronym the first time it appears in your text so that your reader will know what you are talking about. To maintain interest, you may wish to provide a brief explanation of the origin of the acronym. This strategy is equally applicable to jargon. For instance, if you use the term “debug” in a manual, you might attribute the term to Grace Hopper, who coined it after she extricated a dead moth from a relay in Harvard’s Mark I computer.

Be sure to include a glossary for complete explanations of acronyms and data processing terms which your reader probably is not familiar with. A technique that I find useful is to bold or underline those words or phrases in the body of a document which appear in the glossary. By doing this, I make the reader aware that a complete explanation of a term may be found in the glossary.

Lay the manual out in a logical sequence, so it makes sense to the reader. Don’t deal solely with program dialogue and options — provide sections where there are discussions of how the system solves problems and manages data. These discussions need not be complex or highly technical in nature, but most average people are at least mildly interested in how the system or program that they are using works. Brief discussions of this kind can be used to help hold your reader’s attention. Remember the college textbook analogy — a book which monotonously presents a series of facts with little or no supporting information is far less interesting or readable than a book which manages to present the same facts with pleasant accompanying dialogue and examples.

### Writing for the Technical User

Most of the information sought by a technical user — the individual who is more interested in a reference manual than a user’s guide — should be compartmentalized, either in separate chapters or within appendices. Technical information should be thoroughly

indexed, as a technical user is more likely to refer to the index than to the table of contents. Again, you should strive to make the information you are presenting interesting as well as informative. Systems programmers have attention spans, too.

### QUESTIONS TO ASK YOURSELF

In deciding to write a dual purpose manual, you have chosen a more difficult task than that of producing a single purpose manual, where the objective of the manual is singular and clearly defined. In order to produce a clear, useful document, you should keep the following questions in mind throughout the writing and editing of the document.

- Who am I writing this particular section for?
- How am I specifically helping my reader here?
- Will it make sense to my reader?
- Is this material in the right place in the manual?

Answering these questions continuously, always analyzing what you are writing and to whom you are writing it, will assist you in producing a more useful and informative document. Remember that you are addressing a number of different individuals: the data entry clerk, the computer operator, administrative staff, and technical personnel. Each category of reader has different needs, and you should tailor your writing to accommodate your overall reading audience. This is best done by writing different sections of your document for different kinds of readers. The strategy that I employ to effectively address a broad spectrum of users is covered in the following text.

### THE STRUCTURE AND SECTIONS OF THE MANUAL

The structure of the manual is important because your readers are usually introduced to your system through the manual. They must find the manual easy to read, and pleasing in layout and appearance. The manual must contain useful, easily accessible information which is presented in a logical manner.

You must provide separate sections of your manual which address the needs of both elements of your intended audience: the “average user” who wishes to read a user’s guide, and the “technical user” who is interested in the more detailed reference manual. A good philosophy is to attend to the needs of a user’s guide reader within the main body of the manual, and to provide reference information within an appendix or appendices. Through the use of this technique, you can address the needs of both the average and technical reader independently. In essence, you are separating the wheat from the chaff for your readers. Neither type of reader is forced to slog through material that is of no consequence to him.

Your manual should describe the essential components of any data processing program or system: input, processing, and output. These essentials can be treated in a number of ways, depending on the system. One good method is to organize the manual by subject or major activity, and cover the processing of each in turn. Thus, a manual which covers a number of programs may be arranged so that a chapter is devoted to the input, processing and output of each program. Conversely, a manual dedicated to one program or system can be structured so that input, processing and output are covered in separate chapters. These three main processes are augmented by the following manual sections — a preface, a table of contents, an introduction, an appendix (or appendices) and an index. Optionally, a bibliography and a reader’s comment sheet may also be provided.

### The Preface

The preface should provide a very brief introduction to your man-

| No | Device name    | iden. | size  | characteristic           |
|----|----------------|-------|-------|--------------------------|
| 0  | DLO:SYSII .DSK | FILE  | 4800. | * FILE *                 |
| 1  | DLO:SOURCE.DSK | FILE  | 4800. | * FILE *                 |
| 2  | DLO:VER .DSK   | FILE  | 4800. | * FILE *                 |
| 3  | DL1:ERPROG.DSK | FILE  | 4800. | * FILE *                 |
| 4  | DL1:DATBAS.DSK | FILE  | 4800. | * FILE *                 |
| 5  | DL1:ERP .DSK   | ..    | ..    | Not in system            |
| 6  | DLO:DIPOL .DSK | FILE  | 1200. | * FILE *                 |
| 7  | DL1:ER .DSK    | FILE  | 1200. | * FILE *                 |
| 8  | SY :JBINFO.DAT | FILE  | 16.   | * FILE *                 |
| 9  | HL :           | .     | 377   | 0. SPFUN                 |
| 10 | SP :           | .     | LP    | 0. WONLY                 |
| 11 | DL0:           | .     | DL    | 20450. FILST SPFUN VARSZ |
| 12 | DL1:           | .     | VM    | 384. FILST               |
| 13 | DM :           | .     | DM    | 53724. FILST SPFUN VARSZ |
| 14 | MT :           | .     | MT    | 0. SPECL HNDLR SPFUN     |
| 15 | JOB:           | .     | 300   | 0. SPFUN                 |

SPECIAL FEATURES

One special feature is using a remote disk as system disk. When a remote disk is made bootable for the DC-handler (e.g. DC:) and contains the necessary systems components such as monitor file and utilities, it can be booted with the standard command:

.BOOT DC:

The disk may have been made bootable before with the command:

COPY/BOOT DC:RT11FB DC:

However, it could also have been made bootable at the disk's site with the command (assume that the disk's name is DK:):

.COPY/BOOT:DC DK:RT11FB DK:

Press RETURN to continue

| Central device      | DCJOB0 | DCJOB1 |
|---------------------|--------|--------|
| 0 LD0: sys 11/23    | R W    | R -    |
| 1 LD1:              | R -    | R W    |
| 2 LD2:              | R -    | R -    |
| 3 LD3:              | R -    | R -    |
| 4 LD4: Datbas       | R -    | R W    |
| 5 LD5:              | N      | N      |
| 6 LD6:              | R -    | R -    |
| 7 LD7:              | R -    | R -    |
| 8 JBINFO Mailbox    | R W    | R W <= |
| 9 HEL               | R W    | R W <= |
| 10 SP: Spooler      | - W    | - W <= |
| 11 RDO: Remote disk | R -    | R -    |
| 12 RD1:             | R -    | R -    |
| 13 RD2:             | R -    | R -    |
| 14 MT: Magtape      | N      | R W    |
| 15 JOB handler      | R W    | R W <= |

The ability to use a remote system disk is a very powerful feature because it allows using memory-only (or better: no disk!) systems! In order to serve memory-only systems, the DC job also acknowledges a boot command. When such a command is received, it transmits a block of data (256 words, without protocol header and tail!) : the BOOT program. Sending the boot command to the DC job can be done by a small program which has been put in (P)ROM. However, it could also be typed in (toggle-in boot) using CPU-ODT available on most machines. When the BOOT-program is received and activated, it asks a password. When the correct password has been entered, it asks the unit number of the bootable disk, fetches the bootstrap from that disk and activates it. RT-11 will then come up. When using a disk data cache also multiple systems can use the same (remote) system disk [3,4].

Note: R=read, W=write, N=no device

Show all / r&w / CHANGE R&W / exit :

Change ?

Give JOB nr. :1:  
RC, RS, WC, WS :WS:  
Device no.(0-14):13:

Change ?

Give JOB nr. : :

Show all / R&W / change r&w / exit :EXIT

With this utility the read and/or write access to a device for a certain job can be changed. The number to the left in the device list is the DC channel number (also RSP unit number). Note that although on RSP unit no. 12 the device DL1: is specified, this device in fact is the VM: disk ! This is because the logical assignment ASSIGN VM DL1 has been made before the DC jobs where started!

A job is simply stopped by aborting it:

.ABORT DCJOB1  
.UNLOAD DCJOB1

Another feature is parallel processing. By this is meant that data are transferred from the memory of one system directly to the memory of another system. The data can then be processed in parallel by both systems. This feature is realized by using a special purpose, internal queuing, handler (refer to: "Internal queuing handlers" in Chpt. 7, RT-11 Software Support Manual). This special purpose "parallel" handler can accept a next write request before a previous read request is finished [1]. Therefore this handler can transmit data from a buffer within one job to the buffer of another's job.

Display in detail of the activity of a DC job can be realized using a device I/O logging&display package [3]. The packets transmitted and received can in this way be monitored by selecting the job-handler as the device under investigation. The I/O display is activated by loading a special handler and running another system job (SHOWIO or LOGG). Also a test version of the DC job may be generated. This job prints a "C" for each command-packet received, a "R" for each data-packet received, a "S" for each data-packet transmitted and an "E" for each END-packet transmitted. After an "E" the next characters are printed on a new line.

Not a feature but more a problem is that of "job blocking". As the DC job runs as Foreground or System job, it runs at a higher priority than the Background (BG) job. This may be one of the causes

ual, its purpose and intended audience. The preface should tell your readers why they should read the manual and how they will benefit from reading it. The preface need not be very lengthy — in many cases, one page of text will provide an adequate introduction. A cursory examination of the prefaces contained in most Digital Equipment Corporation manuals proves this point. If the manual you are writing is a revision or update of an existing volume, the preface is the logical place to list the differences between the old manual and the system that it documents and the revised version.

### **The Table of Contents**

A well organized and complete table of contents serves as a road-map to your document, and is often consulted by users looking for information before they turn to the index. The table of contents should be at least partially completed before you begin writing the main body of the document. This way, you can use the table of contents as a rough outline for your manual.

Don't overdo the table of contents: for example, if you are writing a chapter which discusses five programs, list the five programs under the chapter number, but save references to the ten subfunctions of each program for the index.

### **The Introduction**

In the introduction, briefly describe the purpose of the manual and the purpose and capabilities of the system. Make it clear that the technical aspects of the system, such as file design, program internals, and a complete description of the system oriented towards the technical user may be found in the appendix.

Include a brief system overview in the introduction, but phrase it in such a way that you will not intimidate an average user.

The introduction is also the logical place to list the conventions that are used in the manual (such as "depressing the RETURN key will be indicated by the symbol "<RET>"). A brief definition of responsibilities may also be included in this section. Here, explain what duties are to be performed by data entry personnel, technical staff, and computer center personnel.

### **Input**

The discussion of the first element of the data processing procedure should include a description of input media and procedures, information about the data elements derived from this media, and a descriptive listing of error messages or routines that may be encountered in this phase.

Bear in mind that this section will most likely be read by data entry personnel. Provide them with illustrations of source documents and screen or menu formats that they will encounter as they use your system.

### **Processing**

While covering the main portion of your system, include a discussion of system commands, step by step operating instructions, examples of system dialogue or menus, "Help" messages, and error recovery procedures. This portion of your manual should be addressed to data entry or computer operations personnel, and should contain flowcharts or diagrams which illustrate processing procedures in the order that they occur.

Pay particular attention to error recovery procedures. If you have ever been a computer operator, the need for these procedures should be obvious to you. If you don't have an operations background, put yourself in the shoes of the new third shift operator who inadvertently deletes a file that is needed for further processing. He or she needs to know what steps to take to rectify the mistake right away. If your recovery instructions are

not laid out in a clear, logical step by step manner, or if you fail to provide these instructions, there is little doubt that you will be disturbed by telephone calls in the middle of the night.

### **Output**

Here, explain the nature of the output — magnetic media, printed forms, graphics, microfiche, or whatever. Supply illustrations of what the output should look like, where appropriate. Provide an explanation of balancing procedures, control totals, and other relevant information. Attention should be devoted to edit and error lists, and error recovery steps that the user may take, if needed. Direct this information to data control and administrative personnel — the end users of your system.

### **The Appendix or Appendices**

Once you have provided detailed, step by step instructions that explain how to use your system, you need to provide the information that goes beyond the essential operating instructions. Using the structure that I suggest, the appendix is the logical place to locate reference and "nice to know" information. This section should be addressed to the technical user who wants detailed information on the functioning of the system — the kind of information to exclude from the main body of your document. It is of little or no consequence to the casual user of the system, and its presence in the "how to do it" section of the manual serves only to confuse and alienate the average user.

Simple flowcharts and examples in the introduction and main body of your document should provide sufficient technical information for the casual or average user. Depending on the size and complexity of your system, several appendices may be needed. One could be devoted to file layout and design, another to programming internals, one to initial system setup, and possibly another one devoted to detailed technical error recovery procedures which would be beyond the scope of the average user.

In some instances, it may be advisable to include a brief appendix for the average user. For example, your system might have nothing to do with VAX/VMS except run on a VAX computer. In this case, it might be wise to include a short appendix on "Using VAX/VMS" so the users of your system will at least know how to log in to the computer and run the programs that your system consists of. Take nothing for granted, particularly with new users.

### **The Glossary**

Because of the number of acronyms and terms common to the data processing business, a glossary containing nontechnical explanations of acronyms and system specific terms is essential. Deciding what to include in a glossary isn't easy, but should be based on common sense: while the average user probably has no need to know what UAF stands for, he or she might need to know what a UIC is, or what DCL stands for. Common terms like "field", "file" and "record" should be explained in this section. These explanations should be thorough without going into further detail than is necessary to help your readers understand your manual.

Although a glossary should be arranged in alphabetical order, there is no need to include multiple definitions for each entry, as a dictionary might. Keep it simple.

### **The Index**

If you are using a text formatter like Runoff, or a sophisticated word processing program, you can insert index entries into the body of the text as you are writing it. Although this will not provide a complete index, it will simplify the final step of reviewing the document and highlighting important topics which should be indexed.

Remember that the index must not merely be an alphabetized version of the table of contents. Not only should it list program functions and subfunctions in greater detail and more thoroughly than does the table of contents, the index should also make direct references to topics that an average user will consider to be important.

Cross-referencing and subindexing entries may also prove to be helpful to your readers. Again, this extra step is most easily accomplished with a text formatter or word processor.

After highlighting and indexing what you feel is important, enlist the aid of a user and find out what he or she thinks should be included in the index. For example, you could write an excellent manual about a payroll system and index it to the hilt — but unless you happened to be a payroll clerk, the significance of an index entry for the Advanced Earned Income Credit might be lost on you. Make use of your users — the results will be beneficial to you as well as them.

### **The Bibliography**

This section is optional, and should only be necessary if you refer to documents or manuals not closely related to your specific system. A bibliographic reference to your master accounting system manual is not needed at the end of the manual which describes the cash disbursements subsystem of the accounting package. However, if you make repeated or continual reference to a magazine, textbook, or third party documentation, such as Digital VAX/11 manuals in your document, you may wish to include these publications in a bibliographic “For More Information” section.

### **The Reader’s Comment Sheet**

Finally, you may wish to include a reader’s comment sheet at the end of your document. By doing this, you can obtain positive and negative feedback on your writing, and use this feedback to improve future editions of your manual, and your overall writing style as well. I strongly advise the use of a reader’s comment sheet, as it represents the only vehicle available to you for user feedback beyond that which is generated during an in house review of your manual.

## **GENERAL ADVICE**

There are probably as many writing styles and techniques as there are technical writers, and what works well for one writer may not necessarily be effective for another. However, there are a number of general techniques which can be implemented by any writer to increase the quality and useability of documentation.

### **First Person Tense**

Where possible, try to write to your reader. Don’t phrase everything in the third person. I prefer to read a document that refers to “you” instead of the anonymous “the user” (a phrase with sinister connotations).

### **Remember Basic Grammar**

Keep each sentence in the same tense. I’ve seen single sentences in documents that include three verb tenses, like “is asking,” “will occur” and “did prompt the user for.” This kind of structure does not make for easy reading or coherent flow of thought. Nor is it a good reflection on the quality of your documentation.

### **Be Consistent**

Maintain consistency throughout your document, in capitaliza-

tion, abbreviation, and in user instructions. Establish a style or format at the beginning of your document, and adhere to it. This will avoid confusion and misinterpretation on the part of your readers.

Consistency is also important in manuals that document different but related systems. It’s far easier to find the answers to questions in a multiple volume document set if each volume is structured in a similar manner. To maintain consistency without going to extremes, you can identify different systems in a software package with color-coded binders or paper.

### **Avoid Forward-Referencing**

Forward-referencing is a common trait shared by many experienced writers who are attempting to convey information to a novice audience. Forward-referencing takes two basic forms: footnotes and examples preceding explanations. An example of footnoting is a five-step checklist that concludes with a “by the way” statement like “before proceeding with Step One above, you must . . .” Few people read instructions thoroughly before carrying them out, so you must ensure that your readers won’t be confronted with any unpleasant surprises at the conclusion of a step-by-step procedure.

Preceding explanations with examples is more common than footnoting and more difficult to eliminate. If you write a five chapter manual that contains examples of program commands or a system command language in the first four chapters, but doesn’t explain these commands until the fifth chapter, you are guilty of forward-referencing. By explaining new topics as they are encountered, you will prevent this form of confusion.

### **Avoid Repetition**

Avoid the use of monotonous, repetitious passages of text. If you must explain the dialogue of fifteen program which are all run in the same manner, give a detailed explanation of the standard way to invoke a program when discussing the first. For the remaining fourteen programs, state that “PROGRAMNAME is invoked in the standard manner.”

### **White Space**

Few things are as threatening to a new user as confronting a manual or document that consists of page after page of dense, closely packed text. Paper costs money, but you shouldn’t present your reader with a document that reads like an unabridged dictionary. Break up your text into chunks and set each new concept or key point apart by surrounding it with liberal quantities of blank paper, or “white space.”

### **Provide Illustrations And Examples**

Make use of illustrations and examples, and keep them as simple as possible. Remember, people don’t like to read documentation. The less they have to read and digest to make a program or system work, the happier they are.

### **Keep It Simple**

As previously discussed, there will be times when you must include acronyms and jargon in your text. To the average user, “four gigabytes of virtual address space” sounds like a quote from a Star Trek script. Not only is the phrase highly technical, it is out of context. It has no bearing on the way your system works, and serves only to confuse your readers.

Remember that many computer-related terms (“swapping”, for example) have very different meanings to average users than they

do to computer programmers or system managers. These terms are generally of no importance to the reader of a user's guide. If a term or phrase has no impact on your system or users, it doesn't belong in the document.

### **Use Analogies**

There will be times when you must use and define complex terms. If you had to explain memory management on a VAX computer to a novice audience, you would definitely have to include a description of paging. To complicate matters, you would have to phrase your explanation in terms that a novice could understand and relate to.

When you are faced with a situation like this, make use of analogies, keeping them as simple as possible. Example 1 (at the conclusion of this paper) shows the analogy I used to explain the concept of paging in a book I wrote for first time users of VAX computer systems. Through the use of analogies and comparison, you can often make complicated concepts understandable to your audience.

### **Adopt A Businesslike Style**

Do not be condescending or patronizing to your audience. There is no place in a well written manual for laudatory or congratulatory phrases, such as "congratulations on your purchase of the Duz-It-All word processing package." Avoid the use of cliches, slang and distracting language. Your readers don't want a pat on the back or a manual full of buzzwords—they want to learn how to use the system.

### **Be Considerate**

Finally, remember that you are writing to people in lieu of talking to them in person. Think of writing as your half of a two way conversation. Try to anticipate the other person's questions, and answer them clearly and completely. Although you likely never will meet your readers, they will appreciate your efforts on their behalf.

### *EXAMPLE 1*

*(From Chapter 1, "INTRODUCTION TO VAX/VMS," by Terry C. Shannon. Copyright © 1985, Terry C. Shannon and Professional Press.)*

### **VIRTUAL MEMORY**

Virtual memory is the ability of a computer to address and use more memory than its central processing unit physically contains. A one megabyte VAX processor can address not just the one million locations contained in its physical memory, but, due to its 32 bit address length, a total of over four billion storage locations. At this point, a question might come to mind: How does a computer make use of over four billion addresses when it can store only one million of them at a time?

The answer is relatively simple when you look at it in simple terms. The VAX does essentially the same thing that you do every time you read a book. Regardless of the length of a book, you read it one page at a time. The computer does the same thing—it processes a program one page at a time.

To the VAX, a program is similar to a book composed of many pages which is stored on its disk drive. When you issue the command to run a program, the computer first scans the entire program, breaks it up into pages, and creates an index to reference the location of each page. Each page is a 512 byte, or word, segment of a program which is read into the computer from the disk and is then processed or acted upon. When the computer is finished with a segment, it scans its index to locate the next page it needs. It then generates a "page fault" by returning the completed page to the disk and then reading in the next segment needed to continue processing the program.

In essence, the page fault executed by the VAX is equivalent to the human action of turning the page of a book. Because of its ability to process a program one page at a time, the VAX can process a program of almost infinite length within the confines of its physical memory.

**RT-11 SIG**





Harry Haenen  
 Dept. Clinical Neurology and Information Processing  
 University Hospital Groningen  
 P.O. Box 30.001  
 9700 RB Groningen, The Netherlands

ABSTRACT

A multiprocessor network concept is described and it's implementation under RT-11. The multiprocessor concept may be seen as alternative to using a multi-user single processor system. However, the multiprocessor option has multiple CPU power and memory available over a single processor system. With decreasing hardware prices, the multiprocessor is the better solution especially in highly demanding environments such as high speed data acquisition and processing. The datacommunication software provides transparent use of remote devices. Memory-only systems may be run using a remote system disk.

INTRODUCTION

Data processing often starts with a single CPU system. A multi-user operating system then seemingly makes CPU and other peripherals available to multiple users. However, with the advent of newer user-friendly software like screen editors, graphics etc. CPU load increased and responsiveness often decreased considerable. A concept with multiple systems, connected by high speed datacommunication links can face the higher demand. Shared disks, printers etc. assure that data are available to multiple users and that expensive peripherals do not run idle for longer periods.

In the laboratory a multi-user system is often inappropriate. High speed data acquisition may block the whole system and frustrate other users. Undisturbed processing may now be realized by giving each application it's dedicated processor. Again, high speed datacommunication links assure that expensive peripherals are not needlessly duplicated, that data may be shared and realise parallel processing (multiple CPU power for a single job).

The multiprocessor goal is believed to have been closely approximated with the package here presented. An earlier version was already described elsewhere [1] (reprints available on request).

In the remainder of this article DC will be used as an abbreviation for datacommunication.

CONCEPT

The multiprocessor concept should fulfill the following requirements:

- High speed communication: remote systems should seemingly be close. The data amount to be stored or processed elsewhere may be quite large,

therefore the transfer speed should be high. Low cost as well as more sophisticated (DMA) hardware should be supported.

- Low overhead, simple communication protocol. This contributes to high speed and may keep CPU load low during transmissions.
- Any network topology may be realized: from the simple point to point connection to complex structures.
- No modification of standard system components: all software should be realized within programs and handlers (device drivers).
- Hardware dependent code should only appear within handlers.
- No arbitration in who issues a transfer request. One site should always be "listening" to the other.

A basic point to point connection is symbolically represented in Fig. 1. System A always issues the transfer requests. It has a DC handler which controls the physical data link. System B has continuously running a so called DC service job (task), which is ready to serve requests from the DC handler at the other side. Note that although the

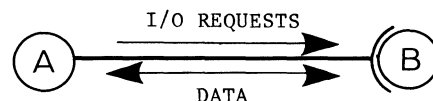


Figure 1. Data link concept

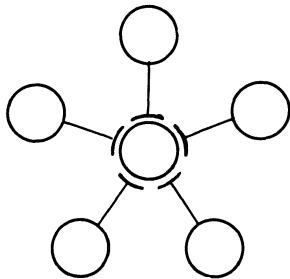
I/O requests go in only one direction, the data go in both directions. With a data read A receives data from B and with a data write A transmits data to B. Besides I/O requests for data transfers, also special function requests may be issued by A. For example by issuing a special function request, A could ask B to return the size of disk unit. Logically there will be several channels within one data link. Each channel is then used to allocate a device unit or file or used to perform a special operation. For example one channel, the message channel is reserved for the exchange of messages ("mail") between A and B. For that purpose B has reserved a "mailbox" file, which stores news for A as well as B and messages received from A for B and visa versa. Such a message from A for B may be e.g. a request to give read and/or write access to a certain device unit.

In order to safely transfer data over the link, a datacommunication protocol is needed. The protocol assures that both sides of the link "understand" what the other is "doing". Also the data integrity can be guarded by applying an error detection algorithm over all data received.

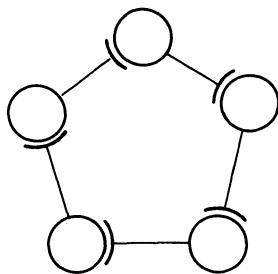
With the basic link now defined even more complex networks can be set up as shown in Fig. 2.



Symmetric link



Star



Ring

Figure 2. Basic network structures

## THE RT-11 LINK

The DC service job runs under RT-11 as a Foreground or a System job. In a monitor with system job support up to 7 DC jobs may run simultaneously. The DC service job links to the DC hardware by using a handler, called the job-handler. A DC job is now started with the following commands:

```
.LOAD QJ ! Load the job-handler
.ASS QJ JOB ! Name it logically JOB:
.FRUN/BUFFER:nnnn DCJOB ! Run the DC job
```

next job:

```
.LOAD DJ
.ASS DJ JOB
.SRUN/BUFFER:nnnn/NAME:DCJOB1 DCJOB
```

etc.

When the /BUFFER:nnnn is specified an extra data buffer of size nnnn words is allocated to the job. This buffer is added to the default internal buffer of 256 words (1 disk block).

The complement of the DC service job is the handler driving the DC hardware at the other side of the link. The protocol used by this handler and the job-handler is a modified Radial Serial Protocol (RSP, [2]). This protocol basically transfers words of data and is described in detail in [1]. The RSP protocol can maintain up to 256 data channels over one link. The current implementation uses only 15 of these channels as these channels are one-by-one coupled to an I/O channel in the service job. Although RT-11 allows defining up to 256 I/O channels for each job, 16 is the default number. As one channel (#15) is used by the job handler, 15 remain to be used for allocating devices. Each data channel has a number 0-14 which also will be called the RSP unit number. An example of DC data channel allocation is given in Fig. 3. As a handler has maximum 8 device units (0-7), only the first 8 data channels can be controlled by the DC handler. Normally the DC handler is defined to the system as a random access device (disk) and therefore cannot be used to simulate e.g. a remote lineprinter.

Both problems, accessing the higher data channels 8-14 and simulating several different type devices are solved with the introduction of pseudo-handlers. These are handlers that do not drive hardware themselves, but use the DC handler for that purpose. The DC-handler has provisions for receiving requests from pseudo-handlers: an internal queue. I/O requests from the DC handler itself and from the pseudo-handlers are stored in this queue and removed from the queue when they are served. These pseudo-handlers make it also possible to use also the data channels higher than 7. For example a service job has allocated channel #10 to a lineprinter. This channel may now be accessed by a pseudo-handler which transforms a request received on device unit number 0 to a RSP unit number 10 by adding the value of 10 to the device unit number. This pseudo-handler may also be defined to the system as a standard lineprinter so that programs and RT-11 utilities cannot "see" the difference between a real lineprinter handler and the pseudo lineprinter handler. Note that the service job may also allocate channels to a DC handler within the same system. In this way devices on all connected

PERFORMANCE

that intermittent problems occur when the BG job does high speed A/D conversion, while the Foreground is also active. The A/D converter may report errors and samples may be lost. In such a situation it would be desired to block the DC jobs until all time critical activity of the BG is stopped. Until now RT-11 has no provisions for such a facility. Therefore the following "trick" is used. A set of subroutines, to be used in a BG program, can block/suspend, unblock/resume DC jobs. When job blocking is required a "no wait" .SPFUN request is send to the job-handler. When the job-handler receives this request, it accepts it but does nothing. This means that no other I/O requests, those from the DC job, can enter the job-handler and the DC job is thus blocked. The BG can in the meantime process it's critical tasks. When the BG wants to resume the DC job it aborts the "hanging" .SPFUN request. The DC job I/O requests can now enter the job-handler and so can resume it's activity.

INSTALLATION OF THE SOFTWARE

First of all the DC job programs DCJOB.REL and in case of special directory support DCJOB.SPD, the utilities and pseudo-handlers are copied to the system disk. The DC jobs and utilities may also reside on another available disk unit. The pseudo-handlers may also be renamed to a more appropriate name. The DC and job-handlers should be inspected for having the correct I/O page and vector addresses. There is also an option, selected by a conditional, for disabling the checksum calculation. When disabled, a fixed bit pattern is transmitted, instead of the checksum, as the tail of each packet. When a packet is received the bit pattern is checked. Although this procedure assures some minimal error detection, data corruption within a packet is not noticed. However, in practice, there are many physical data links which show up seldom an error. And when it occurs, it comes in bursts so that these errors are detected in any case. When the handlers are assembled (with system conditional file SYSGEN.CND) and linked they are copied to the system device.

The DC jobs require that a list of devices is available to which they should open I/O channels at startup. They expect to find this list within the job's data file SY:JBINFO.DAT . Within this file further are stored: default read/write access settings (may be changed while DC jobs run with JSHOW), which channel is the message channel and which reserved for Magtape, a list of the names of available job-handlers, bootstrap programs for memory-only processors and the mailbox.

The file SY:JBINFO.DAT can be created and the data in it are set by the program JOBS. All the modifiable data mentioned above (device&job lists, read&write default access), are stored in a readable format in the file JOBS.CND. Using an editor they may be changed to the appropriate values. Then JOBS should be assembled, linked and run once. During the assembly phase JOBS.CND is read and processed.

The performance of the data link was measured for all hardware types. For this purpose a dummy handler was constructed comparable to the null-handler (NL:). This dummy handler immediately satisfies any I/O request that it receives, but does not perform any data transfer from or to a buffer. Data were transmitted in records of 1024 words (4 disk blocks). As the protocol allows the transfer of max. 256 words/packet [1], four data packets are transmitted for each record. Before these packets are transmitted a command packet is send and after the four data packets an End packet is received. The throughput rates in the table are effective rates. This means:

- including the protocol overhead just described,
- + transfer from memory to interface by DC handler,
- + transfer over the cable (20 m.),
- + transfer from interface to buffer of DC job,
- + 6 I/O requests by DC job to interface handler,
- + 4 I/O requests to dummy when buffer job is 256 w.
- (1 I/O request to dummy when buffer job is 1024 w.)

TRANSMISSION RATES in Kw./s.

|                  | No checksum |      | Checksum calculation |      |
|------------------|-------------|------|----------------------|------|
|                  | 1024        | 256  | 1024                 | 256  |
| <u>Buffer:</u>   |             |      |                      |      |
| <u>Qnector:</u>  |             |      |                      |      |
| 11/23            | 35.7        | 31.3 | 22.7                 | 21.3 |
| +                | (30)        | (28) | (55)                 | (50) |
| 11/23            |             |      |                      |      |
| <u>WB(V)-11:</u> |             |      |                      |      |
| 11/34 ---        | 12.3        | 11.8 | 11.9                 | 11.6 |
| +                | (60)        | (60) | (73)                 | (73) |
| 11/23 ---        | (74)        | (74) | (86)                 | (86) |
| <u>DR-11:</u>    |             |      |                      |      |
| 11/34            | 16.7        | 15.9 | 14.7                 | 14.3 |
| +                | (92)        | (88) | (94)                 | (92) |
| 11/34            |             |      |                      |      |

Note that in the table the machines linked, differ. The PDP 11/34 is in many respects about 20% faster than the LSI-11/23. The Qnector was not set to it's highest speed because of high bus load. However, at it's highest speed a throughput rate of 67.2 Kw./s. (= 1 Mb.) was measured. The hardware specifies max. 250 Kw./s. (= 4 Mb.). Therefore it is demonstrated that the often impressive throughput rates specified by manufacturers do not tell much about the effective throughput under software control! The values between the pharenthesis give the CPU load in % during the transmission at the DC handler side. The CPU load at DC job site shows nearly the same values. Note that these values apply to the test situation! Under "normal" circumstances, where I/O's have to be processed by devices, the CPU load measured is considerable lower (20-40%) !

## CONCLUSIONS

A collection of programs and handlers realises multiprocessing and high speed data communication with RT-11. Remote devices are used in the same way as if they were local. The well-structured software allows all type networks to be setup. Low cost as well as high performance hardware is implemented. Moreover implementing new hardware is a relative small task as only two handlers have to be coded. Cheap memory-only systems can be put to work due to boot capabilities.

## REFERENCES

1. Haenen, H.T.M.  
"A Modular Data Communication Package Providing a Multiuser Environment and Parallel Processing"  
Proceedings DECUS EUROPE  
Coventry U.K., Sept. 1982, pp. 81-88
2. The "Radial Serial Protocol (RSP)".  
Microcomputer Interfaces Handbook. DEC 1980, p. 640
3. Haenen, H.T.M.  
"Disk Usage Analysis and Disk Data Caching under RT-11"  
Proceedings DECUS EUROPE  
Zuerich, Switzerland, August/Sept. 1983, pp. 247-252
4. Haenen, H.T.M.  
"The Disk Data Cache under RT-11"  
Proceedings DECUS U.S.A.  
New Orleans, Louisiana, May 1985

Harry Haenen  
 Dept. Clinical Neurology and Information Processing  
 University Hospital Groningen  
 P.O. Box 30.001  
 9700 RB Groningen, The Netherlands

ABSTRACT

A disk cache is described which speeds up I/O service from disk devices. The cache may also be applied to the system disk in order to seemingly eliminate swapping. The system disk may also be set to write-protect. When the cache runs, only a cache handler (CH:, size ca. 250 words) is added to the RT-11 system. Caching is fully transparent and may run unsupervised.

INTRODUCTION

The usefulness of a disk cache has already been argued before [1] and elsewhere [2,3]. In general cached systems run smoother and faster. Dramatic performance improvement may be seen with so called disk bound programs and/or slow disk devices such as floppies. Many large programs become disk bound because of the large number of data files they often handle. A pleasant side effect of the cache is that it realises setting write-protect of the system disk (novice users can no longer corrupt it). Further in a multiprocessor environment the same system disk may be used simultaneously by multiple processors[4,5]. An earlier version of the cache was already presented some years ago [1]. However, in the meantime one major update was realized. This update assures that the cache is even more easier installed and used. The update also takes into account newer developments such as the logical disk and variable volume size.

WHAT IS A DISK CACHE?

The general principle of a disk cache is well explained by the symbolic two-gear example in Fig.1. The larger gear represents the traditional main disk. It is (relative) large, therefore can contain a lot of data, but is slow in speed and so the access time to data is high. The smaller gear contains less data, but rotates much faster and therefore it's access time is also much lower. So, when there comes a request for data on the large disk, but these data are also present on the small disk, they can be accessed fast. In the remainder of this article, the smaller gear is called the CACHE and physically it will be the well-known virtual memory disk VM:. VM: uses extended memory to store it's data. However, any fast disk could be used to form the CACHE for a (larger) slower disk! So, a Winchester disk could be used to form a cache for a floppy unit.

Now we know that we need a fast storage device, the next question that comes is: "which data should be extracted from the slow disk and be put in the CACHE and how should this be done?". The three most

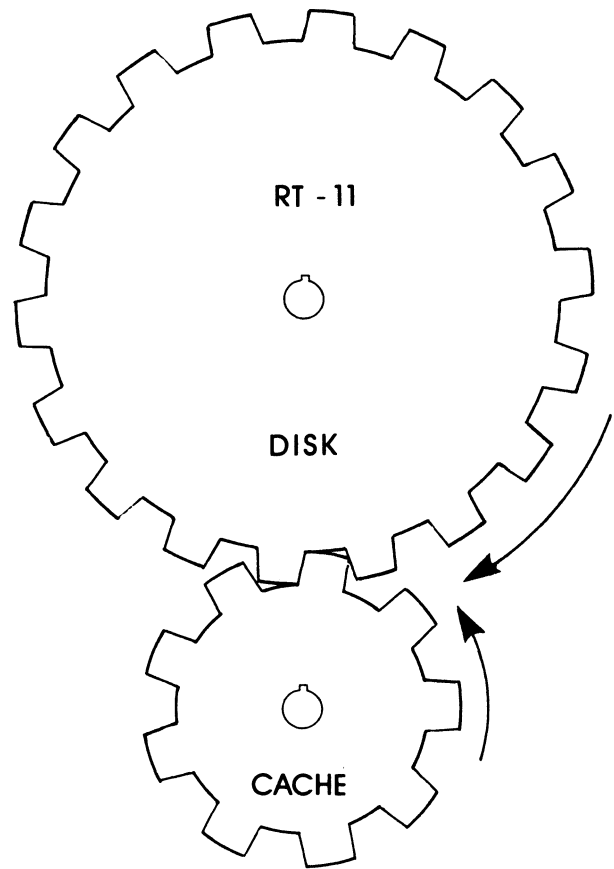


Figure 1. "Gear" cache

well known answers to this "cache strategy" question are, in terms of caching algorithms:

1. Direct Mapping
2. Look Ahead
3. Least Recently Used

In case 1, direct mapping, some predefined disk area's (such as directories, swap files etc.) are permanent present in the CACHE. In case 2, the look

ahead, the disk I/O request data but also data following the requested data are stored in the CACHE. When the next I/O request is for data following sequentially the previous requested data, these data can be retrieved from the CACHE! In case 3, the least recently used, all data from each disk I/O request are stored in the CACHE until it is completely filled up with disk data. When a next disk request comes, the least recently used data from the data queue in the CACHE are removed and the new data are put in. All cache algorithms mentioned above were evaluated given a "typical" RT-11 environment (but is there one!?) and given some defined cache parameters such as CACHE size, maximum I/O request cached etc. The results (see [1], reprints available on request) show that Direct Mapping is far superior over Look Ahead or Least Recently Used. Another advantage is that Direct Mapping is much simpler to implement!

HOW IT WORKS!

We will now concentrate on the details of the direct mapping cache and the implementation under RT-11. With this cache type a contiguous disk space, to be called cache area, is mapped to a contiguous space in the CACHE. In RT-11 reality this contiguous space in the CACHE will be a file, the cache file (named CACHEO.SYS, CACHE1.SYS etc.), on VM: (Fig. 2). Read requests falling within a cache area are serviced fast with data from the CACHE and require no disk access (read hits, fig. 3). Write requests to a cached area update the cache as well as the disk.

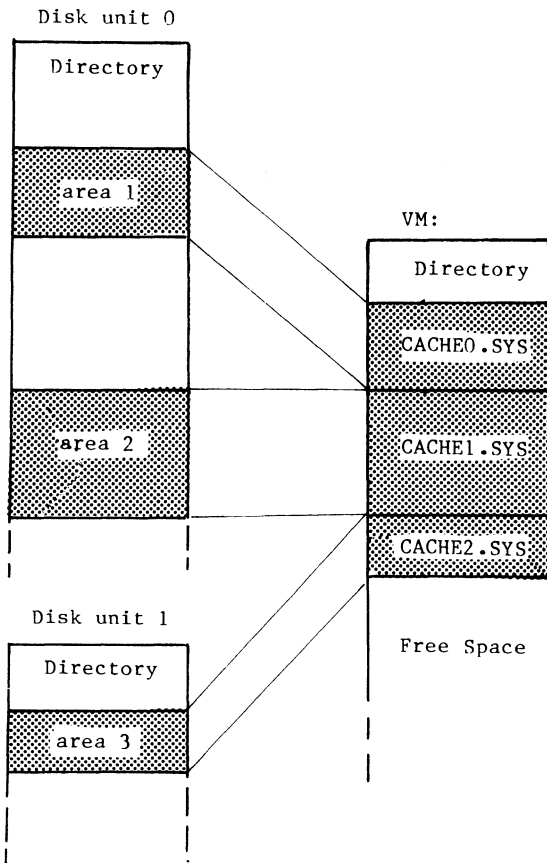


Figure 2. Cache area mapping

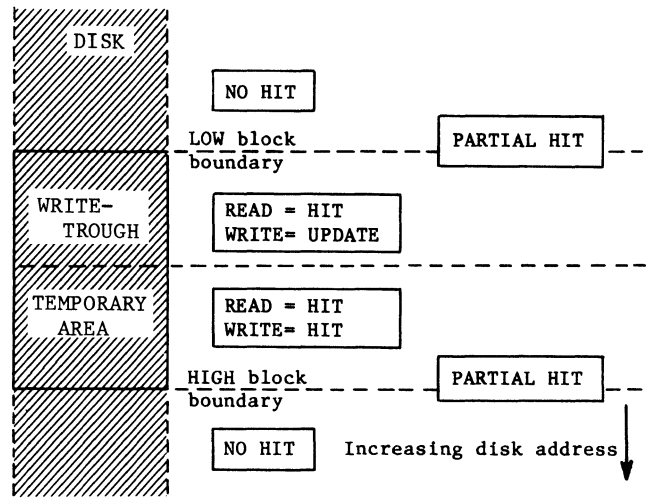


Figure 3. Cache area layout

This is called the WRITE-THROUGH principle, and assures that at any given moment the disk data are correct. As there may also be disk data with only a temporary value, also temporary cache area's may be defined. A cache area may be temporary as a whole or only a higher part. Write as well as read requests falling in a temporary cache area are serviced fast from the CACHE without a disk access (Fig. 3). A good candidate for a temporary cache area is for example the system file SWAP.SYS, which only contains swapped program parts and temporary system data. When the computer crashes, data in the temporary area's are lost while those in the Write-Through area's are retained. I/O requests falling only partially in a cache area are difficult to handle. In order to keep the cache as simple as possible, partial hits only generate a disk request and therefore should be avoided. This can be done by adjusting cache area's to file boundaries and putting not a part but whole directories in a cache area. Partial read hits are no problem. However, partial write hits only update the disk and not the cache. So a next read could retrieve old data from the cache. As partial hits are indicated and adjustment to appropriate boundaries is simple done and often natural, this is no practical problem.

Disk I/O requests are intercepted by the cache handler CH:. A small MACRO (3 words of code) within the disk handler realises this interception (Fig.4) So when the cache runs CH: as well as VM: have to be loaded. The caching algorithm is very simple. Special function requests are returned immediately to the disk handler. However, read and write requests follow:

```

IF (cache hit) THEN
 IF (read) THEN
 (call VM handler)
 ELSE
 (call VM handler ;
 IF (write-through) THEN call DISK hnd)
 ENDIF
 ELSE
 (return to disk handler)
 ENDIF

```

The VM handler is called in a similar way as RT-11 itself calls a handler. The CH: handler also realises setting "write-protect" of disk units. The cache algorithm, coded within CH:, has been worked-out in detail in Fig. 5

HOW THE CACHE IS USED!

The cache package consists of the following components:

- CACHF .SAV      Utility for FILES & DIRECTORIES
- CACHE .SAV     Utility for setting up cache area's by specifying disk block boundaries
  
- CH .SYS        Cache handler CH:
  
- CSHOW .SAV     Utility, prints caching merit, write-protection, disk unit size and cache area's layout.

The components normally reside on the system disk.

Besides the cache handler also VM must be installed in the system. VM: should have enough free space to accommodate the cache files. In a fully occupied 18 bit memory you have 376. blocks available on VM: (a full 22 bit memory something like 5000. blocks!).

The CH: handler determines which disk will be selected for caching. Type SET CH SHOW to check the selected disk. If not appropriate change the disk to be cached by typing SET CH DISK="selected disk number".

CACHE and CACHF are utilities for starting, stopping testing caching and enabling/disabling write protection of disk units. Disk area's to be cached may be specified for each disk unit. Disk units

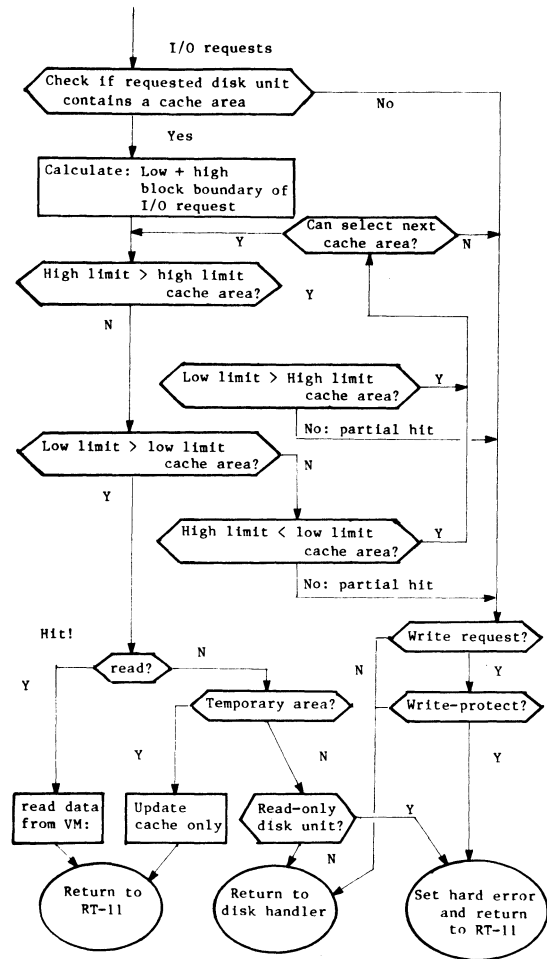


Figure 5. Caching Algorithm

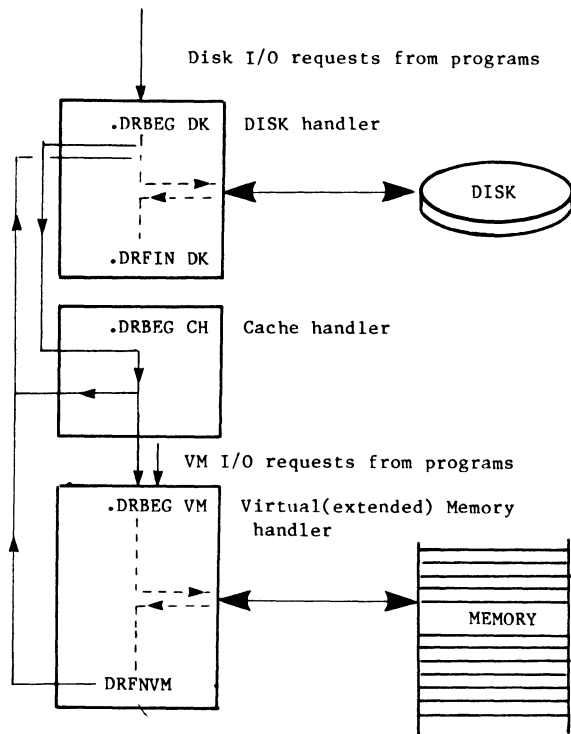


Figure 4. Control and data flow with caching

should be specified in increasing order. If files to be cached are neighbours, they can be put in one cache area/file which has the size of the sum of the individual file sizes.

CACHE sets up the cache area's by specifying absolute disk block boundaries. CACHF is a utility similar to CACHE. However, cache setup is much simpler if you want to cache FILES and DIRECTORIES! You do not have to look for the start addresses and length of files and directories, but you can simple specify filenames for caching files and device names for caching directories. A "/T" switch after a filename puts it in a temporary cache area. All other area's are default Write-Through. With a "/D" switch the directory of the specified device is cached. The disk space from block 1 (the home block) until the end of the directory is put in a write-through cache area.

Also directories and files on logical disks (LD:) which reside on a unit of the cached disk can be easily cached by specifying the logical disk name. Block offsets to the LD disks (Fig. 6) are automatically retrieved from the LD handler by the CACHF program.

The program also creates the necessary cache files on VM:. However, if possible initialize or squeeze VM: before starting CACHF. For it is not permitted that the cached files move when you squeeze VM:!



Note that you can set the system disk to Write-protect if system files to which write operations occur (SWAP.SYS, handler files, ..) are in temporary cache area's. Of course SET operations of handlers are then also temporarily. When the handlers are not in a temporary cache area, KMON prints an error message and the SET command is not executed. The write protection scheme is a nice extra feature of caching and can help preventing corruption of the system disk by "naive" users.

An example of a CACHF run is shown below:

```
.CACHF
Cached device is RK: Caching is OFF

S E L E C T >

Cache/ Stop/ Test/ Read-only/ Write-enable: CACHE

*RK:/D !RKO:=SY:, cache directory of SY:
*RK:RT11FB.SYS !Cache the monitor write-through
*RK:SWAP.SYS/T !Put SWAP.SYS in a temporary area
*RK:X.DAT !File X.DAT in write-through area
*LD2:/D !Cache directory of LD2: (on RK:)
*LD3:Y.DAT !Cache file LD3:Y.DAT (on RK1:)
*/S !Stop entering input
```

Note: /H prints HELP info!

The CSHOW utility prints whether caching is on or off and the disk selected. It further prints the total number of reads and writes to each disk unit.

The size of each disk unit in blocks and also "NoWrite" if a disk unit is set to read-only. A "v" after the size means that the unit is a variable size volume. Further the position of the cache area's, which part is Write-through and temporary, hits rates. An example of the printout is given below:

```
.CSHOW

** RT-11 C A C H E S H O W V6.0 for device QN: **

 Reads Writes DISK Size(blocks)

 5467 709 UNIT 0 4800V
 23 0 UNIT 2 4800V Nowrite
 12014 251 UNIT 5 4800V
 319 0 UNIT 6 1200V NoWrite

!Block-Addr. Write-Thr! Read Up- Part Wrt-tmp DISK
! Cache-area upto ! Hits date Hits Hits unit
!-----!-----!-----!-----!-----!-----!-----!-----!
 1- 148 122 3421 0 0 709 0
 1- 37 37 9810 23 0 0 5
 1- 25 25 219 0 0 0 6
```

HOW IT IS INSTALLED!

First of all the CH: and VM: handlers have to be installed in the RT-11 system. Like other handlers CH: should be assembled with the system conditional file SYSGEN.CND. Default CH has space for maximum 3 cache area descriptors. When a larger number of cache area's will be in use simultaneously, the conditional NAREA in CH: should be set to the appropriate value.

Further a small Macro, called the CACHE Macro, is inserted in the disk handler. This is simply done by running the cache installation program UPD and specifying the two letter filename of the source (e.g. RK.MAC):

```
RU UPD
*RK [Return]
```

UPD then creates a new updated source file with extension .SRC

You can prepare several disk handlers if you want those to have potentially available for caching. The CACHE Macro only occupies 3 words in the handler and performs no function when caching is not on. Assemble, link and copy the updated disk handlers to SY:, e.g.

```
MACRO/OBJ:RK SYSGEN.CND+CHMACR+RK.SRC
LINK/EXE:RK.SYS RK
COPY/SYS RK.SYS SY:
```

What did UPD? The CACHE Macro is inserted directly after the .DRBEG Macro in the handler source. Refer to the following example for the device RK:

```
Existing code(example): After INSERT CACHE:
----- -----
.SBTTL DRIVER ENTRY .SBTTL DRIVER ENTRY
.DRBEG RK .DRBEG RK
 CACHE
 MOV #RKCNT,(PC)+
```

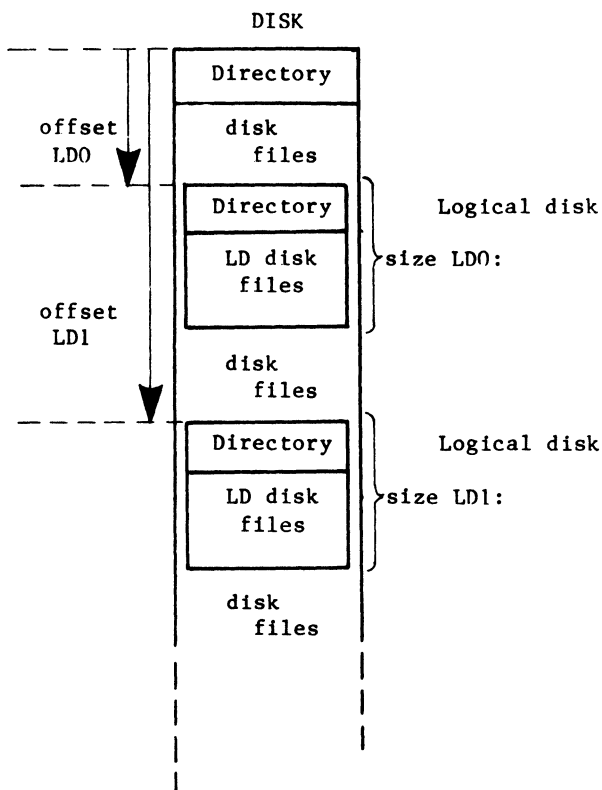


Figure 6. Logical disks on (cached) disk

Update also the VM handler for caching with UPD.  
Copy the VM handler source (VM.MAC) to the disk on  
which you are working, then:

```
RU UPD
*VM [Return]
```

UPD then creates a new updated source file: VM.SRC,  
then assemble, link:

```
MACRO/OBJ:VM SYSGEN.CND+VMMACR+VM.SRC
LINK/EXE:VM.SYS VM
COPY/SYS VM.SYS SY:
```

What is done by UPD? UPD replaces the .DRFIN macro  
in the VM handler source (appears at two locations;  
one for SJ/FB and one for XM) by macro call DRFNVM:

| existing code:      | new code:           |
|---------------------|---------------------|
| -----               | -----               |
| VMERR: MOV VMCQE,R5 | VMERR: MOV VMCQE,R5 |
| BIS #HDERR\$,-(R5)  | BIS #HDERR\$,-(R5)  |
| VMDONE: .DRFIN VM   | VMDONE:DRFNVM       |

Copy CACHF and/or CACHE, CSHOW to SY: and the  
installation is finished!

### CONCLUSIONS

By adding a cache handler to the RT-11 system, disk  
data caching has become a ready-to-use feature.  
Systems run faster and "smoother" with the cache on.  
With disk-bound programs and/or slow disks  
considerable performance increase will be noticed.  
The cache is fail-safe as a Write-Through algorithm  
is used. Therefore using the cache has many  
advantages over e.g. using VM: as a fast disk. Data  
on VM: are lost when a powerfail or boot occurs.  
Further the cache uses VM: only for those disk  
area's which are most frequently accessed. The VM  
cache also may be servicing several disk units.

In a multiprocessor environment the cache is also  
very effective. Data transfers over the data link  
can be reduced by having caches in (memory-only)  
processors. Also multiple processors may use the  
same system disk [1].

### REFERENCES

1. H.T.M. Haenen  
"Disk Usage Analysis and Disk Data Caching  
under RT-11"  
Proceedings DECUS EUROPE  
Zuerich, Switzerland, August/September 1983,  
pp. 247-252
2. P.T. Thordarson  
"Performance and Disk Data Caching"  
Proceedings DECUS U.S.A.  
New Orleans, Louisiana, April 1979,  
pp. 1113-1121
3. D.K. Brown, K. Strutynski, J.H. Wharton  
"Tweaking more Performance from an Operating  
System"  
Computer Design  
May No. 6, Vol. 22, 1983 pp. 193-204
4. H.T.M. Haenen  
"A Modular Data Communication Package providing  
a Multiuser Environment and Parallel Processing"  
Proceedings DECUS EUROPE  
Coventry U.K., Sept. 1982, pp. 81-88
5. H.T.M. Haenen  
"Multiprocessing and High Speed Datacommunication  
with RT-11"  
Proceedings DECUS U.S.A.  
New Orleans, Louisiana, May 1985



Donald J. Mandley  
Engineering Mechanics Department  
General Motors Research Laboratories  
Warren, MI 48090-9057

### ABSTRACT

A software scheme was developed on a DEC computer for a real-time temperature graphics data acquisition system using the F/B monitor of RT-11. This system scans all 67 temperature transducers from a automotive plastic hood mold at 2-second intervals and produces user selected data plots within seconds after each part is molded (each cycle is normally 60-100 sec long). Having data graphs within seconds after each mold cycle enabled the experimentalists to draw meaningful conclusions from the temperature variations during the test, especially during the initial start-up transient. This data acquisition system uses a DEC LSI-11/23 (MINC) computer with a DRV11 parallel interface to communicate with a third-party multichannel digitizer. ReGIS commands to a VT-125 graphic terminal provide fast data plots and a LA-100 printer provides optional hard copies.

Fortran and assembly language routines were written to control the many functions of the data acquisition system. Virtual memory was used to store the background subroutines for fast overlaying and for temporary data storage (up to 90 mold cycles). This menu-driven real-time data acquisition system was developed without modifying the standard RT-11 operating system or the DEC hardware. The important features of the hardware, RT-11 operating system, and the modular-software will be discussed to illustrate the usefulness, ease and user-friendliness of this system.

### INTRODUCTION

A data acquisition system was developed as part of an experimental program to verify an optimal thermal design (1) in a production mold for a fiberglass automotive hood. The object of the design is to minimize the spatial temperature variations on the cavity surface of the mold, thus promoting uniformly cured parts at rapid cycle times.

A method was needed to monitor and present temperature data from 67 mold thermocouples rapidly enough to allow the experimentalist to evaluate these temperature distributions during the molding process. For this reason it was necessary to develop a real-time data acquisition and graphics system.

In order to illustrate the usefulness, ease, and user-friendliness of this data acquisition system to the general reader as well as the experimentalist and computer programmer, this paper will present the full system starting with its general functions and move towards a more detailed discussion of its more significant elements.

### SYSTEM REQUIREMENTS

The inputs to the data acquisition system were 67 thermocouples which measured temperatures on both the molding surface and steamline walls as shown in Figure 1. These measurements provide the experimentalist with check points against predicted values. The data acquisition system must read the thermocouple values from the digitizer and perform the following functions during the molding process.

1. Scan digital inputs from 67 thermocouples every 2 seconds for up to 90 molding cycles of approximately 60-100 seconds duration.
2. Acknowledge signal from press computer to establish the duration of each molding cycle and catalog data according to each cycle.
3. Store all of the data, from every data scan, for up to 6 of the 67 thermocouples which can be selected by the user before, and changed during, the test.

4. At the end of each cycle, compute the average temperature value for each of the 67 thermocouples and store the results.
5. Generate user-selected data plots in real-time.
6. Provide for data storage on a diskette either during or after the test.

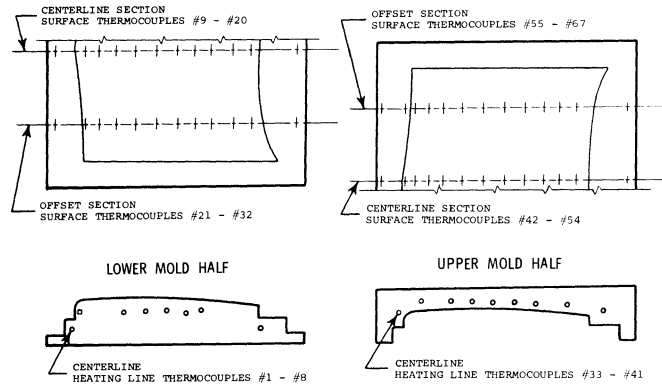


Figure 1 MOLD SHOWING THERMOCOUPLE LOCATIONS

parts to take advantage of the F/B monitor. The block diagram in Figure 3 illustrates the main parts of the data acquisition system. The most important task of the system was to record data from the digitizer every 2 seconds during each mold cycle, average it at the end of each cycle, and then store the data in specific locations in computer memory.

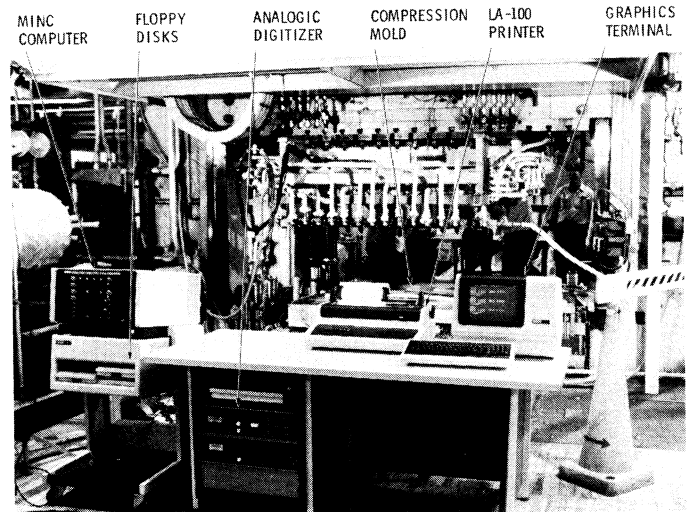


Figure 2 DATA ACQUISITION SYSTEM HARDWARE

### SYSTEM OVERVIEW

Standard data loggers are available to record temperature data, and separate computer software exists to produce off-line graphics. However, to perform both of the above tasks together with the speed and resolution necessary for interactive real-time graphics for the fiberglass mold, it was necessary to develop a data acquisition system with a special computer software scheme that could synchronize a constant data input rate with both data processing and user interactive graphics. Before discussing this scheme in detail, however, we begin by giving a general overview of the system functions.

A Digital Equipment Corporation (DEC) computer and an Analogic digitizer provided the required hardware features necessary to meet the experimental requirements. The data acquisition system hardware is shown in Figures 2 and 3. A terminal, hardcopy printer, and floppy diskettes are available to the user as output devices. The DEC computer (LSI-11/23 microprocessor) is a single-user computer, dedicated to a single task when used with the real-time operating system (RT-11). See Appendix A for additional computer features.

The RT-11 operating system has a unique feature that allows two separate programs to run alternately and independently of each other. This feature is called the foreground and background (F/B) monitor. DEC gives the highest priority to the foreground program allowing the background program to run only when the foreground program is placed in a suspended state.

The data acquisition software was written in two

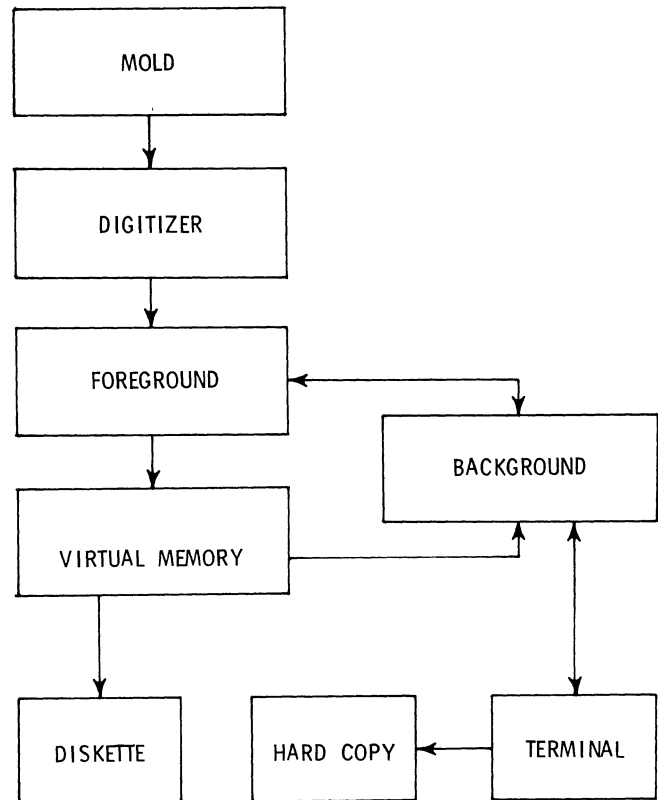


Figure 3 DATA ACQUISITION SYSTEM BLOCK DIAGRAM

Therefore, the software written to perform these tasks was placed in the foreground area to assure a constant data scan rate. One might think of this program as a data logger section. The foreground is activated every 2 seconds by the real-time clock within the computer and can complete its data scanning tasks within 15 milliseconds. Therefore, the background program has most of the available computer time to process the data and produce the user requested graphics.

The foreground program stores the thermocouple data in two different forms. A direct data buffer stores every measurement for up to 6 of the 67 thermocouples and a average data buffer stores the average temperatures for all 67 thermocouples during each cycle. Placing the 6 channel limit on direct data was due to the large amount of memory required for saving each data value from each scan. This limitation, however, did not compromise the experimental objectives since the average data was the primary data form.

The remaining data acquisition tasks are performed by the background which executes only when the foreground is not running. The background programs were written to allow the user to interact with the data collected by the foreground programs. These background programs produce menus which allow the user to choose from six types of data plots. The graph is usually generated on the terminal screen within five seconds after the user selects the last plotting parameter. After this, the user can request a hard copy from the on-line printer. The background programs also allow the user to transfer both the average and direct data files to a floppy diskette during the test for permanent storage. The user could use this option if critical data might be destroyed by computer power failures during a storm or by hostile machines located near the computer. However, normally this data transfer is done after the test is complete by using standard DEC copy commands.

The software scheme that synchronizes the above software programs with the hardware and allows for user interaction will be discussed after some additional technical features are presented.

#### IMPORTANT TECHNICAL FEATURES

The following DEC computer hardware and software features were necessary to accomplish the speed and resolution requirements for the mold evaluation:

1. Fast data storage and retrieval from virtual memory during the test.
2. Software overlays for the background subroutines.
3. Fast data input using a foreground assembly language subroutine.
4. Clock interrupts to maintain a constant data scanning rate from the digitizer.
5. Direct graphics instruction set (ReGIS).

New DEC RT-11 virtual memory support provided 200K bytes additional memory for temperature data. DEC read and write commands are valid for communicating with virtual memory for fast data access. This

feature allowed virtual memory to be a common data area where both the foreground and background can quickly access the thermocouple data.

The software overlay feature of RT-11 permitted the large background programs to operate within a small region of memory. RT-11 is a fast operating system, but it only supports up to 64K bytes of program memory, including its own operating system software. Therefore, all the background subroutines were written as separate independent modules. These background subroutines are placed in virtual memory and only the subroutine required for a specific plot is copied into system memory. This was done from virtual memory to save time whenever the user calls for a new type of graph. Overlaying is much faster from virtual memory than from a floppy diskette which is the normal method.

Rapid data scanning from the digitizer was accomplished through an assembly language subroutine in the foreground. DEC assembly language converts a software program, written with DEC mnemonics, directly to machine code for optimal execution speed. A fast data path from the digitizer to the computer was necessary to allow the user the maximum amount of time to interact with the graphics. It turned out that the assembly language routine was able to ask and receive data faster than the digitizer could transfer it. Therefore, the assembly language routine adds each new thermocouple value to those previously accumulated from that channel while it waits for the next input, thus saving additional time at the end of each cycle to average each channel.

The programmable clock within the computer activates the assembly language data scanning subroutine to provide a constant data sampling rate from the digitizer. However, before reading data, the assembly language routine issues a special set of commands to reactivate the foreground environment which allows the main foreground program to resume after the assembly language completes its tasks.

The most important technical feature of the DEC equipment that allowed real-time graphics to be a part of this data acquisition system was being able to write direct graphic commands. Standard FORTRAN callable subroutines are available from DEC to create graphics. However, they use nearly half of the programming memory and operate too slowly for our purposes. DEC provides a Remote Graphics Instruction Set (ReGIS) to create an image by directly turning on or off each individual pixel within the VT-125 terminal. To use ReGIS, it was necessary to write separate software subroutines for each different graphic option. These software subroutines position all of the graphic lines, text, and data points on the terminal screen. Writing the plotting subroutines with ReGIS commands was time-consuming but allowed the system to operate at the necessary speed.

#### SPECIAL SOFTWARE SCHEME

Synchronizing the various activities of the data acquisition system with the many hardware and software items already described was the most critical element in this project. Creating a software scheme to record data at precise intervals, while giving the user rapid access to the data for a continuous flow of optional temperature plots, provides the

experimentalist with a valuable tool. Figure 4 illustrates the modularly written software and the various data flow patterns available from the mold to the foreground, background, virtual memory, graphics terminal, and the floppy diskette.

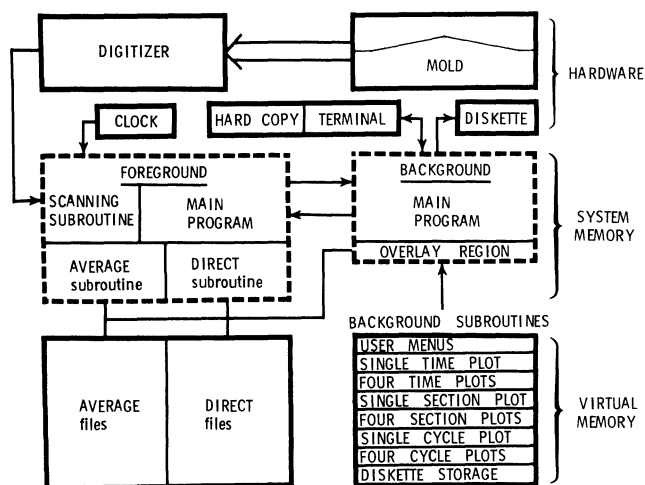


Figure 4 DATA FLOW BLOCK DIAGRAM

Since data scanning is extremely important, the clock was used to activate the foreground data scanning routine from its idle state at 2-second intervals throughout the test. After the foreground is activated by the clock-driven data scanning routine, it must complete the following basic tasks:

1. Accept data from the digitizer.
2. Separate direct and average data measurements.
3. Store data in virtual memory files if current scan is the last one in the mold cycle.
4. Open an input message buffer to receive asynchronous background messages.
5. Respond to any data request from background.
6. Suspend itself until the next clock interval.

These six tasks establish the foreground protocol and can be accomplished in less than 10 milliseconds.

The foreground tasks increase when the user selects one of the temperature plots from the background menu. The background must then communicate with the foreground to determine whether the required plotting data is located in a local foreground buffer or in a virtual memory file. For the background to obtain this information, it must synchronize itself with the foreground by sending a data request message to the foreground. The foreground receives this data request message asynchronously whether it is active at the time or not. If the foreground was not active when it received the message, it cannot respond to the message until it is reactivated by the clock driven data scanning routine and completed

the primary data tasks. Only then will the foreground look for a background data request.

If the foreground did receive such a background data request message, it will establish a synchronized communications link with the background to initiate the sequence of events for a data transfer. Both the foreground and background place themselves in a standby mode after each message, creating this synchronized communication link.

The complexity of this foreground and background chain of events depends on whether the requested data is located in a local foreground buffer or in a virtual memory data file. The simplest form is when the requested data is already available in the foreground area and must be passed to the background to create a plot for the user. This can be accomplished by passing the data buffer to the background with a wait command to make sure the background received the data buffer before the foreground suspends itself. After the foreground suspends itself, the background can run asynchronously to plot the data. If and when the background needs more data, it will again establish the synchronized communication link.

The synchronized communication link becomes more involved when the foreground returns a message to background stating the requested data is located in a virtual memory data file. In order for the background to fetch data from both the foreground and virtual memory, additional steps must be taken to assure the foreground protocol will not be disturbed. Only one program at a time can be attached to the same virtual memory data file; therefore, the foreground must first detach itself from the file, send a message to background stating the data is located in virtual memory, and then place itself in standby. After receiving the message, the background will attach itself to the designated virtual memory file, retrieve the data, detach itself, and return a message to foreground stating it has the data and wants to plot the data asynchronously. After receiving this message, the foreground will reestablish its link to virtual memory and suspend itself allowing the background to create the plot. Even with these extra communication messages for the virtual memory data, the foreground still only needs to be active 2 percent of the 2-second data scan interval. This leaves 98 percent of the time for the user to generate real-time graphics. This is why the user is unaware of the foreground data processing activities.

#### DATA HANDLING

All of the thermocouples are hard-wired to the digitizer. The computer is programmed to read each thermocouple value, individually from the digitizer, every 2 seconds during the entire test. The following sections will describe how the data is processed to produce various temperature plots. See Appendix B for the Analogic digitizer features.

#### Data Processing

All temperature values during a given cycle are held in a separate temporary buffer in foreground. A signal from the press computer notifies the data acquisition system at the end of the mold cycle at which time these data values are passed to the virtual memory storage area. The temporary data

files within the virtual memory area limit the data acquisition system to 90 cycles before it is necessary to copy them to a floppy diskette.

All of the thermocouple data values were stored in millivolt values from the digitizer. Only those data values used for a graph were scaled through the functions shown in Figure 5. This method saved the time necessary to convert each data point from a voltage value to a temperature value.

#### DIGITIZER

```
GAIN = DIGITIZER I/O GAIN (0.2326 V/MV)
CNT = 12 BIT A/D TEMPERATURE COUNT (0 TO 4095)
FSO = DIGITIZER FULL SCALE OUTPUT (20.0 V)
SCALE = RATIO BETWEEN 130 TO 190 deg C (18.1 deg C/MV)

SCALE / GAIN x FSO / 4096 = 0.380 deg C/CNT
```

#### GRAPHICS

```
TEMP = THERMOCOUPLE TEMPERATURE IN DEGREES CELCIUS
PR = PIXEL RANGE ON Y-SCALE
TR = TEMPERATURE RANGE ON Y-SCALE
P = PIXEL COORDINATE AT TOP END OF Y-SCALE
T = TEMPERATURE AT TOP END OF Y-SCALE
PC = PIXEL COORDINATE CORRESPONDING TO TEMPERATURE COUNT
```

```
TEMP = (0.380 deg C/CNT) x (CNT) + 5 deg C offset
PC = (T - TEMP) x PR / TR + P
```

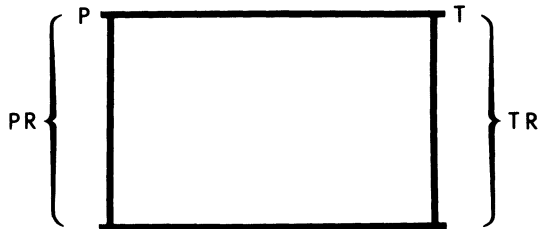


Figure 5 DATA SCALING FUNCTIONS

The usual operating temperature range for the mold was 130 to 190°C. The temperature-voltage relationship within this range is linear within 0.1°C. It should be noted, from Figure 5, that a +5°C offset was added to provide the best linear fit between 130 to 190°C. This causes a 5 degree error at zero; therefore, a small error will occur on the 0 to 200 degree scale. This coarse scale was only included to give the user a rough indication of any thermocouple value during the mold warm-up period. All meaningful data taken during the test was plotted using the 130 to 190 degree option.

All of the data stored in virtual memory and the floppy diskettes are stored in their original form in order to provide maximum precision for off-line data analysis.

#### User Graphics Options

After the first mold cycle, data is available for background options. The background will produce three basic plots with specific options: time,

section, and cycle. Each of these types can be displayed in a large plot (high resolution) or four small plots (low resolution).

The time plots are drawn using the direct data from the last molded hood. Each plot will show all temperature values for a single thermocouple from the last completed cycle as illustrated in Figures 6 and 7. The background fetches this data from the foreground buffer before it is transferred to virtual memory.

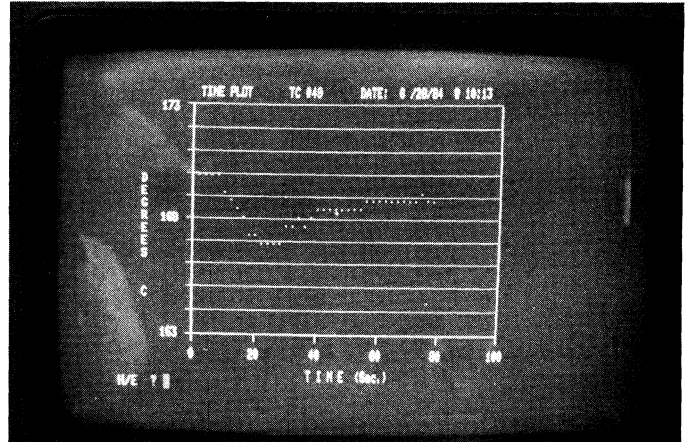


Figure 6 SINGLE TIME PLOT

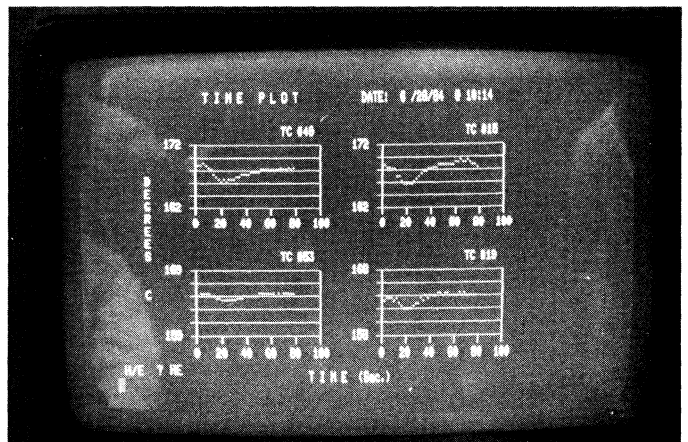


Figure 7 FOUR TIME PLOTS

The section plots are drawn with the data stored in the average data buffer. Each plot will display the average temperature values from the last completed cycle for each thermocouple located along one of the four cross sections shown in Figure 1. The data points indicated with small circles in Figure 8 are the streamline wall temperatures and those indicated by connected lines are the mold surface temperatures. Each data point represents the average thermocouple value from the last molded hood.

The cycle plots are drawn using all of the previous average data for the channels selected as illustrated in Figure 9. This is the only plot that



requires data from both the foreground and virtual memory. This task required the most complex software scheme because the foreground must keep a log for all of the average data buffers. The foreground and background must continually pass messages in order for the background to know where to locate the data. Each plot will display the average temperature values of a single thermocouple as a function of the mold cycle. This is the most used graphics routine because it indicates to the user when the mold temperature has stabilized for specific process conditions.

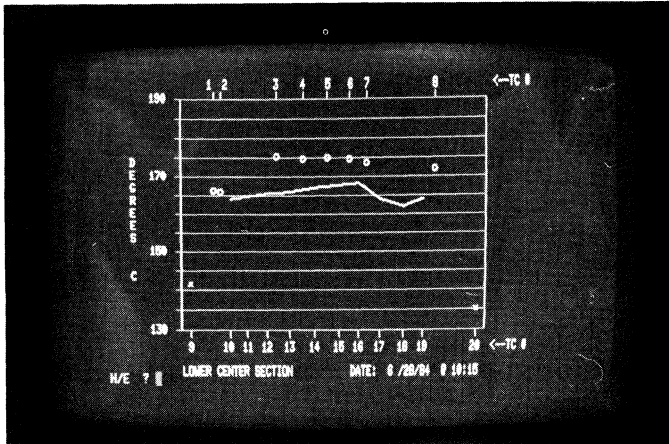


Figure 8 SINGLE SECTION PLOT

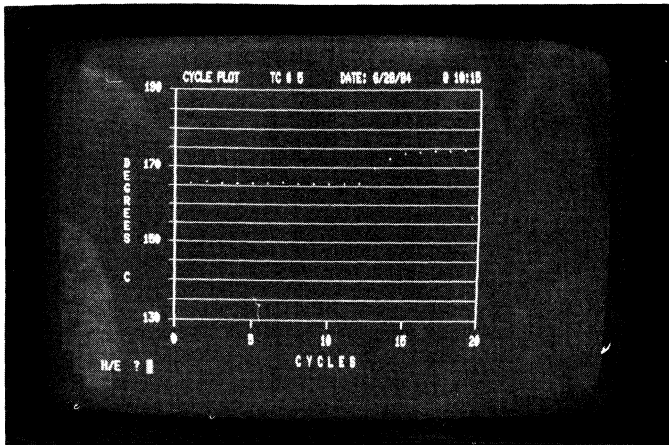


Figure 9 SINGLE CYCLE PLOT

The user will always have the option after each plot to press the return key and wait for an updated plot having the same options or to press the H key for a hardcopy and/or the E key for ending that specific plot. Ending will automatically display the graphics menu on the terminal screen as shown in Figure 10. The user also has the option to copy all of the average and direct files from virtual memory to a new floppy diskette during the test.

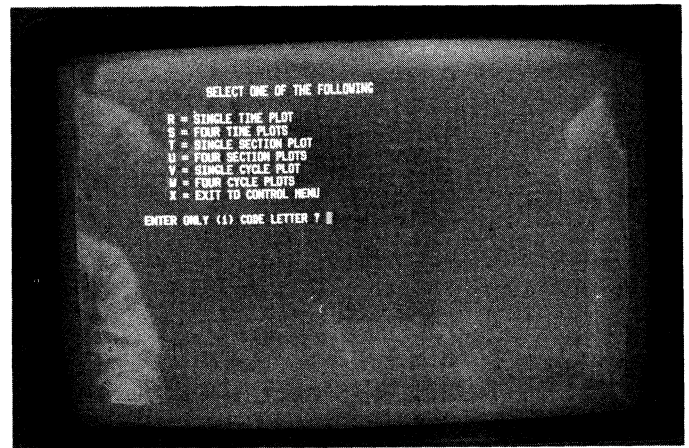


Figure 10 USER GRAPHIC MENU

### Permanent Disk Storage

This option will allow the user to copy the direct and average data files from virtual memory to a new floppy diskette. The user should realize that this data transfer routine could use most of the next mold cycle, leaving little or no time to display the data from the last part. The only reason for the user to use this routine is when important data could be lost due to electrical interference problems from near-by machinery or from power outages. Normally all data files would be transferred from virtual memory to a new floppy diskette after the test is ended with standard copy commands. This must be done before the computer is powered down because the virtual memory is destroyed after the computer power is turned off.

### SYSTEM VERIFICATION

To determine the overall precision of the data acquisition system, each thermocouple channel was connected to a thermocouple which was heated to near 190°C. The test apparatus consisted of a Thermotron oven, a large aluminum bar, and four thermocouples as illustrated in Figure 11. The aluminum bar had a hole drilled into its center for the tips of the four thermocouples. This allowed us to establish a stable reference temperature. Approximately 1 hour was required to connect the four thermocouples to each of the 17 cards (68 thermocouple inputs) and run our own special software program to record each card separately. This test was performed every day for five consecutive days leaving the oven on continuously.

The test data shown in Figure 12 illustrates typical results from one day's test. The data formed a bell shape curve with all of the measurements within  $\pm 1^\circ\text{C}$ . Most of these variations can be attributed to the digitizer (see Appendix B).

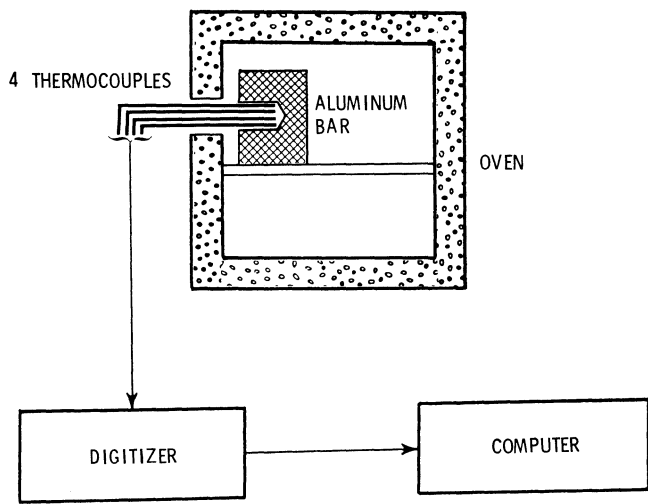


Figure 11 SYSTEM EVALUATION LAY-OUT

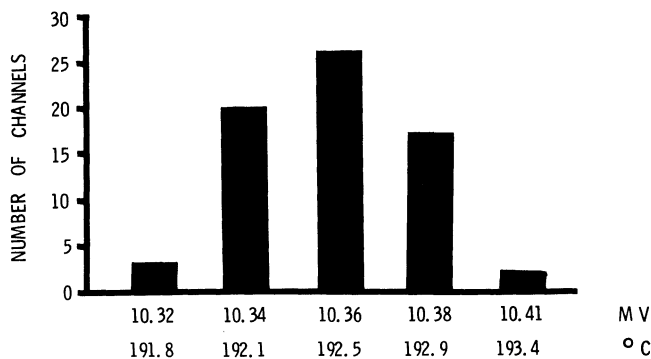


Figure 12 TYPICAL DATA FROM SYSTEM EVALUATION

### SUMMARY

Special software was written with a unique scheme to synchronize, control, and communicate among various independent software and hardware elements to create a real-time interactive graphics data acquisition system. This software was written to interface an ANDS5400 Analogic digitizer to the DEC (MINC LSI-11/23) computer with a graphics terminal to display mold temperatures in various formats. This system was used to verify the results of an optimal thermal design for a production fiberglass hold mold.

This special software enabled real-time interactive graphics to be incorporated into a data acquisition system providing a means to quickly evaluate the changing temperature patterns during the actual test. The system records and plots mold temperatures with a precision of  $\pm 1^\circ\text{C}$  from 67 thermocouple locations in the mold. All 67 thermocouples are scanned every 2 seconds during each mold cycle which is normally 60-100 seconds long. At the end of each cycle, the data may be plotted in six different formats selected through user menus. Having data graphs within seconds after each mold cycle enabled the experimentalists to draw meaningful conclusions from the temperature variations during

the test, especially during the initial start-up transient. Although special software was written for this specific project, the same concept can be applied to other experiments requiring real-time graphic display of transducer measurements.

### ACKNOWLEDGMENTS

The author wishes to thank Edward Clayton for his support with interfacing the digitizer to the computer, making the special cables, calibrating the digitizer, and testing the system for the temperature evaluation each morning for several weeks. Also, to Dan Dowdall for testing the software menus and documenting the problems which made them easy to find and debug.

The author also expresses his appreciation to the Project Leader, Dave Caulk, for his positive attitude and willingness to discuss and resolve technical concerns.

### REFERENCES

1. M. R. Barone and D. A. Caulk, "Optimal Thermal Design of Compression Molds For Chopped-Fiber Composites," *Polymer Engineering and Science*, Vol. 21, pp. 1141-1148 (1981).

### APPENDIX A

#### DEC Computer Features

A Digital Equipment Corporation (DEC) computer was used with a real-time operating system (RT-11). The DEC computer consists of the LSI-11/23 microprocessor, RX02 double density floppy disk drive, 16 bit parallel interface card, VT-125 graphic terminal, and the LA-100 dot matrix printer as shown in Figure 2. The computer supports up to 256K bytes of memory, but the RT-11 operating system will only allow 64K to be used for the system and programming software. Therefore, the remaining 200K bytes were used as virtual memory for fast data storage and background overlaying. Data stored temporarily in virtual memory can be transferred during or after a test to a floppy diskette for permanent storage. The data path to and from virtual memory is illustrated in Figure 4. Data for up to 90 molding cycles can be stored in virtual memory and then transferred to a diskette for off-line analysis.

A standard DEC general purpose interface circuit board (DRV11) was purchased for the computer to provide 16 bit parallel communication between the computer and the thermocouple digitizer. This DRV11 interface board passes 67 thermocouple values in less than 5 milliseconds.

The DEC VT-125 terminal provides both text and graphic viewing for the user through ReGIS commands. After the software package is loaded into the computer, the user is continually being prompted through a text menu displayed on the terminal screen. Once the user has selected a specific data graph option and the text is cleared from the screen, ending the text mode, the terminal enters the graphics mode displaying the chosen graph. ReGIS commands produce graphic images with a resolution of 240 x 768 pixels. Each of these pixels can be turned on by sending ASCII character to the

VT-125 terminal from the background software to create a specific graph. Each pixel location has four gray levels which were used to enhance the data points relative to the grid lines. This programming method is tedious but execution is very fast. For example, Figure 6 was fully displayed on the VT-125 terminal in less than 5 seconds.

## APPENDIX B

### Analogic Digitizer Features

An Analogic ANS5400 transducer digitizer (Figure 2) converts each low voltage thermocouple signal to a 12 bit binary number through a multiplexer and transfers it to the DEC computer upon request. This 12 bit number is left-adjusted in the 16 bit DEC input; therefore, the assembly language routine must shift each thermocouple value 4 bits to the right. The least significant bit is approximately equal to 0.38°C when using type J (I.C.) thermocouples.

The Analogic digitizer, Model ANDS5400, was purchased with the following items:

| <u>Quantity</u> | <u>Part No.</u> | <u>Description</u>      |
|-----------------|-----------------|-------------------------|
| 1               | ANDS5400-B-B    | Master chassis          |
| 1               | ANDS5400-B      | Expansion chassis       |
| 1               | AC261           | Signal processor        |
| 1               | AC2712-2D       | A/D converter           |
| 19              | AC4272-"J"      | 4 ch thermoplexer cards |
| 1               | AC23B           | GPI card                |

### Calibration

Two different calibrations are available for the digitizer. The first calibration mode checks only the digitizer by disconnecting all thermocouples through an internal relay. Voltage levels that produce plus full scale, zero, and minus full scale outputs are internally connected to the input amplifiers of each channel. Both the gain and offset can be adjusted for every channel. The digitizer is normally stable within  $\pm 0.76^\circ\text{C}$  ( $\pm 2$  counts). Details for this calibration can be found in the Analogic ANDS5400 instruction manual.

The second calibration mode includes the thermocouples and allows the user to adjust the cold reference junction on each four channel input card. This adjustment calibrates the temperature sensing resistor for zero volts output with the thermocouple at 0°C. Only one temperature sensing resistor is available for each input card to compensate for the reference junction of the type "J" input thermocouple wire. Each card contains four thermocouple channel inputs, channel #0 to channel #3. Channel #0 is located on the bottom edge of the card and channel #3 is located at the top. The sensing resistor is located in the center between channels #1 and #2. Channel #1 on each card was chosen to calibrate this cold reference junction. The calibration was accomplished by placing the thermocouple into an ice bath (0°C) and adjusting the output of channel #1 to 0.0 volts. The other three channels were normally within 0.76°C (2 counts) of zero.

Robert Walraven  
Multiware, Inc.  
Davis, California

Ralston Barnard  
Sandia National Laboratories  
Albuquerque, New Mexico

ABSTRACT

Good programming style makes code more readable and maintainable. Here are some suggestions from experienced FORTRAN programmers about style that you might find useful.

Don't be overly clever.

Clever programming tricks make for code that is difficult to understand. Say what you mean as simply and directly as you can. It is important to make code clear, so people can debug, maintain and modify it.

For example, what does this code do?

```
DO 1 I=1,N
DO 1 J=1,N
1 X(I,J) = (I/J)*(J/I)
```

It puts ones on the diagonal of the matrix X and puts zeros everywhere else.

Here is a clearer way to code the operation that is actually faster:

```
DO 20 I = 1,N
DO 10 J = 1,N
IF (I.EQ.J) THEN
X(I,J) = 1.0
ELSE
X(I,J) = 0.0
ENDIF
```

```
10 CONTINUE
20 CONTINUE
```

Follow the KISS principle.

Keep it Simple, Stupid!

Use spaces freely to improve the appearance of code.

Here is some code that is badly in need of some spaces:

```
IF(X-Y)30,10,10
10 IF(Y-Z)50,20,20
20 W=Z
GO TO 70
30 IF(X-Z)60,40,40
40 W=Z
GO TO 70
```

```
50 W=Y
GOTO 70
60 W=X
70 ...
```

It is rewritten in the next example with spaces added for readability.

(This example will be rewritten several times in the following examples.)

Don't use ARITHMETIC IF statements. They are too complicated.

Notice how difficult this code is to follow:

```
IF (X-Y) 30,10,10
10 IF (Y-Z) 50,20,20
20 W = Z
GO TO 70
30 IF (X-Z) 60,40,40
40 W = Z
GO TO 70
50 W = Y
GO TO 70
60 W = X
70 ...
```

The next example shows the same code without ARITHMETIC IFs.

Avoid GO TO statements whenever you can.

The following code is difficult to understand because it uses GO TO statements:

```
IF (X .LT. Y) GO TO 30
IF (Y .LT. Z) GO TO 50
W = Z
GO TO 70
30 IF (X .LT. Z) GO TO 60
40 W = Z
GO TO 70
50 W = Y
GO TO 70
60 W = X
70 ...
```

The next example shows the same code without GO TO statements.

Use meaningful comments to explain what logical blocks of code can do.

```
C..... W = the smallest of X, Y, and Z

 W = X
 IF (Y .LT. W) W = Y
 IF (Z .LT. W) W = Z
```

Use meaningful names where possible.

```
C..... SMALL = the smallest of X, Y, and Z

 SMALL = X
 IF (Y .LT. SMALL) SMALL = Y
 IF (Z .LT. SMALL) SMALL = Z
```

Use library and intrinsic functions when possible.

```
C..... SMALL = the smallest of X, Y, and Z

 SMALL = AMINO (X, Y, Z)
```

This performs the same operation as the ten lines of code we started with.

INDENT

It has been experimentally proven that for psychological reasons two to three spaces of indentation make the most readable code.

```
No indent: IF (X .LT. Y) THEN
 IF (Y .LT. Z) THEN
 CALL XYZ
 ENDIF
 ENDIF
```

```
3 spaces: IF (X .LT. Y) THEN
 IF (Y .LT. Z) THEN
 CALL XYZ
 ENDIF
 ENDIF
```

```
Tab: IF (X .LT. Y) THEN
 IF (Y .LT. Z) THEN
 CALL XYZ
 ENDIF
 ENDIF
```

Make IF-THEN-ELSE blocks stand out.

```
--- BAD ---

.
.
CALL SUBRO
IF (I.EQ.1) THEN
X = 12.
CALL SUBR1
ELSE
X = 13.
CALL SUBR2
ENDIF
CALL SUBR3
.
.
```

```
--- GOOD ---

.
.
CALL SUBRO
IF (I.EQ.1) THEN
 X = 12.
 CALL SUBR1
ELSE
 X = 13
 CALL SUBR2
ENDIF
CALL SUBR3
.
.
```

Eliminate simple FORMAT statements.

```
--- BAD ---

 WRITE (5,10)
10 FORMAT (' Enter some text: ')
 READ (5,20) N,STRING
20 FORMAT (Q,A)
```

--- GOOD ---

```
 WRITE (5, * 'Enter some text: ')
 READ (5, '(Q,A)') N, STRING
```

Use spaces to align for readability.

--- BAD ---

```
X = 1.2
YX = 52.1

IF (X .GT. 12.) GO TO 10
IF (YZ .GT. 1.9) GO TO 20
```

--- GOOD ---

```
X = 1.2
YX = 52.1

IF (X .GT. 12.0) GO TO 10
IF (YZ .GT. 1.9) GO TO 20
```

Don't jump out of code block with GO TO.

--- BAD ---

```
C..... A block of code
.
.
GO TO 10
.
.
C..... Another block of code
10 .
.
.
```

--- GOOD ---

```
C..... A block of code
.
.
GO TO 10
.
.
10 CONTINUE
C..... Another block of code
.
.
```

Avoid bushy trees.

```

C..... SMALL = the smallest of X, Y, and Z
 IF (X .GE. Y) THEN
 IF (Y .GE. Z) THEN
 SMALL = Z
 ELSE
 SMALL = Y
 ENDIF
 ELSE
 IF (X .GE. Z) THEN
 SMALL = Z
 ELSE
 SMALL = X
 ENDIF
 ENDIF

```

Use indentations and blank lines to make code more readable.

The previous example without indentations and blanks look like

```

C..... SMALL = the smallest of X, Y, and Z
 IF (X .GE. Y) THEN
 IF (Y .GE. Z) THEN
 SMALL = Z
 ELSE
 SMALL = Y
 ENDIF
 ELSE
 IF (X .GE. Z) THEN
 SMALL = Z
 ELSE
 SMALL = X
 ENDIF
 ENDIF

```

Define symbolic names in PARAMETER statements.

```

Bad:
 IX = IPEEK ('176504'0)

Good:
 INTEGER*2 adac csr
 PARAMETER (adac csr = '176504'0)
 .
 .
 IX = IPEEK (adac csr)

```

Each DO-loop should terminate on a separate CONTINUE statement.

```

Bad:
 DO 10 I=1,10
 DO 10 J=1,10
10 X(I,J) = Y(I,J)

Good:
 DO 20 I=1,N
 DO 10 J=1,N
 X(I,J) = Y(I,J)
10 CONTINUE
20 CONTINUE

```

Don't GO TO an executable statement.

```

Bad:
 GO TO 10
 .
 .
10 DO 100 I=1,J
 .
 .

Good:
 GO TO 10
 .
 .
10 CONTINUE

 DO 100 I=1,J

```

Use good visual separators to divide logical blocks of code.

```

 --- BAD ---

C A BLOCK OF CODE
 X = 1.2
 Y = LOG (X)
 Z = X+Y

C ANOTHER BLOCK
 X = Y+Z
 Y = LOG(X)
 Z = 1.2

 --- GOOD ---

C..... A block of code
 X = 1.2
 Y = LOG (X)
 Z = X+Y

C..... Another block
 X = Y+Z
 Y = LOG(X)
 Z = 1.2

```

Use spaces in argument lists for readability and TABability.

```

 --- BAD ---

SUBROUTINE BAD EXAMPLE (X,Y,Z,N,FLAG)

 --- GOOD ---

SUBROUTINE GOOD EXAMPLE (X, Y, Z, N, FLAG)

```

Miscellaneous:

1. Begin main programs with PROGRAM name
2. Statement numbers should be in ascending order.
3. Avoid the DIMENSION statement. Use explicit type declarations instead.
4. An array in a COMMON block should have its dimensions declared in the COMMON statement.
5. Group EQUIVALENCE statements with the array declarations concerned.

6. Within a program module, the subprograms should be ordered alphanumerically.
7. Within a program unit the COMMON declarations should be ordered alphanumerically.
8. In mixed-mode expressions and assignments, the type conversions should be written explicitly.
9. Logical unit numbers should be symbolic constants.
10. Use RETURN in subroutines and functions only for alternate returns.

#### Commenting your code.

Should be a helpful tool after the code is written.

Should not look like the coder's shorthand.

Bad comments contribute to "Write-only" code.

Purpose: Help maintainers to quickly isolate problematic sections of code.

#### Test Question

"Do these comments tell the reader something he doesn't already know from reading the code itself?"

#### Commenting Techniques

Comments should be written to meet the needs of the audience.

Use spacing and format to improve readability.

Use spacing and format to show subordination.

Be consistent in presentation of comments.

#### More Techniques

Use a prose paragraph overview at the beginning of the module.

Use the overview to pull together the line-by-line comments.

Don't have your comments parrot the code.

Line comments should act as an access device to the code section.

#### Code Labels

Don't use undefined acronyms or abbreviations.

Let labels convey meaning.

USING AN LSI-11/23 AND RT-11  
TO DIGITIZE ANALOG TAPES

John N. Stewart  
Los Alamos National Laboratory  
Los Alamos, NM 87545

ABSTRACT

I describe in this paper the design and implementation of a system on an LSI-11/23 under RT-11 that digitizes 14-channel analog tapes at given intervals and writes the digitized data to a magnetic tape. The analog tapes contain 13 channels of data and one channel of IRIG-B time code. A given interval is selected by entering a time and an interval. The IRIG-B time code is read from the analog tapes and used to start the digitizing. A programmable clock board is used to control the digitizer and to count the interval length. In order to achieve the through-put rates needed, I wrote the code in MACRO and addressed the magnetic tape controller directly instead of using programmed requests.

THE PROJECT

This project started in the spring of 1984 when I was asked if there were any codes to digitize 14-track analog tapes. My reply was that there were none, but that I could probably write one if they were not in a big hurry. I had several LSI systems digitizing 24 hours a day and storing the data on magnetic tape, so I was familiar with the equipment and could program it. I asked a few questions to pinpoint exactly what was wanted. Essentially, there were several 14-track analog data tapes containing seismic events at various times. Of the 14 tracks, 13 were data and 1 was IRIG-B time code. The time and length of the events were known. The desired digitizing rate was 500 samples per second per channel. The digitized data would be written to magnetic tape.

The equipment situation looked good. I had another LSI system running RT-11 (SJ) which was used for development work and also included a magnetic tape unit, a digitizer board, and a programmable clock board. There was an IRIG-B time code reader available, and an analog tape playback unit could be found. Figure 1 shows how the system looked. I decided to write the software in MARCO to gain speed, simplicity, and memory over writing it in a high-level language.

The 16-channel digitizer that would be used could handle +/-10 volts, +/-5 volts, 0-10 volts, or 0-5 volts. It could do a digitization in 25 microseconds and had an automatic channel sequencer. I felt the actual digitizer loop in the code would take less than 100 microseconds which would allow a digitizing rate of better than 10,000 samples per second. Since I only needed 500 samples per second per channel for 14 channels or 7,000 samples per second I seemed in good shape. I would have the code digitize the selected channels of data as quickly as possible and then wait for the next clock interrupt rather than doing an equal interval type of digitizing. This would provide enough time between sampling intervals to

switch buffers and start the write tape process when a buffer filled.

There were no apparent problems with memory. The code would use two output buffers, each big enough to hold one second of data, which would give me plenty of time to write a buffer to magnetic tape while putting data into the other buffer. The size of each buffer would be 7,000 words (14,000 bytes), slightly large for a standard magnetic tape record but acceptable.

I did not see any problems writing the magnetic tape. The tape unit had a density of 1600 bytes per inch and moved at 75 inches per second. The code should be able to easily write 14,000 bytes in less than a second.

THE FIRST PROBLEM

The only analog tape playback unit that could be found forced playback at twice real time which



Figure 1. The Equipment.



meant the code had to digitize at twice the desired rate or 1000 samples per second per channel. That is a total of 14,000 samples per second (28,000 bytes). This meant the digitizer had to take a sample in less than 71 microseconds and also make available an extra slot of time every second to switch buffers and start writing out the old buffer to magnetic tape. Again, considering the 25 microsecond digitizer conversion cycle and needing only a few instructions to digitize a channel, I felt the speed up could be handled.

The amount of memory for the two 1 second buffers would double to 28,000 words (56,000 bytes). This would not leave enough memory for the code even though I would be programming in MACRO. I needed to go to smaller buffers and, hence, records; but I did not know the amount of time the tape unit used to start and stop when writing a record (record overhead time). I was also starting to worry about the volume of data the tape unit could handle in one second. I searched through the tape manual and found the information as shown in Table 1.

TABLE 1

TAPE TIMING (for 9 track, 75 IPS tape)

|                                                                  |   |                     |
|------------------------------------------------------------------|---|---------------------|
| Ramp time<br>(getting up to speed)                               | = | 5.000 milliseconds  |
| Write gap delay                                                  | = | 1.000 millisecond   |
| End-of-record<br>(3 blank frames, CRC,<br>3 blank frames, LLPCC) | = | .067 milliseconds   |
| Write stop delay                                                 | = | .300 milliseconds   |
| Ramp time<br>(slowing to a stop)                                 | = | 5.000 milliseconds  |
| Record overhead time                                             | = | 11.367 milliseconds |
| Time to write<br>1600 bytes (1 inch)                             | = | 13.333 milliseconds |
| Therefore, a 28,000 byte (1 second) record would take:           |   |                     |
| $(28,000/1600) * 13.333 + 11.367 = 233.333 + 11.367$             |   |                     |
| = 244.700 milliseconds.                                          |   |                     |

I had thought that tape operations were much slower and so I was pleasantly surprised with the above numbers. With the record overhead time being so short, I could go to a more reasonable record size of approximately 4000 bytes and the buffers would only total to approximately 8000 bytes. This would solve the memory problem.

A 4000 byte record would take 44.7 milliseconds to write. I decided to have the code calculate a record size based on the digitizing rate such that the record would contain more than 100 wall clock milliseconds of data. This would allow time to

switch buffers and start the tape writing between sampling intervals.

THE TIME CODE READER

The next consideration was how to start digitizing. At first I had planned to take the easy way out. I would have the user position the analog tape close to the desired time by looking at the IRIG-B time code reader display, and then have the user start the analog tape reading and press a key on the input terminal to signal the code to start digitizing. The IRIG-B time channel could be digitized to enable the user find the real time when the digital tape was processed. For one or two events this would do, but not for hundreds of events. I remembered someone saying there was a BCD output on the time code reader. I checked the manual and found that the time code reader had a BCD time output of 32 bits giving seconds through days. I only had one 16-bit parallel interface board so settled for 7 bits of seconds, 7 bits of minutes, and 2 bits of hours. The 16-bit BCD time code is shown here:

HOURS      MINUTES      SECONDS  
-- --xx    xxx    xxxx    xxx    xxxx

where x represents 1 bit.

For example: time 16:58:21 would be received as the following 16 bits

10 101 1000 010 0001  
( 2 5 8 2 1).

Allowing only 2 bits for hours meant the user would have to keep track of the hours. Table 2 shows the correspondence between the 2 bit hour and the actual hour.

TABLE 2

| HOUR BITS | 00 | 01 | 10 | 11 |
|-----------|----|----|----|----|
|           | 0  | 1  | 2  | 3  |
| HOUR      | 4  | 5  | 6  | 7  |
|           | 8  | 9  |    |    |
| COULD     | 10 | 11 | 12 | 13 |
|           | 14 | 15 | 16 | 17 |
| BE        | 18 | 19 |    |    |
|           | 20 | 21 | 22 | 23 |

Since the code would now have access to tape time, I decided to have the user set up a list of event times. The code would compare the event time with the analog tape time and, when found equal, would start digitizing. I would have the code convert the inputted time from ASCII to a BCD time so only one test needed to be done to check for the correct time.

Another problem occurred in that if the code was continuously looking for a time when the user was positioning the analog tape, a false time match could occur when changing analog tape speed to

fast forward and back to regular speed. To get around this I added a flag to each event time that if set to "P" would cause the program to pause until a character was entered at the terminal, otherwise the code would continue checking for the next time. This would allow the user to manipulate the analog tape without worrying about causing false time matches.

#### PUTTING IT TOGETHER

Having thought the project through, it was time to put the code together. The code was named FASDIG for the obvious reason that the code would be digitizing much faster than any of the other codes. What follows mostly reflects the existing state of FASDIG.

FASDIG uses the .GTLIN programmed request to read the input file. This was the only routine I could find to read a line of ASCII characters from a file. Therefore, the user is forced to run FASDIG from an indirect command file. The first line of the input file has to be "RUN FASDIG". The input data follows that. The first line of data contains 4 numbers: the number of channels, the number of .1 milliseconds (wall clock) per digitizing interval, the numerator and the denominator of the ratio of wall clock time to data tape time. Following that are up to seventeen 60-character lines for describing the tape and each of the channels being digitized. Then there is one line for each event consisting of the starting day, hour, minute, second of the event, number of seconds to keep, and the pause flag. The event time is in terms of analog tape time, the number of seconds to keep is in terms of wall clock time. I did not like mixing time bases but did not want to convert number of seconds from analog tape time to wall clock time which was needed for the digitizer. An example of an input file follows:

```
RUN FASDIG
 4,10,2,1, 4CHAN,1000S/SEC,RATIO=2
EXPERIMENT # 2032 S5 MEGA PUMP
CH1= EE-1 VERT (4V) PRECAMBRIAN
CH2= CEPT (4V) PRECAMBRIAN
CH3= BANC (4V) SURFACE
CH4= LAFK (4V) SURFACE
```

```
341,04,24,36,10,P,
341,04,34,22,10, ,
341,04,34,50,10,P,
```

The first action of FASDIG is to execute the SETUP routine. The routine prints a startup message as follows:

```
MUST RUN FROM INDIRECT FILE. WANT EXAMPLE (Y,N) ?
```

If the answer is a Y the routine then prints the startup message as shown in Figure 2.

After the startup message the routine calculates the address of the second buffer by adding half of the available memory to the first buffer address. Care is taken to keep the address at a word boundary.

The first data line is read in. Using the digitizing rate and the number of channels, a

record size is calculated such that it contains more than 100 milliseconds of data and the size is about 4000 bytes and less than the buffer size. For 1000 samples per second and 13 channels of data the size comes out as 4026 bytes and 143 milliseconds. Each record contains an 11 word header and each sampling interval contains the 16-bit BCD time from the time code reader at the start of this sampling interval in addition to the digitized data.

Various other quantities are calculated such as sample intervals per record and time per record. The number of .1 millisecond ticks per sampling interval is put into the clock buffer. When the clock is started, an interrupt will occur after that many ticks. Next, the comment or ID lines are read into a temporary array. The array will be part of the file header.

Finally, the event time lines are read and put into four arrays for the digitizing part of the code. Array 1 contains the ASCII day, hour, minute, and second for each event to search for. This ASCII time is put into the header of each event of the data output tape. Array 2 contains the BCD time to look for, converted from the times in Array 1. Array 3 contains the number of records to form and write for each event. This number is formed from the calculated record size and the number of seconds desired for an event. When digitizing, it is this number which is used to determine the end of an event rather than actually counting the seconds. Array 4 contains the pause flag: 0 for no pause or 1 to pause.

Once the input file has been completely read, a 177777 octal is placed in the BCD array after the last time. The SETUP routine then calls a tape initializing routine. This routine brings in the hardware tape driver and asks if the tape is a new or old tape. If it is an old tape, the routine will search for a double end-of-file and position the tape between the end-of-files; otherwise, it will rewind the tape. Then, it will write out the file header record. Next, the SETUP routine prints "READY TO DIGITIZE. START ANALOG. ENTER LETTER,CR ?" and now we return to the main program. A flow diagram of the SETUP routine is shown in Figure 3.

The rest of the code consists of 4 nested loops as follows:

```
Loop 1: inner-most = digitize all channels.
Loop 2: next = do number of sample intervals
 in a record.
Loop 3: next = do number of records in an
 event.
Loop 4: outer-most = do events until BCD time word
 is 177777 octal.
```

At the start of loop 4 the code continuously reads the 16-bit parallel interface board for the BCD analog tape time and compares it with the BCD event time. When a match occurs the clock, loops 1, 2, and 3 and the digitizer are started and the clock interrupt flag check is skipped. Thereafter, before starting loop 1, the clock interrupt flag is checked. If it is set, then the code is too slow for this digitizing rate and the code stops; otherwise, the code waits for the clock

USING AN EDITOR BUILD A FILE, FILENAME.COM AS FOLLOWS:

1. "RUN FASDIG" (DONOT INCLUDE ")
2. NUMBER CHANNELS, NUMBER .1MILLISEC/SAMPLE, NUMERATOR, DENOMINATOR OF RATIO WHERE RATIO=PLAYBACK SPEED TO RECORD SPEED
3. UP TO 17 LINES OF ID (60 CHAR EACH) IF < 17 LINES, END WITH CR ONLY LINE. LINE 1=GENERAL INFO, THEN ONE/CHAN WHERE 1ST NONBLANK CHAR AFTER = SIGN IS CHAN ID
4. UP TO 100 PICK TIME LINES (DAY,HR,MIN,SEC,# SEC TO KEEP, P OR NOTHING)  
NOTE: P IS FOR PAUSE AT END OF THIS EVENT  
NOTE: SEC TO KEEP ARE WALL CLOCK SECONDS.
5. END FILE WITH A CR ONLY LINE

TO RUN CODE - ENTER "@FILENAME" (DO NOT INCLUDE ")

DO YOU WANT TO CONTINUE (Y,N) ?

Figure 2. Startup Message

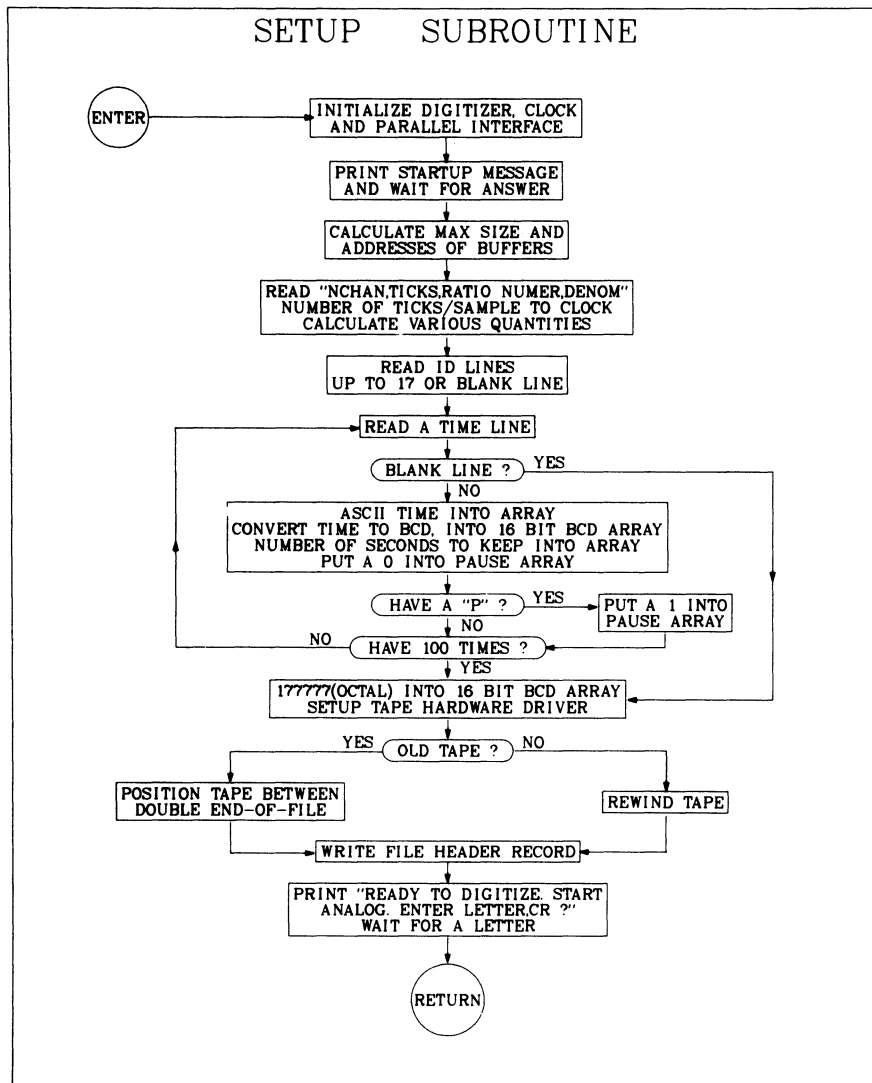


Figure 3. Flow Diagram of SETUP.

interrupt and then starts loop 1. After loop 1 finishes a check is made to determine if the buffer (record) full flag is set. If so, a write of buffer 2 to tape is started and the flag cleared. Loop 2 continues until the correct number of sampling intervals to make a record are in the buffer. After loop 2 finishes the buffer addresses are switched and the buffer full flag is set. Loop 3 continues until the desired number of records have been digitized. After the end of loop 3 the code stops the clock, writes the last record to tape, writes an end-of-file, and backspaces over the end-of-file. Next the code checks the pause flag. If the flag is set, the comment "AM IN PAUSE. ENTER LETTER,CR TO CONTINUE." is printed and the code waits for input from the terminal. If the flag is not set or input has been received, then loop 4 continues until an event time equal to 177777 (octal) is found which ends loop 4. The code does a little clean up and finishes. Figure 4 shows a flow diagram of FASDIG.

The clock interrupt handling routine decrements the clock interrupt flag, clears the interrupt bit, and returns from interrupt. The flag is reset by setting it to a 1. If the flag is ever negative, then more than one interrupt has

occurred since the last time the flag was reset. Although not mentioned previously the clock interrupt flag is checked for +, 0, and -.

#### MORE PROBLEMS

After putting the above code together and debugging it, I started to run the real case of 1000 sampling intervals per second (i.e. 10 ticks or ten .1 milliseconds per sampling interval). Lo and behold I got the message "CODE IS TOO SLOW." I tried several other values and found the code would digitize at 20 ticks but not at 18 ticks. I studied the listing of the code and squeezed out every unnecessary instruction I could find in the inner loops. I also rearranged the order of some instructions. I did everything imaginable to speed up the code, but to no avail.

At that time I was using the .WRITE programmed request to write the data out to tape. Since nothing else I had done speeded up the code, I decided to look at the tape controller manual to find out how easy it would be to control the tape unit directly instead of using the .WRITE programmed request. I found that the tape controller had several registers and only the command,

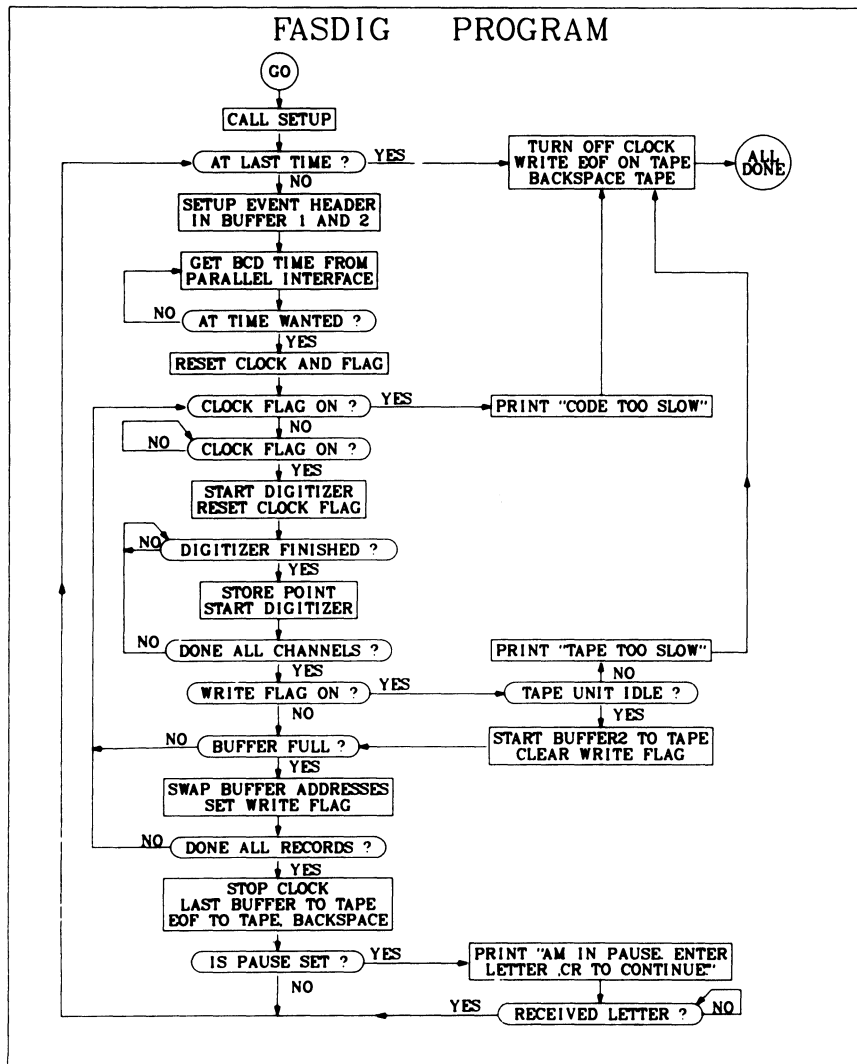


Figure 4. Flow Diagram of FASDIG.

address, and buffer count registers needed to be accessed in order to check activity on the tape drive and to have the controller start writing data to the tape. It seemed simple enough, so I replaced the following piece of coding

```
.WRITE #AREA,#1,@#BUF2,@#WDPREC
BCC 57$
.PRINT #WFAIL
JMP EXIT
57$: Reset write flag
```

with

```
TSTB @#MTCMD
BMI 57$
.PRINT #NOCTRL
JMP EXIT
57$: MOV @#BUF2,@#MTADR
MOV @#NEGBYT,@#MTBRC
MOVB #5,@#MTCMD
Reset write flag
```

where BUF2 contains the address of the data buffer to write out, WDPREC is the number of words to write out, NEGBYT is the negative number of bytes to write out, and MTCMD, MTADR, and MTBRC are the magnetic tape controller registers: command, address, and buffer count. WFAIL and NOCTRL point to two error comments. Thus, the .WRITE programmed request is replaced by three MOVE commands: two to tell what data to use and one to signal the controller to start writing. The error checking is slightly different in the two cases; in the old

case the carry bit was set to denote that the .WRITE was unable to queue the write request, in the new case the byte sign bit of the command register is cleared when the controller is busy. In either case, if the carry is set or the sign bit is cleared, the tape can not be written and the code is stopped. With this new change, the code had no trouble digitizing at a total rate of 13,000 samples per second and writing the 4026 byte tape records every 143 milliseconds. Figure 5 shows a sample of the data that was digitized.

### CONCLUSION

In the above project, I almost didn't try any method other than the .WRITE programmed request. I was worried about the conflicts that might occur with the system and the amount of effort fighting the system would take. However, I did try and the method I used did work with very little effort. My first conclusion is, therefore, don't give up before you try.

Secondly, I want to point out that the ease with which this change was made was largely do to running under the RT-11 operating system. I have tried to do similar projects under RSX-11M and usually have to spend months reading up on what data bases and packets must be built to accomplish what I want. Even then I would guess that the system will have a few "gotchas" that I overlooked. The simplicity of running under RT-11 is wonderful.

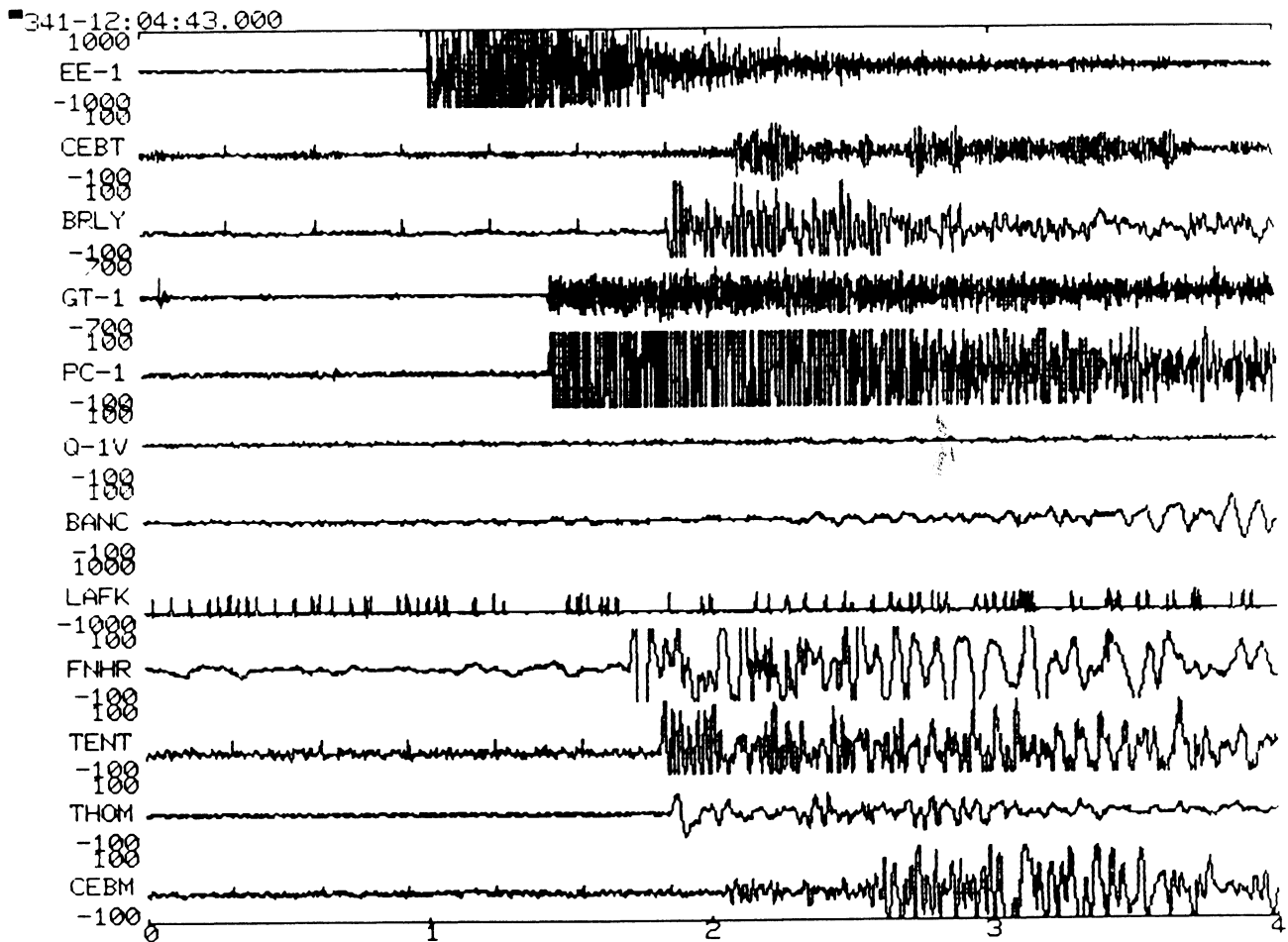


Figure 5. Sample of the Digitized Data.





LOADABLE DEVICE DRIVER DATA BASES  
IN RSX11M SYSGEN

Carl T. Mickelson  
Goodyear Aerospace Corporation  
Akron, Ohio 44315

ABSTRACT

In order to support multiple and differing configurations of peripherals on different computer systems, a method for modifying RSX11M SYSGEN is discussed that allows device drivers to be built with loadable, rather than resident, device data bases. System tailoring is done at VMR time when the particular devices for each configuration are included in the bootable system image. The method allows the addition of new devices without a new SYSGEN.

INTRODUCTION

What do you do when your OEM employer's business is growing and he acquires a room full of new computer systems for a variety of different customers? Each system has a different configuration of discs and terminal interfaces and you are expected to provide a consistent software development environment on all these different systems so that user's may build programs on any development system and move them to any test stand system to check out your product hardware. The users don't want to be confronted with different operating procedures on each machine and would like to move even privileged tasks, when possible, from system to system without re-building them. In addition, you don't want to be confronted with performing a new system generation for every new system as it arrives.

It would be much more convenient to maintain a system generated for each generic processor (11/34, 11/24, 11/40, 11/70) and memory configuration and simply build an operating system for a new computer by selecting an existing pre-generated system and specifying what devices are in the configuration at VMR time. Further, the tailoring of the terminal driver for different systems raises an additional complication for systems having the same processor but differing terminal I/O controllers.

This paper discusses a method of achieving this degree of hardware/software independence for RSX11M, and discusses how this goal to provide a common operating environment was achieved. It should be noted that privileged tasks that map over the Executive can still only be ported between systems with the same or similar processors and the same memory address range (18 bit or 22 bit). This represents an improvement over "vanilla" RSX11M since device configuration is no longer a consideration.

ISSUES NEEDING CONSIDERATION

The principal method needed to achieve the goals stated above is to modify the RSX11M SYSGEN procedure to make all DEC supplied device drivers look like user-written drivers, each task-built containing its own loadable device data base structure. By doing this, device data structures are no longer resident in the Executive task image

(RSX11M.TSK), and individual devices can be configured into a system by loading their data bases, together with their driver code at VMR time. In fact, the only device drivers that need to be present in the bootable system image are the boot device and terminal device drivers. All others can be loaded into GEN with a /HIGH switch in the system startup file. The ability to add additional devices is also enhanced, since a SYSGEN can now be avoided. There are, however, a number of issues that must be addressed in order for this technique to be successful.

First, all link time references to I/O data structure symbols must be eliminated, since the I/O data structures are no longer resident in the Executive. Without doing this, the link time references to .TTO and .COO in the Executive (SYSCM) and any privileged tasks such as the Pool Monitoring Task (PMT) and the Console Output Task (COT) will produce undefined references when they are task-built.

This requirement is satisfied by adding some code to the Executive module INITL.MAC to walk down the device DCB chain when the virgin executive is initially started to locate .TTO, the console terminal UCB address, and .COO, the console output UCB address. Once found, these addresses are loaded into the required fields of SYSCM, and RSX11M continues as if the data bases were resident. Similar code is added to PMT and COT at their entry points, and the references to these symbols are made indirectly to local copies of the addresses. This isolates these privileged tasks from the actual locations of the data structures.

Second, the Executive module SYSTB.MAC, generated by SYSGEN during the device configuration phase, must be split into individual files (xxTAB.MAC), one for each type of device selected for inclusion in the SYSGEN process. The only data structures remaining in SYSTB are the pseudo-device structures for SY:, LB:, etc, and the clock queue list-head. The device data base files must later be assembled with the RSX11M driver sources, so additional commands must be added to RSXDRVASM.COMD. Also, the driver task build command file, RSXDRVBLD.COMD, must be modified to task build each driver with its own local data base.



Finally, to support different configurations of terminal interfaces on systems with similar processors, a method must be found to build multiple versions of the terminal driver. Each version of the driver must have a loadable data base structured for that particular system so that the driver for the proper terminal configuration can be loaded at VMR time. It is assumed that the full-duplex terminal driver is the driver of choice for this effort as it provides full system functionality for the user.

#### DEVELOPING THE CHANGES

Once it is understood what must be done, a method must be developed to automatically apply the changes when a system is to be generated. This method should be applied in two parts.

The first part applies the needed changes to SYSCM and the privileged tasks to eliminate the link time references discussed earlier. These changes are required for the SYSGEN changes to be successful, but they can stand alone so that the tasks do not directly access I/O structures at link-time.

The second part of the modification package should apply the changes to the SYSGEN procedure files, and should only be applied if the first part changes are made. These modifications amend the SYSGEN procedure to achieve the functionality described earlier.

In order to develop the changes, a number of utilities are necessary, among them the standard utilities SLP, PIP, MAC, TKB, etc, and a DECUS supplied disassembler DOB. This utility is needed to disassemble certain modules, delivered only in object format so that they may be modified in source form, re-assembled and re-inserted into their appropriate object library.

A trial SYSGEN was performed, stopping at all end of section and end of execution breakpoints to study the state of the files produced by SYSGEN. The SYSGEN command files themselves were examined to determine where changes had to be made and a set of command procedure (.CMD) and source update (.SLP) files were developed to apply the changes automatically. The specific modifications made to the SYSGEN procedure are discussed below.

#### SYSGEN PROCEDURE CHANGES

The default option for all device drivers is set loadable, and this specific question is eliminated from the SYSGEN dialogue. The need for this is self-evident since the desired goal is to make a system with loadable device data bases. Also, the user is given the opportunity to include the Executive routines \$PTWRD and \$GTWRD regardless of whether user-written device drivers are to be included during the system generation. This is done to insure that these routines are available if a user-written driver will be incorporated at some future time.

Additional questions are added to the procedure to allow the user to establish an identifier (xxx) for each specific version of the terminal driver and system executive configuration being simultaneously generated for a common processor type. This identifier is appended to the terminal driver task

and symbol table file names, and to the Executive symbol table and bootable system file names to designate the system configuration pertinent to each set of files.

A tailored system VMR command file (SYSVMRxxx.CMD) is generated for each individual system configuration, and the user is given the opportunity to specify the device drivers that should be included in the system image prior to the first bootstrap. The balance of the VMR file is tailored to install only those privileged tasks needed for support of the devices loaded into the system image. Thus, if DUDRV is loaded, RCT is installed, and queue manager support is included only if LPDRV has been loaded.

Finally, just before VMR is invoked to create a bootable image, the user is permitted to select the version of the system and hence TTDRV version to be used to create the bootable system image for the SYSGEN host system.

The modified procedure, when completed through Phase II creates a set of virgin system (RSX11Mxxx.SYS) files that can be moved to another disc medium and together with the drivers and privileged tasks in [1,54], libraries in [1,1] and other system directories such as [1,2] can be used to VMR another system for a different hardware configuration. The resulting system can be booted on the new system configuration and has the same Executive and resident library characteristics as the originally VMR'ed system.

#### USING THE MODIFICATION KIT

The modifications described here are performed by a set of command procedures that will be available on the RSX SIG tape from the New Orleans DECUS. They should be used in accordance with the following procedure.

First copy the RSX11M baseline distribution kit to scratch media as described in the RSX11M SYSGEN reference manual. Then apply DEC's distributed Update D patch procedure to recreate the RSX11M baseline kit against which these modifications are performed. If the Able Computer Technology ENABLE Memory Management Unit (and Cache) is to be installed in your host system, apply the Able supplied patches next. Remember that any system built with the ENABLE hardware support cannot be moved to a system without the same hardware. This is a restriction of the Able patches, and not the changes discussed here. Finally, apply the procedure described in this paper by invoking the DECUSMODS.CMD command procedure.

This command procedure supports all three types of RSX11M distribution kit and will ask the user to specify which kit is in use. The procedure will then ask the user to re-build the indirect command processor (ICP) to enlarge its symbol table if a very large system configuration is being generated. This step should be taken, and the new ICP installed if the SYSGEN being performed will generate a system with a very large number of devices.

DECUSMODS should be re-invoked to apply the modifications described here. A set of hexadecimal modifications can be installed by this procedure if the user wishes. These hex modifications are

discussed in the companion paper "RSX11M Hexadecimal Command Line Numerics" presented at this DECUS symposium. The next set of changes that can be applied are the first part or I/O symbol linkage changes discussed earlier. These must be selected if the loadable driver data base changes are to be applied.

Each set of changes will be applied in sequence as the user selected. The procedures are designed to verify the modified files against expected results, but due to DECUS restrictions about copyrighted source code content on SIG tape submissions, not all the required files can be included. This should pose no problems, so long as Update D of V4.1 is used as the baseline for the procedure. After the modifications are performed, the normal SYSGEN procedure is initiated as described in the SYSGEN manual.

#### DIFFERENCES IN SYSGEN PROCEDURE

The following procedural changes should be observed when performing a SYSGEN with these changes applied.

If the system is autoconfigured, and the autoconfigure results are not overridden and amended, the resulting system will be restricted to supporting only the devices found by the autoconfigure program. This defeats a major purpose of using these changes, that is to generate a single system capable of supporting many different hardware configurations.

If autoconfigure is suppressed or overridden, when SYSGEN asks for the number of controllers of each device type in the system, the user should specify the controllers that comprise the union of all controllers in all the systems to be supported. Thus if three systems are to be supported on three different systems disc types, the controllers for each disc should be specified in the hardware configuration section. The number of each controller type to be generated should be specified as the maximum across all the systems to be supported. By following these two rules, SYSGEN will build all the device drivers necessary to support all the systems and will generate data bases to support the largest hardware configuration. Smaller systems will indicate off-line status for those devices not present at boot-time.

The same rules pertain to specifying the number of terminal controllers YL, YH, YZ, etc during the system hardware configuration. Later during the procedure, where each system's terminal driver is configured, the procedure will permit the user to specify the actual number of each controller type supported by a given system configuration. This is where each TTDRV data base is reduced in size to support only the specific terminal interfaces present in a system.

A word of caution applies here however. If the maximum number of any specific terminal interface class (YL, YH, YZ, etc) is greater than one, the minimum number that should be specified during this tailoring phase is either 0 (none) or 2 controllers for that class. This is particularly important for the YL class of single line controller. If this rule is not followed, the driver will assemble for a multi-controller system configuration, but the LOAD processor will create an Interrupt Transfer Block

(ITB) in POOL to support only a single controller. If this should happen, the generated Executive will CRASH ON THE FIRST KEYSTROKE after the virgin Executive is bootstrapped!

The SYSGEN procedure continues normally, configuring Executive, terminal driver and other system features. During the device configuration phase, where CSR and vector addresses are specified for each device controller, when the terminal devices are being configured, the modified procedure asks for the system identifier discussed earlier. The procedure then asks for the number of each type of terminal controller in the system. The procedure creates a terminals data base containing the currently specified subset of controllers and asks if another driver data base is to be built. If another is to be built, a new identifier and controller set is requested and the new data base is created. This process is continued until no more data bases are requested.

During this section, if the first system identifier specified is the NULL string, a standard single configuration system is built, including support for loadable data base device drivers. A non-null string causes a tailored terminals data base (TTTABxxx.MAC) to be created for the system.

Phase I of SYSGEN continues normally, assembling the Executive, drivers and individual driver data bases and preparing for Phase II task building. Chaining to Phase II continues the SYSGEN procedure where the system libraries, Executive and privileged tasks are built. It is during this phase that the modified procedure asks the user to re-specify the list of system identifiers used to designate the different terminal driver configurations. This list is used to create a virgin system boot file for each individual system configuration. In addition, the user is given the opportunity to specify a list of device drivers to be installed in each system. This list of drivers is used to tailor a SYSVMR file for each system configuration. At the end of this stage, the user is asked to specify the system identifier for the system on which the SYSGEN is being performed. The proper version of the terminal driver is selected for inclusion in the host's new system, and the system VMR process is executed to configure the new operating system for the host machine. When Phase II of SYSGEN ends, the new Executive is booted and saved with the bootstrap block written in the usual manner.

When the system re-boots itself, SYSGEN Phase III can be performed to re-build any needed utilities to use RESLIB, ANSLIB or other system features. After completing Phase III, and purging the system disc, the system is ready to be transported to another hardware configuration.

#### MOVING TO ANOTHER HARDWARE CONFIGURATION

To move the newly generated system to another hardware configuration, the following minimum set of directories and files should be copied to a new system disc device, ([1,1], [1,2], [1,54], [11,10]RSXMC.MAC).

Directory [1,1] contains the system object and macro libraries; [1,2] contains the help files and standard startup procedure files; and [1,54] contains the system privileged tasks and utilities.

RSXMC.MAC is included to provide definitions of the supported system features so that privileged tasks needing this information can be assembled correctly.

The correct version of the terminal driver task image and symbol table should be copied from TTDRVxxx.\* to TTDRV.\* and VMR executed using SYSVMRxxx.CMD to configure the bootable Executive for the new system. The new system image RSX11Mxxx.SYS should then be bootstrapped and saved with bootblock to create a system disc that is bootable on the new configuration.

This new system has exactly the same Executive, resident libraries, privileged tasks, and utilities as the original system. Further, neither system has suffered a loss of dynamic storage or POOL space due to unnecessary device data structures resident in the Executive. Also, the terminal driver has been tailored to support the specific kind and number of controllers on each hardware configuration.

#### ADDING A NEW DEVICE

Adding a new device to a system built with these changes is a relatively easy job. There are a number of different cases that need to be considered, but the job can typically be accomplished without the need of another full system generation.

If a device is to be added on an existing controller, the procedure is as follows. First, the data base source file xxTAB.MAC is edited, adding a new UCB structure to the file. The new UCB is added to the end of the existing UCB's in the file and is linked to the DCB and SCB already present. The highest unit number field of the DCB is incremented to enable the new unit.

The revised file is re-assembled with RSXMC.MAC and EXEMC.MLB to produce an object file for the new data base. The driver task and symbol table are re-created with task-building the new data base with the driver code object file. The new driver is placed in service by simply LOADING it at system startup if it was not installed by VMR. Otherwise, a copy of the virgin Executive, RSX11M.TSK, must be reinitialized with VMR and re-booted to install the new driver.

Installing new devices on a new controller is a similar procedure, except that a new SCB together with the UCB(s) must be added to the data base. Additional information on performing these steps can be found in the Guide to Writing a Device Driver in the sections describing how the I/O data structures are organized.

If a new type of device is to be added to the system and the system was originally generated anticipating the device's inclusion in the configuration, this task simply entails LOADING a driver that is already available. If a pre-built driver is not available, the data structure for a similar type device could be used as a model for the new device. However, this approach is difficult to accomplish successfully except in the simplest of cases for straight forward devices. The approach breaks down if the device UCB is extended to include device specific fields. In this case, the driver should be built during the SYSGEN procedure.

The techniques discussed here and embodied in the files of the modification kit permit multiple configuration systems to be generated simultaneously, with fully loadable device drivers and data structures. This permits a running system to be expanded easily and can eliminate the need to do an additional SYSGEN to make simple system configuration changes.

These techniques have been successfully used to generate systems supporting PDP 11/34, PDP 11/24, and PDP 11/40 UNIBUS processors. Systems with both 18 and 22 bit address support have been built from big disk, RK06/RK07 and RLO2 distribution kits. The following cautions and caveats are applicable when considering using these changes:

1. Controller CSR and vector addresses must be consistently set across all systems. SYSGEN'ing different types of controllers at the same address is acceptable as long as their device drivers are not LOADED into the same system image during VMR processing or system operation.
2. Support for multiple terminal driver configurations is provided for the full-duplex terminal driver only.
3. The changes apply to Update D of RSX11M V4.1. Proper application to other updates of V4.1 is not guaranteed. Since Update D added DEUNA driver support to RSX, and changes are made in building its data base, earlier Update kits are known to be incompatible with the changes.
4. Laboratory devices (A/D and D/A converters) and the IC series devices have not been addressed. Since the systems at the author's site do not use these types of interfaces, no changes have been designed for these devices during SYSGEN.
5. Any layered product that expects to find some I/O symbols defined in the Executive symbol table will not link successfully. This applies only to layered products that are built as privileged tasks as they are the only ones with the potential to access these I/O structures directly.
6. The compilers in use at the author's site have been built and used without modification. DECnet, however, has not yet been added to these systems, so it is not known if the changes will effect the NETGEN procedure. Inclusion of DECnet in these systems is planned for the near future.
7. While the changes made to the SYSGEN procedure do not depend upon the processors being UNIBUS based, no Q-bus machine configurations have been generated. It is not known if these changes will properly support a Q-bus based system.
8. Last, but not least, it must be remembered that systems generated using this procedure are transportable only between multiple hardware configurations that share the following characteristics:
  - a. same or similar processors; if built for EIS support, all processors must support EIS.
  - b. have memory management hardware; must be

mapped systems to provide the support necessary for loadable device drivers.

c. same memory address range; if built for 22 bit extended memory, all systems must support 22 bit addressing.

#### CAVEAT

The kit of files comprising the SYSGEN changes described here are supplied for information purposes only. Use of these changes at any PDP-11/RX11M site is entirely at the risk of the using organization. Neither the author, nor Goodyear Aerospace Corporation, nor DECUS, nor the RSX SIG is responsible if these changes do not perform successfully in any particular system.



RSX11M HEXADECIMAL COMMAND  
LINE NUMERICS

Carl T. Mickelson  
Goodyear Aerospace Corporation  
Akron, Ohio 44315

ABSTRACT

This paper describes a set of modifications to RSX11M SYSLIB and FCSRES to support the processing and display of numeric quantities as hexadecimal character strings. The modifications provide additional system subroutine entry points to allow application programs to parse hexadecimal quantities from command lines and to display numeric values as hexadecimal numbers when outputting data to the user.

BACKGROUND

Since RSX11M was first released, the operating system has been limited to representing numeric quantities on command lines and in program generated output as either octal or decimal character strings. Some programs such as DuMP provide a hexadecimal dump mode supported by routines within the program. However, RSX11M does not provide a means within SYSLIB for any program to "hear" or "speak" hex by way of a simple subroutine call.

The tools provided by RSX11M include a number of system library entry points to convert byte wide, word wide, and double-word wide data to and from octal and decimal character strings. Some of the applications developed at the author's site required a similar facility for processing MCR-like command switch parameters in hexadecimal.

The goal of this effort then was to develop a set of modifications to the numeric conversion routines in SYSLIB to provide a similar set of facilities for processing hexadecimal quantities as already exist for processing octal and decimal quantities.

ISSUES NEEDING CONSIDERATION

The code that processes octal and decimal numeric strings is contained in a number of modules in SYSLIB, the system object library. These routines also reside in the system resident library, if this option is selected during system generation.

The changes discussed here are changes to these modules that make the modules larger. Programs that need to use these new functions can simply link copies of the modules into their task images to do hexadecimal numeric conversions. A potential problem exists, however, if these modules are replaced in a resident library.

If these larger modules are included in a resident library on a single system, it could prevent tasks that do not use the new functions from being transferred between systems with different resident libraries because of differences in entry point addresses for the two library versions. The solution to this problem was introduced in an Update to RSX11M V4.1. The entry points into the resident library were vectored into the code portion of the library using a technique that is similar to the way

VMS vectors calls to system functions through a transfer vector.

The transfer vector consists of a JMP instruction to each entry point within the resident library task. Any task linking to a library routine transfers control to one of the JMP instructions which transfers control to the local entry within the library. By keeping the order of the JMP instructions, and hence the library entries, fixed within the transfer vector, the internal library modules can be changed without effecting a task's ability to find entries in the modified module. Tasks that are linked to a library containing modules supporting the new hexadecimal functions, but that make no use of them, can successfully be moved to a system with a resident library that does not support the hex conversions. All that is required is that the order of the transfer vector entry points are preserved between the library versions. The module entry point addresses within the library are different between the two versions, but are important only within the context of the library task.

NUMERIC CONVERSION TOOLS IN SYSLIB

Table 1 summarizes the numeric conversion facilities supplied with the standard RSX11M distribution. These routines are documented in chapters 4, 5, and 6 of the RSX11M System Library Routines Reference Manual. There are entry points available for input conversions from octal and decimal ASCII digit character strings to internal binary, and output conversion entries for translating from internal binary to octal or decimal ASCII digit character strings. In addition, a generalized formatting subroutine is provided that permits a MACRO programmer to create character oriented output records from a buffer of binary data. The record format is determined by a control string that is interpreted by the subroutine in a manner analogous to a FORTRAN FORMAT statement. The subroutine supports both octal and decimal format specifiers.

The new hexadecimal functions added to these SYSLIB modules are summarized in Table 2. The changes include support for both input and output conversions using a hexadecimal radix. In addition, format descriptors are added to the formatting

subroutine to support hexadecimal output. The calling sequences for these new functions are the same as the calling sequences for the existing functions. Detailed documentation on the use of the new functions is included as an appendix to this paper.

ASCII to Binary Conversions

| Radix   | Data Length |             |
|---------|-------------|-------------|
|         | Word        | Double-Word |
| Octal   | \$COTB      | .OD2CT      |
| Decimal | \$CDTB      | .DD2CT      |

Binary to ASCII Conversions

| Radix                              | Data Length |                |
|------------------------------------|-------------|----------------|
|                                    | Word        | Double-Word    |
| Octal<br>(Magnitude)<br>(Signed)   | \$CBOMG     | \$CBTMG (byte) |
|                                    | \$CBOSG     |                |
| Decimal<br>(Magnitude)<br>(Signed) | \$CBDMG     | \$CDDMG        |
|                                    | \$CBDSG     |                |

Generalized Formatting Descriptors (\$EDMSG)

|    |                                                         |
|----|---------------------------------------------------------|
| %B | Binary Byte to Octal String                             |
| %D | Binary Word to Signed Decimal String                    |
| %M | Binary Word to Decimal Magnitude String                 |
| %O | Binary Word to Signed Octal String                      |
| %P | Binary Word to Leading Zero Octal Magnitude String      |
| %Q | Binary Word to Zero Suppressed Octal Magnitude String   |
| %T | Binary Double Word to Decimal Magnitude String          |
| %U | Binary Word to Zero Suppressed Decimal Magnitude String |

Table 1 - Standard RSX11M Numeric Conversions

ASCII to Binary Conversions

| Radix       | Data Length |             |
|-------------|-------------|-------------|
|             | Word        | Double-Word |
| Hexadecimal | \$CHTB      | .HD2CT      |

Binary to ASCII Conversions

| Radix                                  | Data Length |         |
|----------------------------------------|-------------|---------|
|                                        | Word        | Byte    |
| Hexadecimal<br>(Magnitude)<br>(Signed) | \$CBHMG     | \$CBGMG |
|                                        | \$CBHSG     |         |

Generalized Formatting Descriptors (\$EDMSG)

|    |                                             |
|----|---------------------------------------------|
| %G | Binary Byte to Hexadecimal String           |
| %H | Binary Word to Hexadecimal Magnitude String |
| %J | Binary Word to Signed Hexadecimal String    |

Table 2 - Hexadecimal Numeric Conversions

There are six modules in SYSLIB that are changed to support the hexadecimal functions described earlier. The modules CATB, CBTA, .ODCVT and OD2CT have additional instructions inserted to perform the actual conversions of character strings to and from binary. The EDMSG and .TPARS modules are modified to make reference to the new function entry points when hexadecimal numbers are to be converted.

The changes to these modules are designed and implemented as source language update (SLP) correction files to be applied to a source file for the module. The sources for the modules are generated by an object code disassembler. Once the modules are changed, the updated source files are re-assembled and replaced into the system library object file. Any program needing access to these functions can link the revised modules into the task image to use the new functions. In addition to the changes made to these SYSLIB modules, two system macro definitions, CSI\$SV and ISTAT\$, are enhanced to allow a MACRO programmer to access these hexadecimal conversions from the Command String Interpreter, CS11 and CS12, and the Table Driven Parser, TPARS.

The only remaining detail is to include the new functionality in any resident library that is built during SYSGEN. The resident library built during the SYSGEN procedure is approximately 6 Kwords long, and is organized into two overlay sections so that the entire library can be accessed using only one 4 Kword APR. During the SYSGEN procedure, the resident library symbol table is created from the object modules FCSST1 and FCSST2. The library task includes the modules FCSLB1 and FCSLB2, each linked into the beginning of an overlay segment. The FCSLBx modules contain the library entry point transfer vectors, while the FCSSTx modules define the externally available entry point addresses to each transfer vector in the library. It is critical that the order of the entries in the FCSSTx and FCSLBx modules are and remain the same. To maintain compatibility with other libraries, any new entry points must be added at the end of the transfer vector list. The new hexadecimal support entry points are added to the end of the DEC specified transfer vector list, leaving some empty vectors available for additional DEC supplied functions in a future release of RSX11M.

The modules FCSSTx and FCSLBx are disassembled, changed with a SLP correction file, re-assembled and replaced into SYSLIB. The resident library task build driver file, FCSRS1BLD.BLD, is changed to include the new entry points in the overlay symbol table. Finally, the modified resident library is built by the SYSGEN procedure in the normal fashion.

USING THE MODIFICATION KIT

The modifications described here are performed by a set of command procedures that will be available on the RSX SIG tape from the New Orleans DECUS. They should be used in accordance with the following procedure.

First copy the RSX11M baseline distribution kit to scratch media as described in the RSX11M SYSGEN reference manual. Then apply DEC's distributed Update D patch procedure to recreate the RSX11M

baseline kit against which these modifications are performed. Finally, apply the procedure described in this paper by invoking the DECUSMODS.COMD command procedure.

This command procedure supports all three types of RSX11M distribution kit and will ask the user to specify which kit is in use. The indirect command processor re-build step can be skipped if the SYSGEN procedural modifications described in the companion paper "Loadable Device Driver Data Bases in RSX11M SYSGEN" are not being performed. Application of the HEXMOD command procedure should be selected to install the functionality described in this paper.

The procedure is designed to verify the modified files against expected results, but due to DECUS restrictions about copyrighted source code content on SIG tape submissions, not all the required files can be included. This should pose no problems, so long as Update D of V4.1 is used as the baseline for the procedure. After the modifications are performed, the normal SYSGEN procedure is initiated as described in the SYSGEN manual.

#### CONCLUSIONS AND COMMENTS

The techniques discussed here and embodied in the files of the modification kit permit support to be provided for processing and generating hexadecimal numerics in PDP-11/RSX11M programs.

These techniques have been successfully used on PDP 11/34, PDP 11/24, and PDP 11/40 processors. However the following comments and cautions are applicable when considering using these changes.

1. The changes apply to Update D of RSX11M V4.1. Proper application to other updates of V4.1 is not guaranteed.
2. The extra code added to certain SYSLIB modules to support hexadecimal numerics may make large programs that include copies of the modules within their task images too large to be built successfully. Tasks mapping a resident library should not be affected by this size increase. The changes do not make the library bigger than the single APR already required to map the resident library code.

#### CAVEAT

The kit of files comprising the hexadecimal changes described here are supplied for information purposes only. Use of these changes at any PDP-11/RSX11M site is entirely at the risk of the using organization. Neither the author, nor Goodyear Aerospace Corporation, nor DECUS, nor the RSX SIG is responsible if these changes do not perform successfully in any particular system.

#### APPENDIX

The following information provides documentation on using the hexadecimal conversion functions added to SYSLIB and described in this paper. The structure of this documentation is similar to that appearing in Chapters 4, 5, and 6 of the RSX11M System Subroutine Library Reference Manual, and should be regarded as a supplement to that manual.

The changes made to the macro definitions CSI\$SV and ISTAT\$ are also documented here. This information supplements data appearing in the RSX11M IO Operations Reference Manual. The details of using the macros are explained in Chapters 6 and 7. Only the changes made to the macros, and the additional functionality these changes provide is discussed here.

#### Hexadecimal to Binary Double-word Routine (.HD2CT)

The .HD2CT routine converts an ASCII hexadecimal string to a double length (two-word) binary number.

The routine accepts leading plus (+) or minus (-) signs, and the numbers 0 - F. A preceding ! symbol is legal in the hexadecimal number string. A ! symbol and a period in the same string is invalid. A trailing decimal point will be accepted in the input string, but will cause the decimal, rather than hexadecimal, radix to be used. This condition exists because the .HD2CT routine is an entry point in the .DD2CT routine, which converts decimal number strings to binary double-word values.

Any other characters in the ASCII hexadecimal number string will cause the .HD2CT routine to terminate the conversion procedure.

The value range of a hexadecimal number to be converted is  $-2^{31}$  to  $+(2^{31} - 1)$ .

To call the .HD2CT routine:

1. Supply three input arguments in the task's source code:
  - a. in Register 3, the address of the two-word field in which the converted number is to be stored.
  - b. in Register 4, the number of characters in the string to be converted.
  - c. in Register 5, the address of the character string to be converted.

2. Include the statement

```
CALL .HD2CT
```

in the source program.

The .HD2CT routine saves and restores all the calling task's registers. Outputs from the .HD2CT routine are:

1. The converted number, where the high order 16 bits are stored in word 1 of the field specified in R3 input, and the low order 16 bits are stored in word 2 of the field.

2. Condition Code:

C bit = clear if conversion was successful.

C bit = set if an illegal character was found and conversion was incomplete.

The user can determine, in the task, whether conversion was complete by testing the C bit in the Condition Code.



### Hexadecimal to Binary Conversion Routine (\$CHTB)

The \$CHTB routine converts an ASCII hexadecimal number to binary. Valid characters in the number to be converted are 0 - F.

The end of the string must be marked by a terminating character, which may be any ASCII character except the numbers 0 - F. Examples of terminating characters are: blank, a tab character, an alphabetic character other than A - F, and a special symbol. Leading blanks and tab characters are ignored.

The maximum value of a hexadecimal number that can be converted by the \$CHTB routine is FFFF.

To call the \$CHTB routine:

1. Input, in Register 0, the address of the first byte of the ASCII characters to be converted.

2. Include the statement

```
CALL $CHTB
```

in the source program.

The \$CHTB routine calls the \$SAVRG routine to save and restore registers 3 - 5 of the calling task.

The outputs returned from the \$CHTB routine are:

1. R0 = the address of the next byte in the input buffer.

2. R1 = the converted number.

3. R2 = the terminating character.

The user can determine, in the task, whether an input string was successfully converted by testing the content of R2. If the content is other than the expected terminating character, the conversion was incomplete, since some other invalid character was found in the input string.

Successive input string conversion may be effected by setting up a processing loop to repetitively call \$CHTB.

### BINARY TO HEXADECIMAL CONVERSION

There are three system library routines that convert internally formatted binary numbers to external ASCII hexadecimal format:

1. Convert Binary to Hexadecimal Magnitude Routine (\$CBHMG), which converts an internally stored binary number to a 4-digit unsigned ASCII hexadecimal magnitude number.

2. Convert Binary to Signed Hexadecimal (\$CBHSG), which converts an internally stored binary number to a 4-digit signed ASCII hexadecimal number.

3. Convert Binary Byte to Hexadecimal Magnitude Routine (\$CBGMG), which converts an internally stored binary byte to a 2-digit ASCII hexadecimal number.

These routines use predefined parameters that are

passed to the general purpose conversion routine (\$CBTA), which performs the actual binary to ASCII conversion.

### Convert Binary to Hex Magnitude Routine (\$CBHMG)

The \$CBHMG routine converts an internally stored binary number to a 4-digit unsigned ASCII hexadecimal magnitude number.

The \$CBHMG routine uses the following predefined conversion parameters:

```
radix = 16.
field width = 4. characters
sign flag = UNSIGNED
leading zeroes flag = NOSUP (no suppression)
```

To call the \$CBHMG routine:

1. Supply three input arguments in the task's source code:

- a. in Register 0, the starting address of the output area in which the converted 4-digit number is to be stored.

- b. in Register 1, the binary number to be converted.

- c. in Register 2, the zero suppression indicator, where:

R2 = 0 to specify suppression of leading zeroes in the converted number. The output number will be left-justified.

R2 = nonzero to specify that leading zeroes are not to be suppressed.

2. Include the statement

```
CALL $CBHMG
```

in the source program.

The predefined conversion parameters are automatically pushed to the stack on entry to the \$CBHMG routine. If the user specifies, via R2 = 0, that leading zeroes are to be suppressed, the NOSUP parameter is reset. In any case, the \$CBHMG routine passes the parameters in Register 2 to the General Purpose Binary to ASCII Conversion Routine (\$CBTA), which performs the actual conversion of the binary number.

The \$CBTA routine calls the \$SAVRG routine to save and restore registers 3 - 5 of the calling task. Registers 1 and 2 are destroyed.

Outputs from the \$CBHMG routine are:

1. The converted number, a maximum of four digits in length, in the specified output area.

2. R0 = the next available address in the output area (the pointer to the location following the last digit stored).

The \$CBHMG routine does not return error conditions to the caller.

### Convert Binary to Signed Hex Routine (\$CBHSG)

The \$CBHSG routine converts an internally stored binary number to a 4-digit signed ASCII hexadecimal number.

The \$CBHSG routine uses the following predefined conversion parameters:

```
radix = 16.
field width = 4. characters
sign flag = SIGNED
leading zeroes flag = NOSUP (no suppression)
```

To call the \$CBHSG routine:

1. Supply three input arguments in the task's source code:

a. in Register 0, the starting address of the output area in which the converted 4-digit number is to be stored.

b. in Register 1, the binary number to be converted.

c. in Register 2, the zero suppression indicator, where:

R2 = 0 to specify suppression of leading zeroes in the converted number. The output number will be left-justified.

R2 = nonzero to specify that leading zeroes are not to be suppressed.

2. Include the statement

```
CALL $CBHSG
```

in the source program.

The predefined conversion parameters are automatically pushed to the stack on entry to the \$CBHSG routine. If the user specifies, via R2 = 0, that leading zeroes are to be suppressed, the NOSUP parameter is reset. In any case, the \$CBHSG routine passes the parameters in Register 2 to the General Purpose Binary to ASCII Conversion Routine (\$CBTA), which performs the actual conversion of the binary number.

The \$CBTA routine calls the \$SAVRG routine to save and restore registers 3 - 5 of the calling task. Registers 1 and 2 are destroyed.

Outputs from the \$CBHSG routine are:

1. The converted number, a maximum of four digits in length, in the specified output area.

2. R0 = the next available address in the output area (the pointer to the location following the last digit stored).

The \$CBHSG routine does not return error conditions to the caller.

### Convert Binary Byte to Hex Magnitude Routine (\$CBGMG)

The \$CBGMG routine converts an internally stored binary number to a 2-digit ASCII hexadecimal number.

The \$CBGMG routine uses the following predefined conversion parameters:

```
radix = 16.
field width = 2. characters
sign flag = UNSIGNED
leading zeroes flag = NOSUP (no suppression)
```

To call the \$CBGMG routine:

1. Supply three input arguments in the task's source code:

a. in Register 0, the starting address of the output area in which the converted 2-digit number is to be stored.

b. in Register 1, the binary byte to be converted in the low order byte.

c. in Register 2, the zero suppression indicator, where:

R2 = 0 to specify suppression of leading zeroes in the converted number. The output number will be left-justified.

R2 = nonzero to specify that leading zeroes are not to be suppressed.

2. Include the statement

```
CALL $CBGMG
```

in the source program.

The predefined conversion parameters are automatically pushed to the stack on entry to the \$CBGMG routine. If the user specifies, via R2 = 0, that leading zeroes are to be suppressed, the NOSUP parameter is reset. In any case, the \$CBGMG routine passes the parameters in Register 2 to the General Purpose Binary to ASCII Conversion Routine (\$CBTA), which performs the actual conversion of the binary byte.

The \$CBTA routine calls the \$SAVRG routine to save and restore registers 3 - 5 of the calling task. Registers 1 and 2 are destroyed.

Outputs from the \$CBGMG routine are:

1. The converted number, a maximum of two digits in length, in the specified output area.

2. R0 = the next available address in the output area (the pointer to the location following the last digit stored).

3. R1 = low order byte is unchanged; high order byte is cleared by the \$CBGMG routine.

The \$CBGMG routine does not return error conditions to the caller.

### GENERALIZED FORMATTING

Generalized output formatting is provided by the Edit Message Routine (\$EDMSG). The changes

described in this paper have added three types of output format conversions to the capabilities of this routine. These format conversions are documented here. The terms used in explaining these conversions have the same meaning as that used in Table 6-1 in the RSX11M System Library Routine Reference Manual.

| Directive                                         | Form | Operation                                                                                                                                                                                                 |
|---------------------------------------------------|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| G<br>(binary byte to hexadecimal conv)            | %G   | Convert the next binary byte address in ARGBLK to hexadecimal and store result in OUTBLK.                                                                                                                 |
|                                                   | %nG  | Convert the next n binary bytes from address in ARGBLK to hexadecimal numbers, and store results in OUTBLK; insert space between numbers.                                                                 |
|                                                   | %VG  | Use the value in the next word in ARGBLK as repeat count, convert the specified number of binary bytes from address in ARGBLK to hexadecimal numbers, and store results in OUTBLK; space between numbers. |
| H<br>(binary to hexadecimal magnitude conversion) | %H   | Convert the binary value in the next word in ARGBLK to hexadecimal magnitude and store result in OUTBLK.                                                                                                  |
|                                                   | %nH  | Convert the next n binary values in ARGBLK to hexadecimal magnitude and store results in OUTBLK; insert tab between numbers.                                                                              |
|                                                   | %VH  | Use the value in the next word in ARGBLK as repeat count, convert the specified number of binary values to hexadecimal magnitude, and store the results in OUTBLK; insert tab between numbers.            |
| J<br>(binary to signed hexadecimal conversion)    | %J   | Convert the binary value in the next word in ARGBLK to signed hexadecimal and store the result in OUTBLK.                                                                                                 |
|                                                   | %nJ  | Convert the next n binary values in ARGBLK to signed hexadecimal and store the results in OUTBLK; insert tab between numbers.                                                                             |
|                                                   | %VJ  | Use the value in the next word in ARGBLK as repeat count, convert the specified number of binary values to hexadecimal and store the results in OUTBLK; insert tab between numbers.                       |

## COMMAND STRING INTERPRETER MACRO CHANGES

The use of the Command String Interpreter switch value macro CSI\$SV enhancement is explained below. This information supplements the explanation of the 'type' parameter to the macro. See the RSX11M IO Operations Reference Manual, section 6.2.4.2 for more information.

The conversion-type argument can take on the value:

HEX - Indicating that a numeric switch value is to be converted to binary using hexadecimal as a default conversion radix.

On the next page of the referenced manual the last two paragraphs describing numeric conversions should be replaced with the following:

On numeric conversions, the default conversion type specified for a switch value can be overridden by means of a pound sign (#), an exclamation point (!), or a dot (.). A numeric value preceded by a pound sign (for example, #10) forces the conversion type to octal; a numeric value preceded by an exclamation point (for example, !10) forces the conversion type to hexadecimal; a numeric value followed by a dot (for example, 10.) forces the conversion type to decimal. Note also that a numeric value can be preceded by a plus sign (+) or a minus sign (-). The plus sign is the default assumption. If an explicit octal or hexadecimal value is specified using the pound sign (#) or exclamation point (!), as described above, the arithmetic sign indicator (+ or -), if included, must precede the pound sign or exclamation point (for example -#10).

If the conversion type is decimal, the switch value is evaluated as a single number; an overflow into the high-order bit (bit 15) results in an error condition. However, if the conversion type is octal or hexadecimal, a full 16-bit value may be specified.

## TABLE DRIVER PARSER MACRO CHANGES

The use of the Table Driven Parser has been enhanced to include two state transitions based on a hexadecimal digit or a hexadecimal number. This information supplements sections 7.1.2 and 7.2.1 of the RSX11M IO Operations Reference Manual.

The following two state transitions are defined for use in the 'type' parameter to the TRAN\$ macro:

\$HXDIG - Matches any single hexadecimal digit (0 - F).

\$HXNUM - Matches a hexadecimal number. Such a number consists of hexadecimal digits, followed optionally by a period. If number is not followed by a period, it is interpreted as hexadecimal. Numbers followed by a period are interpreted as decimal and the decimal point is included in the matching string. A number is terminated by any non-hexadecimal character. Values through 2<sup>32</sup> - 1 are converted to 32-bit unsigned integers.

The transitions in a state may represent several syntax types; a portion of a string being scanned often matches more than one syntax type. Therefore, the order in which the types are entered in the state table is critical. Transitions are always scanned in the order in which they are entered and the first transition matching a string being scanned is the transition taken. Therefore, the following order is recommended for states containing more than one syntax type:

- char
- keyword
- \$EOS
- \$HXDIG
- \$ALPHA
- \$DIGIT
- \$BLANK
- \$HXNUM
- \$NUMBER
- \$DNUMB
- \$STRNG
- \$RAD50
- \$ANY
- \$LAMDA

Placement of !label transitions in a state depends on the types and positions of other syntax types in the state as well as on the syntax types in the starting state of the subexpression.



# A REAL-TIME MULTIPROCESSOR DATA ACQUISITION NETWORK

Mark Podany  
James M. Galm  
Francis L. Merat  
Department of Electrical Engineering  
Case Institute of Technology  
Case Western Reserve University  
Cleveland, Ohio 44106

A multiprocessor, real time data acquisition system has been developed for use in an ongoing meteorological study. Using custom designed modules, the system features high speed fiber optic data paths, a distributed network architecture and precise micrometeorological instruments. A PDP-11/24, running RSX11-M acts as system coordinator and mass storage structure.

## INTRODUCTION

Recent studies have indicated a need for millimeter wavelength propagation measurements correlated with changes in the complex index of refraction of the atmosphere [1]. The atmospheric (complex) index of refraction is responsible for absorption and dispersion at millimeter wavelengths and is a spatial function of the atmospheric temperature and humidity.

The requirements for a millimeter wavelength propagation experiment are, thus, a means of measuring the millimeter wavelength propagation characteristics while simultaneously measuring the (micro) meteorological parameters of the atmosphere. This paper will not address the measurement of propagation parameters themselves [4], but will address the design of a network for the collection and processing of acquired meteorological data within the International Standards Organization's Open System Interconnect model.

The meteorological data acquisition system (MDAS) is a distributed multiprocessor network operating in a master/slave relationship with a host computer. (Detailed description of the hardware and software aspects of MDAS can be found in Reference [2].) MDAS utilizes the concept of minimum hardware implementation with maximum software flexibility to accommodate experimental changes. For MDAS this resulted in the use of microprocessors with integral data communications features, having minimal memory and user defined peripherals. A modified ISO-OSI protocol is used for the network controlling protocol [3]. System flexibility is attained by using the host

computer to dynamically reprogram the distributed system; thus, the logical architecture of the system is of a multidrop nature rather than a star or ring network which can be quickly changed to meet new data acquisition or control requirements with no hardware changes.

The physical layout of MDAS has been dictated by an ongoing propagation experiment [4]. For the experiment referred to, six (6) meteorological measurement sites (or towers) are equally spaced at 320 meter intervals over a 1.6 Km linear distance. The host computer provides overall system control and data display and storage; the network provides data communications hardware, controlling software and data communications protocols; the towers provide measurements of such atmospheric parameters as static and dynamic temperature, humidity, and wind velocity.

## DESCRIPTION OF MDAS HARDWARE

A schematic diagram of the system may be found in Figure 1. The host computer is a Digital Equipment Corporation PDP-11/24 running RSX-11M with disk and tape data storage. The PDP-11 is responsible for the overall control of the network, the system clock (a KW11-K real time clock), data storage, and interaction with the human operator. This latter includes selection of executable programs, data display, and display of system status. A DR11-C bidirectional parallel interface connects the host computer to a dynamically reprogrammable, microprocessor based front end processor (FEP). This interface handles the details regarding communications to and from the towers. Communications between towers is done with fiber optic data paths. This imposes some restraints on possible tower interconnection schemes because of the one-way communications nature of most optical fiber systems. Commercial fiber-optic transmitters, receivers, and cable were used

throughout the system[5]. From a logical standpoint, various ways of interconnecting the towers could have been devised. In MDAS, two fiber optic cables - one carrying information away from the FEP and one carrying data to the FEP ( see Figure 1 ) - connect the towers to the FEP. Each tower actively receives and retransmits all information on the two fiber optic channels with hardware to logically determine the towers' interconnections under software control. A logical diagram of the possible routing paths at each tower is shown in Figure 2.

The Intel 8031 microprocessor was chosen to implement the communications and control functions of each tower. This microprocessor was chosen primarily because of the 8031's on-board full duplex UART with special ninth-bit mode. As will be discussed in the next section this ninth-bit mode permitted a very simple command format. The 8031 is a high speed (12 MHz clock) microprocessor with a control-oriented architecture, ideally suited for data acquisition and control applications.

The FEP is an 8031 system similar in form to those in the towers. The FEP includes extended memory and a parallel sixteen-bit interface to the host computer. Each tower contains a snapshot 8-channel A/D, i.e. all data acquisition occurs on all eight channels simultaneously. The snapshot A/D is a standard 8 channel A/D preceded by eight sample/hold amplifiers. A CONVERSION command causes all sample/hold amplifiers to enter the hold mode until all data conversion is complete. The maximum network latency time (time between transmission of a command and the receipt of an acknowledgement to that command) permissible between towers for the propagation experiment is approximately 100 microseconds. In MDAS, a CONVERSION command sent to all towers simultaneously yields an overall maximum latency (due to a combination of transmission delays and skewing of the individual microprocessor clocks) between any two channels on any two towers within the above maximum. The system is capable of simultaneously acquiring 128 (16 towers x 8 channels/tower) channels of analog information and transmitting the data to the host computer at a rate on the order of 250Hz. A maximum sampling speed of 6 KHz can be attained with the system collecting from a single analog channel on one tower.

#### DESCRIPTION OF MDAS SOFTWARE

The philosophy of software flexibility required the software to be organized in a modular fashion, permitting the software to evolve as the experiment grows or changes. A protocol based on the ISO-OSI model as interpreted by Tanenbaum (Figure 3) provides the necessary modularity [6]. Tanenbaum's model has been somewhat altered to include the master-slave relationship between the FEP and the towers. In the control protocol "local" refers to the FEP (the master), and "remote" to the towers (slaves). The

ISO-OSI model contains seven layers of protocol interpreted as follows (also refer to Figure 4):

1. APPLICATION LAYER - Responsible for obtaining and transferring necessary files to run network and collect real time data. This is done by using a friendly human interface. The human interface enables a user not familiar with the network to easily set up data collection from various instruments on the towers at any rate he chooses (in increments of .01 seconds) or use a previously saved experiment. By being able to save the setup of the system a knowledgeable user once having saved an experiment can send anyone out to run it again. The human interface also enables someone who knows how the network works to be able to control the network from the Session layer (the lowest layer directly attainable to the user).

2. PRESENTATION LAYER - This layer contains the implementation of frequently used subroutines called by the Application layer i.e. tower diagnostics, sensor checks, logical reconfiguration of link, and link commands.

3. SESSION LAYER - This is the lowest user interface into the network. The software executes in the PDP-11 and controls the DR11-C, the data communication between the host computer and the FEP. If the user was in this layer he would be able to have direct control, through the software, over the FEP and all the remote towers.

4. TRANSPORT LAYER - The FEP software makes up this layer. The FEP is responsible for the control of the subnet, synchronization of data acquisition, packaging and formatting data and encoding commands as well as displaying network and instrument status on the status panel.

5. NETWORK LAYER - Responsible for sending/receiving data or commands to/from the FEP and the remote towers. It packages the command (Figure 5) into a two byte packet containing the tower address, command and checksum. This checksum could be shortened so that the packet could also contain tower status information. Data is set in variable length blocks ranging from 0 to 255 bytes long. The tower however is only able to send the FEP its data table. The data table contains the data captured from the instruments and is presently seven bytes long (although it can be increased or decreased at will).

6. DATA LINK LAYER - Uses the Physical Layer to provide an error free data path. The 9th bit is added to the data packet to signal a command work. By setting this bit high, the FEP is able to interrupt the towers with a command or leave it low and only be connected to a previously selected tower.

7. PHYSICAL LAYER - This is the hardware level and includes the fiber-optic communi-

cations system and the internal UART on the 8031's.

The human interfaces asks the user for the instruments to be used on each tower. This information is encoded into the parameter block, along with the desired rate of sampling. The block is individually sent to each tower but all the towers simultaneously start execution with a global run command. The data table is uploaded from each running tower and stored in the FEP. After the FEP accumulates enough data, it will interrupt the PDP-11 through the DR11-C so that the host computer can display and/or store all the data on disk for subsequent transfer to magnetic tape.

#### METEOROLOGICAL INSTRUMENTATION

The meteorological instrumentation is important from two standpoints - it had to be custom designed for this experiment and its function should be under network control. In current experiments, there are two classes of tower: normal and special. The normal towers presently number six and contain instrumentation for measuring temperature and humidity differences and 3-axis wind speed. A special tower contains instrumentation which controls the millimeter wave source. A second special tower incorporates a Motorola 68000 based detector control subsystem for receiving and processing the millimeter wave radiation.

In the current experimental configuration differential rather than absolute (single point) temperature is being measured. The instrumentation for measuring temperature differences is based upon work done at the U.S. Army's Atmospheric Sciences Laboratories with incandescent light bulb filaments [7]. The tungsten filament wires respond to changes in ambient temperature with corresponding changes in resistivity sufficiently linear and fast as to make the sensors usable for many atmospheric measurements. Two such filament sensors are operated in a bridge configuration to measure differential temperature in the vicinity of each tower. This sensor configuration is operated approximately 2 meters off the ground with an adjustable sensor spacing of 10 cm to 1 meter.

As light bulb filaments are quite delicate, the most likely cause of temperature sensor failure is the opening of a filament. Sensor failure information is communicated through the network to the Transport Layer. A network status panel driven by the FEP indicates the status of all sensors the network for operator convenience.

Humidity is measured using a Lyman-Alpha hygrometer, similar to that described by Buck [9] but with improved electronics [10]. An ultraviolet source emits at the Lyman-Alpha absorption line of hydrogen. An ionization-type detector measures the source power at a fixed distance from the source. The detected signal will

correspond to the absorption of the UV radiation by the hydrogen atoms of water molecules between the source and detector. This received signal yields the absolute water vapor density (humidity) between the source and detector tubes when correctly calibrated. The nature of the detector tube allows very fast (sub-millisecond) measurement of humidity and is the reason why such a complex device was used.

For differential humidity measurements these detectors cannot be put in a bridge configuration as the temperature sensors were; consequently, the output signals from individually matched Lyman-Alpha hygrometers must be subtracted and filtered to yield a differential humidity. Considerable network control is exercised over the Lyman-Alpha hygrometers because of their complexity. The Applications layer is responsible for turning on the source tubes, waiting for the tube temperature to stabilize, and detecting tube failures. The most common cause of hygrometer failure is catastrophic failure of either the source or detector tubes. As with the differential temperature sensors, tube failures are detected by monitoring the difference signal between two hygrometers for a large, constant value. In the future, additional information on graceful failure, i.e. tube aging, might be developed by monitoring source tube impedance.

The wind sensors are commercially available three-axis generator-type anemometers [12], employing small, permanent magnet proportional tachometers, rather than optically encoded digital tachometers. Use of the former type is motivated by the much simpler processing electronics needed by the generator anemometers. Furthermore, optical tachometers produce pulse outputs which must be processed over relatively long periods to yield wind velocity - resulting in long response times and limited bandwidth for low wind velocities. An offset buffer amplifier provides signal conditioning to convert the bipolar voltages generated by the anemometers to a 0 to 5 volt signal for the A/D system with a level of 2.5 volts representing a zero velocity along the anemometer axis. Values greater than 2.5 volts indicate wind from the positive direction, values less than 2.5 volts are obtained for wind from the negative direction.

Initially, as shown in Figure 1, the focal plane scanner subsystem will not be directly tied to the network, but rather connected through two DZ11 ports on the PDP-11 for testing and debugging purposes. When testing is completed the scanner subsystem will be connected to the network and considered to be another tower. Because of the modular hardware design, this connection can be accomplished without having to break the fiber optic data path by merely connecting the scanner system to the fiber optic communications board of an existing tower. In essence, piggybacking two towers together to create a local tower cluster. This was done using the logical interconnection



feature of the fiber optic system. This ability is advantageous because it allows the transmitter-scanner path length to be easily changed for short path experiments or checking beam alignment along the propagation path. Tower clustering can be done easily anywhere in the network since tower address decoding occurs in the CPU of each tower rather than in the communications hardware.

#### PDP-11 CONTROL AND ACQUISITION SOFTWARE

Current software in the PDP-11 allows the user to access the network at the session layer since the applications and presentation layers are not currently implemented. A menu driven program called "DLOAD" written in MACRO-11 allows a knowledgeable user to load programs into the remote towers and the FEP for hardware testing and data collection. Towers are uniquely addressed by using octal values representing the tower number in the address byte of the command codes assembled in the FEP. A global tower address allows the user from the PDP-11 to start and stop program execution and data acquisition in the all of the remote towers simultaneously. After the towers are running, the user can either choose to display the data being collected in real time or have the data stored on disk for later analysis. If the user chooses the real time display of the collected data, the values representing the A/D channels and status bytes for each tower along with the time will be displayed on a VT125. The screen is only updated every 600 milliseconds due to the overhead in doing the data conversion and screen I/O in the PDP-11. If the user chooses to store the incoming data on disk, DLOAD prompts for an output file name, opens the file if the output volume is mounted and proceeds to write out the data in two block increments. Should some error occur in trying to open the data file the user is returned to the main menu. In the disk data acquisition mode data frames are collected in one second increments. The current version of the software uses the MRKT\$ (mark-time) directive and event flags to control data acquisition timing, this presents a drawback for us since the minimum sample interval can therefore only be 1 clock tick (1/60 sec.) which is too slow for some of the atmospheric phenomena we wish to study. The use of the KW11-K real time clock will alleviate this problem and allow burst data acquisition rates of 1 KHz. Some minor hardware modifications in the FEP and software modifications in both the FEP and the PDP-11 must first be done before high frequency data acquisition can be implemented.

#### SUMMARY AND CONCLUSION

A meteorological data acquisition system has been described which is being used for a variety of atmospheric propagation experiments. A simple, robust hardware design permits maximum experimental flexibility through software. The software can

permit up to sixteen towers on the system with the maximum data acquisition rate being a function of the complexity of the tower software. For simple tower functions and 6-8 towers in the system, 200 Hz data acquisition rates are possible. The limiting factor is how fast the FEP can transfer data to the host computer. A 16 bit FEP possibly based on the DEC J-11 chip would help to alleviate this bottle neck.

Tower instrumentation has been designed for simultaneous measurement of all analog inputs. Resistance type temperature sensors and Lyman-Alpha humidity sensors allow measurement of differential temperature and humidity at each tower. These can be processed by the host computer to yield the complex index of refraction. Wind sensors allow corrections to be made for air movement across the sensors.

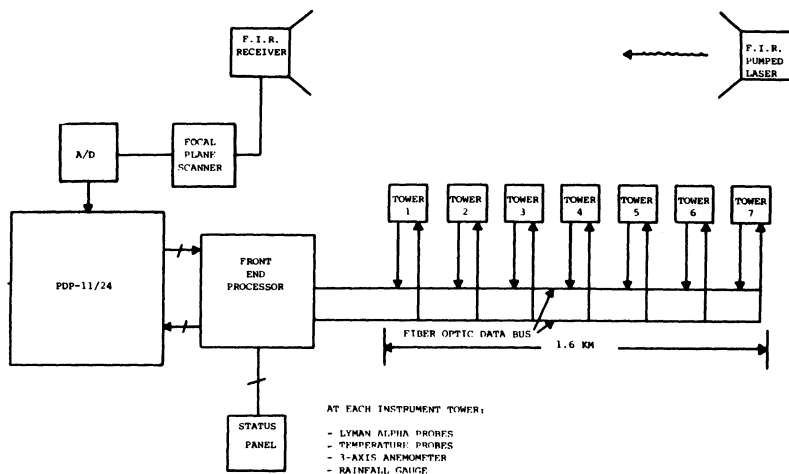
The network as described in this paper is being used to study atmospheric effects upon millimeter wavelength propagation, but there is no reason why such a local area network cannot be applied to such diverse applications as industrial control (where individual controllers may be networked to produce an integrated factory under complete computer control) and sensor systems for robotics.

#### REFERENCES

1. W.A. Flood, "Overview of Near Millimeter Wave Propagation," Proceedings SPIE 259, 52, October 1980.
2. J.C. Gibbons, "A Distributed Multi-Microprocessor Data Acquisition System," M.S. Thesis, Case Western Reserve University, Cleveland, Ohio, May 1983.
3. H. Zimmerman, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," IEEE Transactions Communications COM-28, 425-432, April 1980.
4. P.C. Clasper and F.L. Merat, "Atmospheric Propagation Studies at Near Millimeter Wavelengths," Proceedings SPIE 337, 81-87, May 1982.
5. Fiber optic transmitter HFBR-1002, receiver HFBR-2001, and HFBR-2000 cable. Hewlett-Packard Corporation, 640 Paste Mill Road, Palo Alto, California, 94304.
6. A.S. Tanenbaum, Computer Networks, Prentice-Hall, 1981.
7. D. Brown, U.S. Army Atmospheric Sciences Laboratory, White Sands, New Mexico, personal communications, 1981-1982.
8. D. Gillooly, P. Henneuse, "Multifunction Chip Plays Many Parts in Analog Design," Electronics, 121-129, April 1981.
9. A. Buck, "Notes on the fabrication of a Fixed-Path Lyman-Alpha Hygrometer," RSF 040-032-004, National Center For Atmospheric Research, Boulder, Colorado, December 1979.
10. R. Simons, J. VanderVoort, "Improved Instrumentation Electronics for the Lyman-Alpha Hygrometer," LLTR-20, Case Western Reserve University, Cleveland, Ohio, April 1983.
11. Analog Devices, One Technology Way, P.O. Box 280, Norwood, MA 02062.

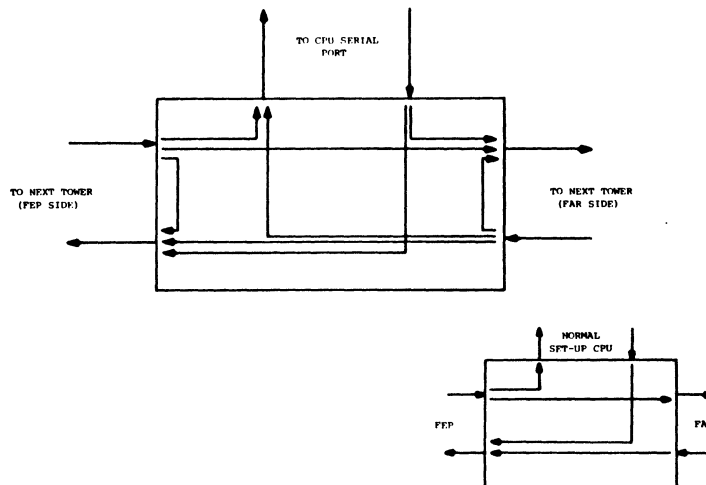
ACKNOWLEDGEMENTS

This work is supported by the U.S. Army Research Office under contract No. DAAG 29-81-K-0172. In addition the authors wish to acknowledge helpful discussions about micrometeorological instrumentation with Dr. W. Flood of ARD, Dr. R. Olsen of the Atmospheric Sciences Laboratory at White Sands, and Drs. S. Clifford and R. Hill of NOAA.



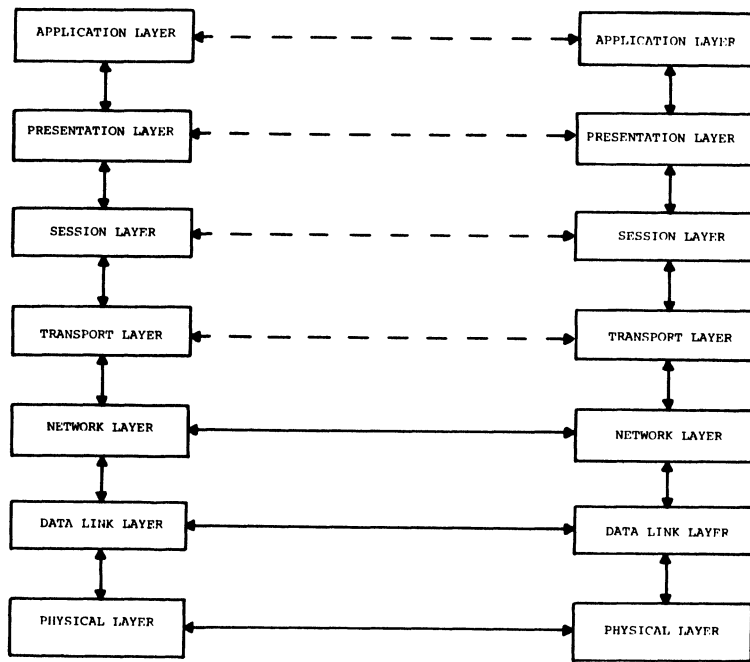
Meteorological Data Acquisition System

Figure 1



Communication Optics Logical Setup

Figure 2



THE ISO-OSI REFERENCE MODEL

Figure 3

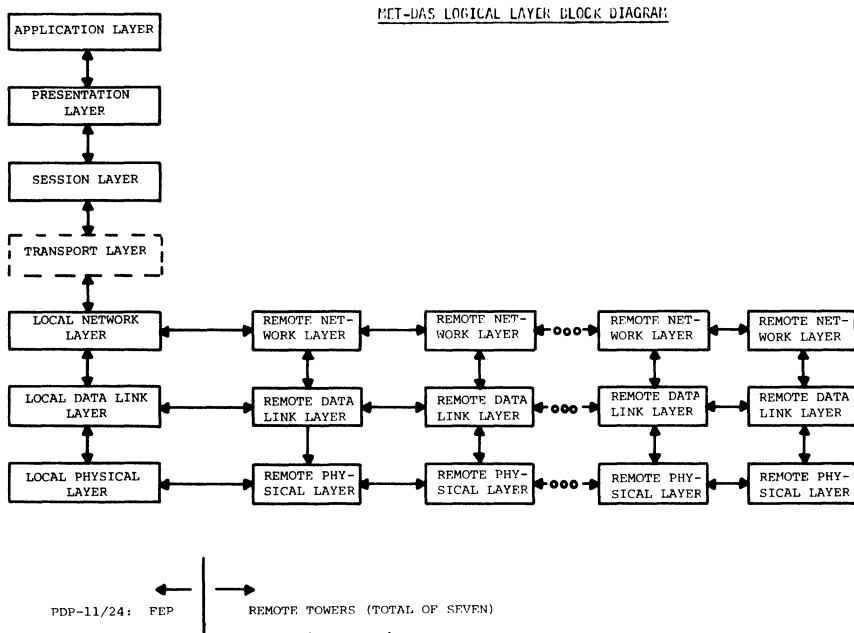


Figure 4

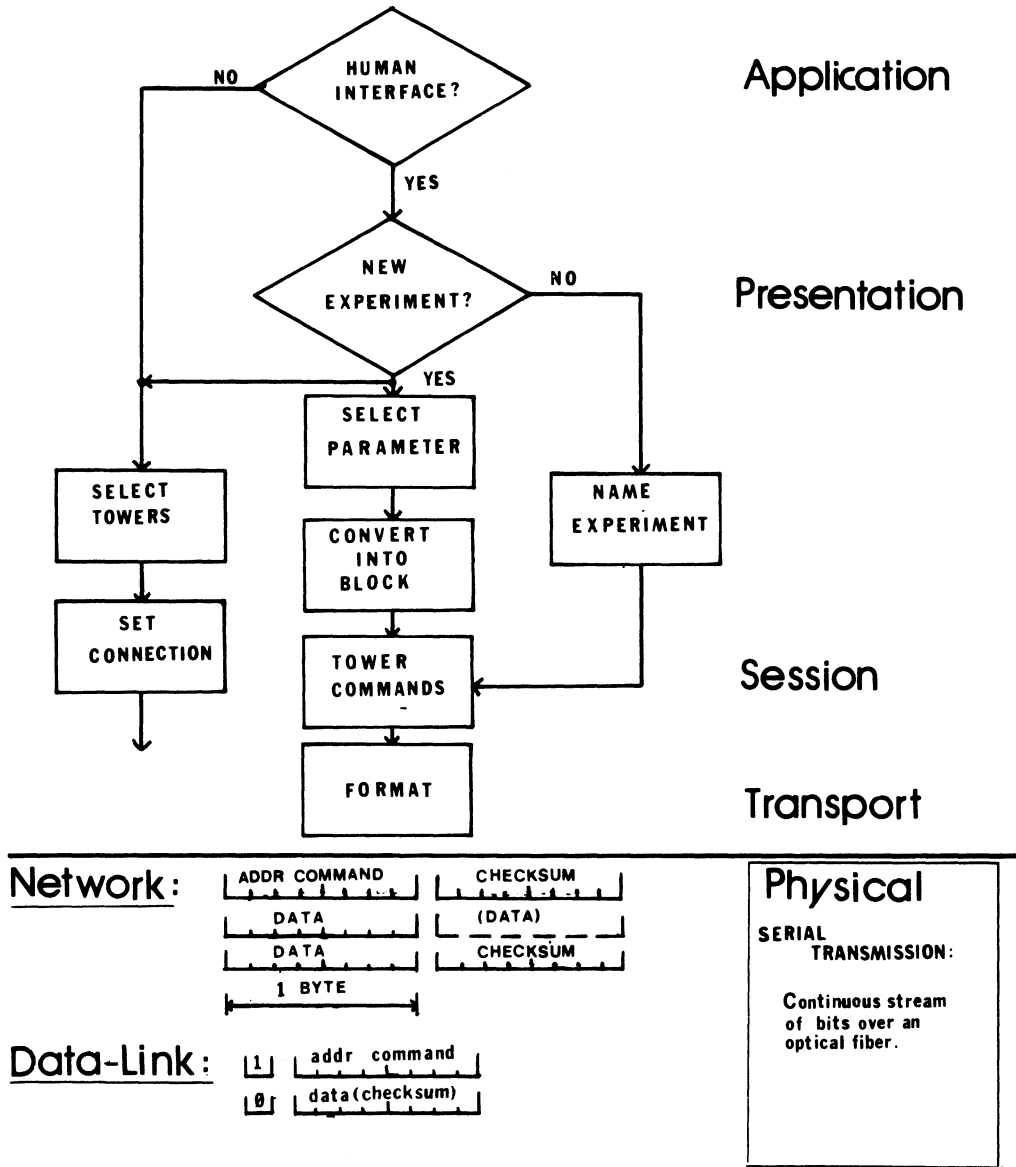


Figure 5



# PROGRAMMING WITH INDIRECT COMMAND FILES

Sharon Linnea Johnson  
Division of Epidemiology  
University of Minnesota  
Minneapolis, MN

## ABSTRACT

The indirect command processor has features which allow it to be used as a programming language. The essential programming language features of the Indirect command file processor and other selected directives, are included in this paper. Methods of structuring command files for readability and ease of modification are suggested.

The purpose of this paper is to describe those features of the RSX indirect command processor which make it a programming language and to suggest an approach to indirect programming style. According to Digital Equipment Corporation, the purposes of the indirect command processor are the following:

- (1) Testing hardware or system and user data structures
- (2) Manipulating user files and data structures
- (3) Execution of control structures
- (4) Creating a context for execution of tasks.

However, approaching indirect command files as programs results in more creative use of command files for applications development and system management. Remaining aware of programming style when structuring indirect command files produces files which are easier to understand and maintain.

The indirect command processor has four features which make it a programming language. There are three types of variables, which are referred to as symbols in the documentation and in this paper. Basic control structures for flow control and logical tests are provided. Special directives for terminal input are included as well as basic file input and output. It is in its executable statements that the indirect command processor is most powerful. Any DCL or MCR command can be used as an executable statement in an indirect command file. The values of symbols can be substituted into these DCL or MCR commands.

### Using the Programming Features

To use variables in a command file, symbol substitution must first be enabled. There is no reason not to enable substitution. Substitution must be enabled at the beginning of the command file, before initializing any symbols. This is done by using the statement below:

```
.ENABLE SUBSTITUTION
```

Legal symbol names can be from one to six characters long. All characters must be alphanumeric or dollar signs. A symbol name must begin with either a letter or a dollar sign. There are three types of symbols, logical, numeric, and character string. Numeric symbols are integers ranging in value from 0 to 65535(10). Floating point numbers are not supported.

Unlike other programming languages such as FORTRAN or Pascal, which include separate statements for declaration and initialization of variables, indirect symbols are both declared and initialized with .SETa statements, where a determines the type of symbol in question. Numeric and string symbols each have a .SET statement and there are two types of .SET statements for logical symbols, depending upon whether the symbol is being assigned true or false. The sample below demonstrates the initialization of symbols of various types and values:

```
.SETT MOUNTD
.SETF OVERLY
.SETN COUNTR 0
.SETS MAX "MAXBUF="
```

To substitute the value of a symbol into an executable statement, the symbol name is surrounded by single quotes. In the example below, a FORTRAN-77 compile would be performed, with the values of the string symbols OBJ and FTN substituted into the command.

```
F77 'OBJ','OBJ'/-SP='FTN'
```

Various special symbols of all three types are defined within the indirect command processor. They are indicated by reserved names surrounded by square brackets. <EOF>, a special logical symbol is true when end of file is reached reading an input file. <STRLEN> is a special numeric symbol returned by certain directives. <EXSTRI> is a string symbol returned by string manipulation and other statements

Indirect numeric operations include addition, subtraction, multiplication and division. The operators, +, -, /, and \*, are similar to those of FORTRAN. These operations are performed using the .SETN directive. Separate directives for incrementing or decrementing a numeric symbol by one are also included. These are particularly useful when coding loop structures. The code segment below includes all

three types of numeric operations.

```
.INC COUNT1
.DEC COUNT2
.SETN C A+B
```

Despite the limitations of its arithmetic, the indirect command processor provides a good variety of string operations. To assigning character strings, all literals must be surrounded with single quotes. Substrings are surrounded by square brackets and are referenced with n1:n2 by character number, similar to substring references in FORTRAN. The concatenation operator for merging multiple strings into one symbol is a plus sign. Strings can be divided into substrings based on the occurrence of a selected delimiter character using the .PARSE directive. In the example below, if the string symbol FILE contained "DR2:[300,300]PROG.TSK", the string DEV would contain "DR2" and the string NAME would contain "[300,300]PROG.TSK".

```
.PARSE FILE ":" DEV NAME
```

The .TEST statement serves as an index function. In the example below, the string FN2, a filename, is tested to determine if it contains a substring. If the string contains the substring in question, the special numeric symbol <STRLEN> contains the value of its index. If the string does not contain the substring, <STRLEN> equals zero.

```
.TEST FN1 ".PAS"
```

Using substitution, strings can be converted to numerics. In the example below, the special string symbol <TIME>, which contains the system time, is converted to a four digit numeric value. In the first statement, concatenation is used to create a new string symbol, T, without a colon. Next, the value of the string T is substituted into the numeric symbol NTIME.

```
.SETS T <TIME>[1:2]+<TIME>[4:5]
.SETN NTIME 'T[1:4]'
```

The commenting features of the indirect command processor make it easy to create internally documented programs. A semicolon in the first column of a line indicates a comment which is echoed on the terminal unless the statement .ENABLE QUIET has been invoked. Enabling quiet also suppresses the echo of any operating systems commands used as executable statements. If quiet is disabled, the text, including the semicolons, are displayed on the user's terminal. Enabling quiet is used to its best advantage when you wish a series of commands to be executed, but not echoed on the user's terminal. This is useful in command files designed for naive users. Using .ENABLE and .DISABLE QUIET to prevent comments from being

echoed becomes confusing to program and is not recommended.

Instead of enabling and disabling quiet around documentation text, the other type of comment line should be used. A period in the first column and a semicolon in the second column of a line indicates a comment which is not displayed regardless of the status of QUIET. This is the best way to include documentation text in a command file. Exclamation points can be used following indirect statements to place comments on the same line. It is important to remember that this exclamation point syntax does not apply to MCR or DCL commands used as executable statements. Since indirect statements are not echoed, inline comments with exclamation points are not displayed.

A special set of statements for querying users for values of symbols are included in indirect. There is a separate .ASK statement for each type of symbol. These statements consist of .ASK, .ASKN, or .ASKS followed by an optional set of default values, time out intervals, ranges for numeric symbols, or acceptable lengths for strings enclosed in square brackets, and then the name of the symbol followed by a string to be used as a prompt. Default values are used if the user responds with a carriage return. If ranges have been set and out of range values are entered, the prompt is repeated.

To use time out intervals, .ENABLE TIMEOUT must be included in the command file. If a query statement times out without a user response, the special logical symbol <TIMEOUT> becomes true. Based on the value of <TIMEOUT> conditional action can be taken. In the first example, the logical symbol OVERLY has a default value of false and a response time out of two minutes. "Using overlay?" is the prompt which appears to the user. Acceptable user responses are the letter Y or the letter N. In the second example, the numeric symbol OPTN has an acceptable range of one through nine, a default value of nine, and a time out limit of five minutes. The string symbol BLDFIL can be up to 32 characters long, has a default value of whatever the string symbol P contains, and a time out limit of two minutes.

```
.ASK [<FALSE>:2M] OVERLY Using overlay
.ASKN [1:9.:9.:5M] OPTN Enter option
.ASKN [1:32.: 'P':2M] BLDFIL Name for .BLD
```

Files for input or output can be opened for read, write, or append. If more than one file must be open simultaneously, an optional number can be associated with each file. Individual files can also be closed. Text is read from files into string symbols, one line at a time. Lines up to 132 bytes long can be read. The .DATA directive writes one line of text, which can include substituted symbols, to a

file. The statements .ENABLE DATA and .DISABLE DATA can write more than one line of data at a time. Opening the device TI: as a file and using .ENABLE and .DISABLE DATA allows larger sections of text to be displayed without the semicolon in the first column of each line which would appear if comments were used. The section of code below demonstrates the various file processing directives.

```
.OPENR #1 CHOICES.PRM
.OPEN #2 'NAM'.CTL
.OPENA #3 'TEST'.OUT
.READ #1 LINE1
.DATA #3 This is your test output.
.CLOSE #3
.OPEN #3 TI:
.ENABLE DATA

 working

.DISABLE DATA
```

Special symbols which contain the status of files include <EOF> and <FILERR>. When end of file is reached attempting to read a file, the logical symbol <EOF> is true. The FCS file code for each read attempt is in the special numeric symbol <FILERR>. If <FILERR> equals 230(10), then no such file was found. Both of these symbols can be tested and used in error recovery code.

#### Structuring Indirect Command Files

Indirect command files have only a few built-in control structures. In an unstructured language, a structured, disciplined style depends the programmer's own efforts. There are several general rules which improve readability and make command files easier to maintain. Modular structure should be used as much as possible. Avoid creating a spaghetti-like network of .GOTO statements and labels. A consistently structured typing style, using indentation, blank lines to create white space, and comment lines containing rows of periods to separate subroutines greatly increases readability.

Labels and .GOTO statements must be used to build control structures in indirect command files. DO loops and IF-THEN-ELSE blocks are not built into the language. A label contains up to six characters which can be alphanumeric or dollar signs, is preceded by a period and is followed by a colon. Descriptive names make command files easier to follow than simple statement numbers. A sample series of statements needed to implement a fixed count loop are shown below.

```
.SETN NSTOP 10.
.SETN COUNTR 1
.LOOP1:
 .IF COUNTR GT NSTOP .GOTO EXLOOP
```

```

 .
 .
 .
 .INC COUNTR
.GOTO LOOP1
.EXLOOP:
```

Two types of logical test statements are used to test the value of a logical symbol or a logical condition. To test the value of a logical symbol, .IFT or .IFF is used. Conditions are tested with .IF. The logical operators EQ, NE, LT, GT, LE, GE can also be typed as =, <>, <=, >= . The logical and logical or operators must be preceded by periods, .AND and .OR. Logical test statements are followed on the same line by the action to be performed if the condition is true. The action can be an indirect command statement or an operating system command.

```
.IFF CONTIN .GOTO FINISH
.IF ICNT GT 3 .OR .IFT <EOF> .GOTO EXIT
```

My personal preference for creating well-structured command files is to use subroutines wherever possible. The main routine of the command file can then be reduced to a series of calls to subroutines which perform each step. A subroutine is called with the statement .GOSUB followed by the name of the subroutine. The subroutine itself begins with the name of the subroutine in the same syntax as a label and ends with the .RETURN statement. .RETURN causes the command file to continue processing at the line following the subroutine call.

Blocks can be used within a command file to create sets of local symbols. All symbols defined within the block are local and become undefined outside the block. Blocks are delimited with the statements .BEGIN and .END.

Global symbols which remain defined when one command file invokes another command file are designated by a dollar sign as the first character of their names. To use global symbols in a command file, the statement .ENABLE GLOBAL must be invoked first. A good place to do this is at the beginning of the file. One command file can invoke another with @filename or with .CHAIN followed by the command file name. Up to nine command line parameters can be included after the file name when invoking indirect command files. The are parsed using spaces into the reserved symbols P1 through P9.

Two different statements are used to stop indirect command processing. Each behaves slightly differently. The .STOP statement stops processing altogether, including the invoking command file if the command file containing the .STOP was invoked from another command file. Using .STOP in command files which are invoked by other command files can cause

unexpected side-effects, especially if you invoke the file from the middle of a system startup command file. In these cases, .EXIT should be used instead. The .EXIT statement causes a return to the invoking command file and continuation at the line following @ or .CHAIN. If the file was not invoked from another command file, .EXIT stops processing.

## Some Special Symbols and Statements

A variety of special statements and symbols are provided. This section describes only a few. The statements .TESTDEVICE and .TESTFILE are used to obtain information about the status of devices and files. .TESTDEVICE places a string containing device attributes in the special string symbol <EXSTRI>. Using .TESTFILE places the FCS file code in the special numeric symbol <FILERR>. This information can be checked to determine if the necessary files or devices are available before attempting processing.

Command files can delay for a specified time span or pause until restarted by the terminal which invoked them. The .DELAY statement is followed by an number and a time unit, which can range from ticks to hours. Following a .DELAY statement with a control-G character on a comment line causes the terminal to beep after the designated time has passed. The .PAUSE statement displays a "PAUSING" message and the command required to restart the command file. It is unwise to include .PAUSE in command files invoked from batch jobs.

The screen control features of the VT-100 terminal are available from indirect command files. This is done by opening the device TI: as a file and using the .DATA directive followed by the chosen escape sequence. The escape sequences are described in detail in "Nifty Things to Do with RSX Indirect Command Files", by Watson, et. al. in the DECUS Spring 1983 RSX/IAS SIG Symposium Handout.

Special symbols contain useful information about the system, the terminal on which the command file is running, and the success or failure of commands invoked. The exit status of the last MCR or DCL command line is contained in the special numeric symbol <EXSTAT>; 0 is warning, 1 success, 2 error, 4 severe error, and 17 no status returned. The logical unit number of the user's default device is in the numeric symbol <SYSUNIT>. A range of octal values corresponding to various baud rates are used to code the current terminal speed in <TISPEED>. A string containing the login UIC (user identification code) of the current user in the form "[ggg,mmm]" is found in <LOGUIC>. The current system date as "dd-mm-yy" and system time as "hh:mm:ss" are found in <DATE> and <TIME> string symbols. If timeout has been enabled, <TIMOUT> is true if the last directive timed out waiting for a response.

## Uses of Indirect Command Files

This section is a description with selected examples of some indirect command files implemented in the Division of Epidemiology. Our major computer applications are research and statistical analysis. For brevity and format consistency with the previous examples, all code samples are shown in upper case with comments in lower case and large documentation sections have been omitted. Typically, I use lower case for indirect command statements and MCR or DCL commands and upper case for comments. A typical documentation section contains the original date of the command file, the programmer's name, a description of the command file's purpose and algorithm, and its relationship to particular research projects, if applicable.



Indirect command files have been especially useful as interfaces for naive users running production jobs. Interactive command files prompt the user for the answers to a series of simple questions, then formulate the appropriate commands, perform preliminary file manipulation, run the necessary programs to complete the job, and delete temporary files. The ability to set acceptable ranges and default values for user responses is particularly useful in this type of application. One of our applications for this type of command file is the production of test result letters for participants in health surveys.

Another application for indirect control files is structuring batch jobs. One example of this was the reformatting and recoding of death certificate data files involving three states and twenty-three years of deaths. The command file provided loop control to run the program in batch on selected series of states and years. Command files run in batch obtain answers to .ASK queries by reading the series of responses from the batch command file. Responses to the indirect command statements are not preceded by dollar signs, as shown below.

```
$@DFCODE
Y
65
4
```

We have developed a set of utilities to create statistical analysis data files for research users. The multiple FORTRAN-77 sources, command files for task-building, and command files for running the tasks are stored in a universal library. To rebuild the utility from source, I wrote a command file which extracts all sources and command files from the universal library, compiles the FORTRAN-77 programs, recreates the object libraries, builds the tasks, and then deletes the sources, objects, objects libraries

and other files which are no longer needed. This file, CREATE.CMD, was then added to the same universal library with the optional descriptors EXTRCT:ME:FIRST. This reduced the steps required to rebuild the entire utility to extracting CREATE.CMD from the universal library and then invoking it.

The BMDP (Bio-Medical Computer Programs) statistical package is used for most of our statistical analyses. To prevent major degradation of system performance, BMDP must be run in batch between 8 a.m. and 5 p.m. Three batch processors are available during the day, one of which is limited to test runs. At 7 p.m., two more batch processors are added and the test run batch processor becomes available to longer jobs. At 7 a.m. the two additional processors are deleted. The addition and removal of batch processors is implemented with a task run in the clock queue at 12-hour intervals. BAPPER.TSK, which was written in FORTRAN-77, calls the system subroutine SPAWN to invoke a command file, BAPPER.CMD, with the parameter AM or PM, depending upon the system time.

The main routine of BAPPER.CMD is extremely short. It consists of the three lines shown below. The value of the command line parameter P1 is substituted as the name of the subroutine in the .GOSUB call. The subroutine AM contains the operating system commands to deassign BAP2 from the batch queue, assign it to QWK, the test queue, and delete BAP3 and BAP4. The subroutine PM deassigns BAP2 from QWK, assigns it to the batch queue, creates BAP3 and BAP4 and assigns them to the batch queue.

```
.ENABLE SUBSTITUTION
.GOSUB 'P1'
.EXIT
```

option number chosen, a subroutine is called. When the subroutine completes the menu is displayed again. A timeout interval of five minutes applies to the main menu and a two minute time out interval to all other prompts. If <TIMEOUT> is true, the terminal logs off. The main routine of this command file is shown below, followed by the subroutine DISOP which displays the option list. Subroutine DISOP illustrates the use of VT-100 screen control escape sequences to produce a bright blinking display and then return to normal display.

```
.IFENABLED QUIET .DISABLE QUIET
.;if called from another cmd that
.; had it enabled
.ENABLE SUBSTITUTION
.ENABLE TIMEOUT
.GOSUB INI
.;
.OPT:
.; main loop for option query to which
.;subroutine returns
.;
.GOSUB DISOP
.; display option list
.ASKN [1:9.:9.:5m] OPTN Enter option:
.IF OPTN EQ 1 .GOSUB ED
.IF OPTN EQ 2 .GOSUB COM
.IF OPTN EQ 3 .GOSUB BLD
.IF OPTN EQ 4 .GOSUB MAKBLD
.IF OPTN EQ 5 .GOSUB RUNTSK
.IF OPTN EQ 6 .GOSUB PURG
.IF OPTN EQ 7 .GOSUB TYPEIT
.IF OPTN EQ 8 .GOSUB OTHER
.IFT <TIMEOUT> .GOTO DONE:
.IF OPTN EQ 9. .STOP
.GOTO OPT
.;
.DONE:
.; beep & log off
;^G
BYE
.DISOP:
.; display main list of options on crt
.;
.OPEN TI:
.DATA <ESC>[1m
.ENABLE DATA

1-- edit
2-- compile
3-- build
4-- create .BLD file
5-- run
6-- purge
7-- type
8-- pause
9-- exit

.DISABLE DATA
.DATA <ESC>[0m
.CLOSE TI:
.RETURN
```

#### Conclusion

The indirect command processor is a useful tool for implementing both applications and systems programs. The ability to include any MCR or DCL command within a command file makes it an extremely powerful programming language. It does lack structured programming features. To compensate for the lack of built-in structures, it is important to develop a disciplined, modular style in order to create indirect command files which are easily revised.





Martha R. Szczur, Dorothy C. Perkins, David R. Howell  
NASA/Goddard Space Flight Center, Code 635  
Greenbelt, MD 20771

## ABSTRACT

The Transportable Applications Executive, developed by NASA's Goddard Space Flight Center, is designed for use on interactive analysis systems, and provides executive services such as menu and command user interfaces, command procedures, program library management and asynchronous and batch processing. Subroutines for use by applications programs are also provided. These include services for parameter acquisition, user-program communications, message logging, disk input/output and device-independent image display. TAE is designed to be portable to new operating systems; use of its service subroutines enhances the portability of application programs as well. This paper explores TAE concepts and structure, and the application of TAE to an interactive system.

## 1.0 INTRODUCTION

In 1975, the Information Extraction Division at NASA's Goddard Space Flight Center took on its first charter to build an interactive analysis system for Goddard research scientists (Bracken et al., 1977). That was followed in 1977 by a second system (Dalton et al., 1981), and in 1978 by a third (Dalton et al., 1979). In 1979-80, planning for four more systems -- for meteorology, earth resources, oceanography and data base management -- began. Previous experience and the new system requirements clearly indicated that there existed common structures and system service routines that could serve all of these systems, and that what was unique to each was application software. At the same time, only limited manpower was available to support the development of all of these systems simultaneously.

In this environment, the Transportable Applications Executive (TAE) was conceived. It was proposed to be a general purpose software executive which could be applied to the various systems being developed. The idea of a reusable executive was viable because -- despite diversity in the potential user communities -- they all made essentially the same demands on their supporting computer systems

These shared requirements determined the initial requirements placed on TAE: a consistent, controlled, easily learned interactive user interface; easy incorporation of interactive imaging and graphics; batch processing; and potentially large collections of programs, interrelated by the need to share data.

Experience with prior systems and changing technology led to the adoption of some additional goals: the system must be usable by novice and casual users, yet give flexibility and freedom to expert users; a user should be

able to locate data and programs easily, and to get on-line information which explains the substance and operation of the system; the system should shield the user from the host operating system; it must be easy to reconfigure the system and to add new programs; the executive must be written to be portable to new machines, thereby being a catalyst rather than a hindrance for users with changing computational needs; the executive must supply common services needed by applications programs, which would enhance their portability as well; and the system must be independent of project, discipline or data.

TAE has grown from an early, limited prototype, first made available in August, 1981, to a powerful, broadly used operational version, most recently upgraded in March, 1985. Its development has seen regular stages and releases, feedback at every step from a growing user community and interested observers, and an evolutionary growth shaped by the experiences and criticism of its users. Throughout its development, TAE has been influenced by work in virtual operating systems, human factors research, command language design and standardization efforts, and system portability.

This paper discusses the features of TAE and how they can be and have been effectively applied to interactive analysis systems. Brief attention is also given to details of implementation which apply to the DEC environment.

## 2.0 TAE FEATURES

TAE consists of two distinct bodies of software. The first is the TAE Monitor (TM). It handles all user-computer communication, and locates, executes, and interacts with the applications and utility programs initiated by a user. The second is a subroutine library which provides several packages of commonly needed functions for applications programmers.

## 2.1 User-TAE Communication

TAE is most visible in its extensive interface for interactive users. Inexpert or casual users can make their way through a system driven by menus and augmented by extensive on-line help, while experienced users have a powerful language for commanding the system and controlling the environment. Any user always has a variety of on-line explanation of system operation and functions. A user can type the string HELP (or, H) at any time to get assistance on how to operate the system. The HELP given is tailored to the particular situation a user is in, and tells him/her what actions are available, including what other kinds of HELP can be obtained.

An interactive user's purpose in using an applications executive is to exercise the analysis functions of that system. To accomplish this under TAE, a user manipulates two entities: procs and parameters.

### PROCS

A proc is some function -- whether analysis, display of products, or housekeeping -- which a user wants to exercise. Internally, a proc may be either a process (an executable program) or a command procedure, a predefined sequence of commands, including the execution of procs. A proc is made known to TAE by the existence of a disk resident, editable text file called a proc definition file (PDF), which internally identifies itself as a process or procedure. TAE understands and treats each differently. To the user, however, the distinction is transparent.

Sets of related procs are stored in "libraries." On any one system, there may be many libraries of procs, and users may choose to employ all libraries or a selected subset. The libraries are searched in a defined order whenever a proc must be located. First searched is the "user library," usually a user's private proc collection. Next are "applications libraries," one or more libraries, typically of related applications, which a user (or system manager on the user's behalf) may choose to include in the search. Finally there is a "system library," which typically contains procs of interest to all users.

### PARAMETERS

Each proc may have associated parameters, defined within the proc's PDF. Parameter values are the means a user has to adapt the actions of a proc to immediate needs. For example, an input parameter may be defined as the data file on which a proc is to work. Parameters may be integer or real numbers, strings or files, and may be declared to be either input or output. They may have default values, multiple values, and restrictive ranges (or lists, for strings or files) which will be validated by TAE before they are received by the proc. They may also be declared to be optional (null value). Whenever a proc is executed, parameter values chosen by a user (whether explicit or default) are packaged by TM and sent to the proc for processing. TAE

allows a user to save sets of parameter values for easy, repeated application to a proc.

In addition to defining the bounds of a parameter, a PDF also contains help text for a proc: information on the proc itself, and descriptions of each parameter.

#### 2.1.1 Interaction With TAE Through Menus

The menu user of TAE sees a controlled, ordered interface in which he/she chooses a path through the system from a series of formatted lists of options. The choices which can be made from any one menu are restricted, but menus are typically designed to group related functions, in anticipation of the likely choices a user will make. Under TAE, menus are arranged in a tree. The leaves of the tree are procs. Any one menu or proc may appear in the menu tree as often as desired.

Once a user reaches a proc, he/she interacts with TAE by "tutoring," to enter parameter values. A user may also have occasion to respond to messages. These interactions are described in this section.

### MENUS

Figure 1 shows a typical TAE menu.<sup>1</sup> The actions available to a menu user are presented to the user through a prompt line at the bottom of the menu. These options include selecting a numbered proc or menu, requesting help information, returning to previous menu or to top of the menu tree, traversing to another named menu, switching to command mode or logging out of the system.

Figure 1: TAE Menu

```
Menu menu "MUTIL", library "TAE$MENU"

* LAS Utilities Menu *

1) Image modification utilities
2) Image analysis terminal utilities
3) General purpose utilities

Enter: selection number, HELP, BACK, TOP, MENU, COMMAND, or LOGOFF
? HELP
```

### TUTORING

In "tutor" mode, a formatted list of parameters is presented to a user for perusal and editing. The name tutor implies part of the function of this mode, which is to teach a user the command language of TAE. Tutor mode assists a user in understanding and entering parameter values, and in running a proc.

A menu user enters tutor mode whenever the node chosen from a menu is a proc. Tutoring uses the proc's PDF to find the names and brief descriptions of parameters. Initial tutoring on a proc occurs before the proc is initiated. A proc may also use tutoring to acquire additional parameters from a user.

Figure 2 shows a typical tutor screen. The options available to a tutor mode user include: assign values to parameters, request Help, select a particular tutor page, scroll through values of a multi-value parameter, execute a proc, and save/restore parameter sets. In tutor mode, there is a simple editor which allows users to retrieve and edit parameter values displayed on the screen.

Figure 2: TAE Tutor Screen

| parm   | description                                     | value |
|--------|-------------------------------------------------|-------|
| OUT    | Name of output image file                       |       |
| WINDOW | Window of input image<br>(IN)<br>=(SP,SL,NP,NL) | 0 (1) |
|        |                                                 | 0 (2) |
|        |                                                 | 0 (3) |
|        |                                                 | 0 (4) |

Enter: parm=value, HELP, PAGE, SHOW, RUN, EXIT, SAVE, RESTORE; RETURN to page.  
?

A variation on tutoring which does not require a full screen and minimizes the amount of information written to the terminal is also available. It is used for hardcopy terminals, unsupported CRTs, or user preference.

#### MESSAGES

Messages may come from TAE itself, or from procs through TAE. A typical message has the form:

(WHOSE-WHAT) description

where WHOSE identifies the source of the message (e.g., TAE, GEMPAK, METPAK) and WHAT is a key (e.g., NOSUCHFILE). Any message may be supplemented by additional help, which a user accesses by typing "?" when the message occurs, or by typing "HELP-MESSAGE WHOSE-WHAT" in command or proc interrupt mode.

#### 2.1.2 Interaction With TAE Through the TAE Command Language

Users who choose to interact with a system through the TAE Command Language (TCL) can freely control and direct system activities. The extent of flexibility is directly related to the user's understanding of system operation and content. Unlike menu users, who have a limited set of possible actions and very little to remember at any one time, TCL users are limited only by the breadth of available system functions and their ability to remember how to use them.

##### 2.1.2.1 Interactive Use of TCL

The commands available to a user are either names of procs or TAE intrinsic commands. Procs are the application and utility functions of a system, as described in section 2.1. TAE intrinsic commands perform simple functions, generally to change a user's operating environment. Table 1 is a sample list of some of the TAE intrinsic commands available to interactive users.

Table 1: TAE Interactive Intrinsic Commands

- TUTOR
- DEFCMD/DELCMD
- ENABLE-LOG/DISABLE-LOG
- DISPLAY
- LET
- MENU
- SETLIB
- SHOW
- EXIT
- LOGOFF
- RESTORE/SAVE

Commands may be modified by subcommands, in order to group related functions, e.g., DELETE-GLOBALS, DELETE-LOCALS. Users may also define their own commands, using the DEFCMD intrinsic, which assigns an alias to a command string. Whenever that alias is used as a command, TAE substitutes the equivalenced string in place of the alias before processing the command. The hierarchy for command resolution is first user commands, then intrinsics, and finally the proc search.

TAE allows procs to be run synchronously, asynchronously or from a batch queue. Saved parameter sets may be applied to a command, or parameter values may be entered on a command line, positionally or by keyword. Parameter names may be abbreviated.

An in-line editor is available for retrieving, editing and resubmitting previous commands.

A command user enters tutor mode on any proc or intrinsic command by using the TUTOR command. For a command user confronting an unfamiliar or complex proc, tutor mode is a form of help or a fallback position. Like menu users, command users will see tutoring whenever a proc requests parameter input.

##### 2.1.2.2 Command Procedures

A TAE command procedure is a contained collection of TAE commands, executed as a single named function. This powerful feature allows users to set up standard, repeatable sequences of commands for regular operations. Procedures are invoked with exactly the same syntax as processes. Figure 3 is an example of TAE command procedure.

Figure 3: TAE Command Procedure

```
PROCEDURE HELP=*
 PARM BAND TYPE=STRING
 REFGBL DATA
 LOCAL X TYPE=STRING
BODY
 let X=DATA//BAND
 DISPLAY X
 COPY IN=&X OUT=TEST.DAT
END-PROC
.TITLE
 INFORMATION FOR THE PROCEDURE
 AND ITS PARAMETERS.
:
.END
```

TCL supports several intrinsic commands which control the definition and execution of command procedures. A sample list is provided in Table 2.

Table 2: Intrinsic Commands Used in Procedures

|                   |                                                    |
|-------------------|----------------------------------------------------|
| FOR/END-FOR       | ITEMIZED LOOP CONTROL                              |
| LOOP/END-LOOP     | INFINITE LOOP                                      |
| IF/ELSE/END-IF    | CONDITIONAL EXECUTION                              |
| IF/ELSE-IF/END-IF | MULTI-WAY CONDITIONAL EXECUTION                    |
| BREAK             | EXIT A LOOP                                        |
| GETPAR            | GET DYNAMIC PARAMETERS                             |
| GOTO              | UNCONDITIONAL BRANCH                               |
| LET               | VARIABLE ASSIGNMENT                                |
| NEXT              | FORCE NEXT ITERATION                               |
| PUTMSG            | WRITE A MESSAGE TO STANDARD OUTPUT/<br>SESSION LOG |
| RETURN            | TERMINATE PROCEDURE EXECUTION                      |
| STOP              | TERMINATE ALL PROCEDURE LEVELS                     |
| WRITE             | WRITE A STRING TO STANDARD OUTPUT                  |

TCL language features such as global and local variables, substitution, assignments, input and output parameters and expression evaluation support programming through procedures.

## 2.2 TAE Subroutine Library

The TAE subroutine library provides common TAE or host services for the application programmer. These routines also isolate applications code from host operating system services. Such standard interfaces, resident on all TAE hosts, allow application programs to be host independent, thereby enhancing their portability.

TAE provides service routines for parameter processing, image I/O, terminal I/O, message display and a few miscellaneous initialization and termination functions.

At its simplest, parameter processing includes the passing of parameter values -- those values entered by a user and validated by the TAE monitor -- into a program. Unless there are interdependencies between parameters too complex to be checked by TAE, a program accepts the values as valid and carries out its functions.

A program may also "dynamically" tutor a user for additional parameter input by using all or some of the parameters defined in a PDF, with optional modification of their default values before display. A user may be tutored for the complete, initial PDF, for a subset of that PDF, or for parameters from a totally different PDF, as required by the program.

Formatted parameter sets may also be written to disk for later retrieval by the same or another program.

The image I/O subroutines provide a simple package which handles band sequential or band interleaved images and allows user-defined labels to be inserted at the front of an image. This package is optimized for efficiency of I/O, not storage. While the internal labels are designed to describe simple image files, this package in fact can be used for any direct access block I/O requirements.

## 2.2.3 TAE Subsystems

TAE provides a strong basis for the development of special purpose services which are run under TAE and are options to the basic executive system. Four of these optional subsystems have been released as prototype versions. They are briefly described below.

DMS, a Display Management System: provides device-independent access to raster image devices, and user services to support this access.

CM, Catalog Manager: provides a system independent means of building and maintaining catalogs of disk files and related descriptive information. CM does not replace the host system file handling capabilities, rather compliments them by allowing users to store more meaningful information about their files.

RCJM, Remote Communications Job Manager: allows a TAE user on a local machine to run TAE procs on remote machines. The prototype was designed to furnish some of the remote TAE capabilities and to provide some experience in designing and implementing TAE in a network environment.

Window Manager: provides as an extension to TAE, a user interface that uses graphic windows and is based on current concepts of effective user displays and user dialog. The prototype currently executes on the VAX Station 100 and the VT220. Other targeted work stations include the IRIS, APOLLO and SUN workstations.

## 3.0 BUILDING A SYSTEM WITH TAE

TAE is not a complete system in and of itself. Rather, it is designed to provide a core of services needed in any interactive system, and to be the framework upon which customized applications can be installed and managed. The major TAE services -- user-interface, command language, proc activation and service subroutines -- were described in section 2. Some of the ways in which a system builder can tailor a TAE-based system are mentioned below.

Global Variables Globals are session wide variables used to pass information between procs and to provide the TAE Monitor with dynamic control information. In a particular system, certain parameters or special conditions -- such as file names, area definitions, or an index into a data base -- may appear repeatedly. To avoid requiring the user to enter the same information each time, the parameter may be set up as a global variable.

TAE provides a basic set of global variables for session control. These vary from letting a user define his own commands line prompt to controlling how messages are displayed (quiet or bell, pause or continue).

A system builder may also create additional sets of global variables which may be individually loaded. These sets are defined in special PDFs

which when "RUN" are loaded and made ready for use.

Session Initialization/Termination: At the beginning of a session, TAE executes a system wide logon procedure which can perform a variety of functions, such as setting up standard application libraries and global variables, or restoring variables saved in a previous session. It is also possible at logon to display an installation "bulletin board." After the session is initialized, a user logon proc is run to customize the session for an individual user. For example, special application libraries could be added to the standard set.

At the end of a user's session, system and user logoff procs are run. These are used primarily for cleanup and saving the user's environment.

System Appearance: The physical appearance of TAE is designed to help the user learn the system quickly. Such attributes as standardized display formats, rigorous categorization, sequential task subdivisions, and functional redundancy create a consistent learning environment. For example, menu interaction gives an introduction to the system and helps the user construct a mental picture of the system's configuration. In addition, menu choices may appear in several places, allowing frequently used entries to be displayed where they logically belong. In adapting TAE for a specific site, care must be taken to produce a good menu structure. However, once this conceptualization is complete, its implementation is relatively easy.

TAE provides mechanisms for establishing and formatting extensive on-line help and message information. Content, however, is particular to each system. Experience has shown that, unguided, programmers will typically write help information for applications functions in programming terms rather than in user terms. Experience has also shown that poor help information can frustrate and discourage a user, even in an otherwise well designed system.

Host Commands: Without replacing the host computer operating system, TAE overlays it with an environment tailored to a specific application. TAE insulates the user from the operating command language and error messages. However, both the user and the software developer may want to use certain functions in the operating system command language. Rather than reinvent these commands, TAE allows fast access to them to complement the portable commands in TCL, while retaining the TAE environment.

Host commands can be executed interactively or they can be "captured" in a proc to isolate the user from them. Procs containing host commands may be entered into a TAE menu in the same way as any other proc. Entire menus of host commands may be thus created with no applications code present. While this gives more power and flexibility to the system builder, procs containing host commands are in most cases not portable.

Prototyping: A particularly valuable feature of TAE is the ability to prototype a system. An entire functioning model of the system -- with all menus, proc definitions, help files and global variables -- can be built and run by the user without writing a single line of applications code. Users can actively use and easily and quickly change or reconfigure the system by editing text files. Participation at this level may lead to experimentation in the appearance and functioning of the system, and should increase the likelihood of building a successful system. The prototype ultimately serves as a detailed model to guide the implementers in the development of the applications code.

Modularity/Extensibility: TAE is an open system. It allows easy growth, incremental development and addition of new functions. Both procedures and processes may be added dynamically at any time by simply adding a PDF file to one of the libraries, including the user's private library. A menu can be updated quickly by editing the appropriate text file.

The power and task orientation of TCL encourages programs to be small and specific. Procedures may then be written to treat the processes as "subroutines," with input and output arguments. With an appropriate library of specific function applications, TCL becomes a higher level applications oriented programming language in which new applications procs can be created rapidly.

Portability: Portability is a key consideration in the development of TAE. All system dependencies are isolated from the portable code, which comprises approximately 85% of the total. The remaining code is machine specific and all or part of each routine must be reimplemented on each new host.

Experience with TAE and its prototype in three prior ports indicates that a typical minicomputer porting of TAE requires 4 to 5 person-months. TAE does not address portability problems of applications programs that may be associated with differences in compilers, word length or address space. However, most applications can be ported between heterogeneous TAE systems if they adhere to a standard like FORTRAN 77, use TAE service routines, and avoid operating system interactions. (Rigid adherence to such standards can be difficult in practice.)

Layering: Layering is a method of customizing the system to individual needs. Layering for a user may be done in three different ways: (1) Creation of unique utilities and applications. (2) Using the TAE command language to create higher level or customized procs that either do a larger, more complex task or provide a simpler interface that hides the existence of some parameters from a user. (3) Creation of new intrinsic commands within the TAE Monitor. These would likely be functions which must respond very quickly or special operations on particular kinds of data.



## 4.0 Documentation

A variety of TAE documentation is available in hardcopy form:

- "Conceptual Design"
- "User's Reference Manual"
- "Application Programmer's Reference Manual"
- "System Manager's Guide (VAX and UNIX)"
- "Utilities Reference Manual"
- "Guidelines for Designing Menus and Help Files"
- "Primer"
- "Functional Specification"
- "System Internals"
- "Display Management Subsystem"
  - "Functional Specification"
  - "Programmer's Reference Manual"
  - System Programmer's Reference Manual
- "VAX/VMS Release notes"
- "VAX/UNIX 4.2BSD Release Notes"
- "SUN/UNIX 4.2BSD Release Notes"
- "RCJM Prototype for TAE: VAX/VMS Release Notes"

The user's, programmer's, system programmer's and utilities manuals are included as text files on the TAE delivery tape.

## 5.0 TAE Support Office

The TAE Support Office (TSO) was established to ensure the availability of information about TAE and to assist users and developers in resolving problems. The TSO staff give tutorials in programming and in using TAE. They receive problem reports from user sites and prepare responses. They help organize the annual TAE Users' Conference and issue a TAE newsletter three times a year.

## 6.0 Applications of TAE

When TAE was delivered as a prototype in late 1981, it was incorporated into three new systems at Goddard. The operational TAE is seeing a much broader use. A few of the larger systems are:

- Land Analysis System (LAS) - general image analysis, Thematic Mapper data processing, earth resources, geographic information system
- MultiMission Image Processing Laboratory (MIPL) - general image analysis, planetary image processing
- General Meteorology Package (GEMPAK) - graphics and gridded data analysis, gridding and analysis of surface and upper air observations
- Atmospheric and Oceanographic Image Processing System (AOIPS) - meteorology image display and navigation, and some analysis, RADAR, Stereo displays, Stereo cloud height analysis

Agencies of the U.S. Department of Defense are also developing a number of TAE-based systems. TAE software is in the public domain, with distribution through the Computer Software Management and Information Center (COSMIC) at the University of Georgia.

## 7.0 IMPLEMENTATION

This section describes some implementation details of general interest which have not already been addressed elsewhere within the paper.

With the exception of a small number of host-dependent primitive subroutines, the TAE Monitor (TM) is written in the C programming language. Standards for this coding are imposed for the sake of portability. They are described in the TAE Internals Manual. The subroutine libraries are also written in C, but are FORTRAN-callable through a bridge front end to each routine. The C versions are made available to C programmers.

Under VAX/VMS, processes and DCL commands are run in a permanently active, spawned subprocess. Parameters and messages are passed between TM and a process by mailbox. Under UNIX, the vfork feature is used to run processes. A pipeline is established for passing parameters.

TM is essentially an interpreter and maintainer of a variety of tables which define the current contexts for a user, and a parser for user commands.

Three symbol tables, implemented as singly linked lists, define TAE command language (TCL) variables:

- a symbol table for global variables, accessible to any proc;
- a symbol table for local variables, accessible only within the context of a single proc;
- a symbol table for parameters passed to a proc.

Each variable is defined by a substructure, which is allocated dynamically as needed.

TM also maintains a data structure which defines the context of a procedure that is being interpreted. This contains, for example, the parameter and local symbol tables, pointers to the global symbol table, a table for the command qualifiers, nesting level, proc type, etc. Procedure invocation may be recursive, and is limited only by the amount of dynamic memory available for storing structures.

The structures of TCL intrinsic commands are defined by include files. Each intrinsic command has an associated function which processes that command. Any installation can create its own intrinsic commands by updating the include files and adding a function.

Parsing of a command line is done in a single pass, with substitution performed at the time a substitution operator is encountered. A parsing machine (Aho and Ullman, 1974) evaluates expressions. This machine also handles TCL and installation specific functions. Stacks are maintained for nested control structures -- for example, nested IF statements.

Special processing occurs for menu and tutor displays. Both are source driven from menu and proc definition text files. Menu processing

formats and presents menus, and maintains a stack of a user's menu path. It returns a "TUTOR procname" command when a menu selection is a proc. Menu commands are handled within the menu processor module. Menu contents and tree structure are defined through the menu definition files. Tutor processing builds memory-resident structures from PDFs to assist in the display and processing of subcommands, parameters and help information.

## 8.0 CONCLUSION

The increasing costs of development are driving diverse system developers to pursue common solutions to meet their common requirements. The upgrading of systems to accommodate new technology and to escape the costs of maintaining older systems has become a standard operation in environments which support changing user requirements and communities. As a flexible, open and portable system, TAE is a powerful tool in making such transitions cheaper and less formidable.

TAE is also proving to be highly adaptable to system developers with diverse objectives. The rapid increase in the number of TAE installations supports the original thesis that "...TAE will be a multi-purpose interactive executive...intended to serve as a common foundation for future systems development...." (Howell et al., 1980).

Current plans call for TAE to be upgraded to a network operating system command language for distributed processing between TAE-based systems in a local area network. Expansion of the display management subsystem to handle groups of images and to incorporate standard graphics processing is also planned.

## 9.0 ACKNOWLEDGEMENTS

TAE is being developed by the Image and Analysis Center, NASA/Goddard Space Flight Center and by Century Computing, Inc. The work is sponsored by the NASA Information Systems Office, which is part of the Office of Space Science and Applications.

## 10.0 FOOTNOTES

1. Examples of screens in this paper are taken from the Goddard Space Flight Center's Land Analysis System.

## 11.0 REFERENCES

AHO, A.V. and S.C. Johnson, "LR Parsing", ACM Computing Surveys, June, 1974, pp. 99-124

ANSI, "Operating System Command and Response Language (OSCRL) Language Specification (DRAFT)," Rev. 18, ANSI X3H1/09-SD, January 1984

Beech, D., "What is a Command Language?" in

Command Language Directions, D. Beech, ed., North-Holland, 1980

Carlson, Patricia, "User-Programmer Dialogue: Guidelines for Designing Menu and Help Files for Interactive Computer Systems", NASA TM-84980, 1983

Dalton, John, et al., "Interactive Color Map Displays of Domestic Information", ACM Computer Graphics, Volume 13, No. 2, 1979

Demers, Richard, A. et al., "System Design for Usability", Communications of the ACM, Volume 24, No. 8, August, 1981.

Dalton, John, et al., "The Visible and Infrared Spin Scanning Radiometer (VISSR) Atmospheric Sounder (VAS) Ground Data Systems", Society of Photo-Optical Instrumentation Engineers Technical Symposium, April, 1981

desJardins, Mary, and Ralph Petersen, "GEMPAK: An Interactive Display and Analysis System", Proceedings of the 9th Conference on Aerospace and Aeronautical Meteorology, 1983, pp. 55-59.

Engelberg, Norman, and Charles Shaw, "Considerations of Command and Response Language Features for a Network of Heterogeneous Autonomous Computers", NASA TM 86089, 1984

Howell, David, et al., "Conceptual Design for a Transportable Applications Executive," GSFC internal document, 1980

Ling, Robert, General Considerations on the Design of an Interactive System for Data Analysis", Communications of the ACM, Volume 23, No. 3, March 1980, pp. 147-154

Moran, Thomas P., "An Applied Psychology of the User", ACM Computing Surveys, Volume 13, No. 1, March, 1981

Perkins, Dorothy, et al., "A Device Independent Interface for Image Display Software", Proceedings of the National Computer Graphics Association Conference, 1984

Shneiderman, Benjamin, Software Psychology: Human Factors in Computer and Information Systems, Winthrop Publishers, 1980

Bracken, P. A., et al, Atmospheric and Oceanographic Information Processing System (AOIPS) System Description, NASA Publication X-933-77-148, March 1977.



## Mini-disaster Prevention Planning for the VMS System Manager

Marisa Riviere  
2407 Irving Ave. South  
Minneapolis, Minnesota 55405

### Abstract

The security and reliability of a computer system needs more "behind the scenes" preventive preparatory work than is usually assumed. This paper addresses the use of standard VMS tools to carry out such preparation. Some of the concepts described here cover general system maintenance issues that can also apply to other systems.

### VMS mini-disaster prevention planning

When observing a running computer system, its quality can be appreciated easily. It is seen, for example, in the turn-around time for tape mounts and printed output, and in the response of the interactive jobs. One would like to assume that a good system is reliable and secure, but that may not always be the case. In general, one could say that a safe and reliable system can protect itself against any possible disaster. For example, floods, hurricanes and fire can destroy a system. A good insurance policy, which should include a backup mainframe available on another site, should be part of the system maintenance planning. This will not be enough, however, if the system manager does not complement the insurance policy with off-site storage of all the needed data files to reconstruct the operational system. This should include dumps of on-line data as well as copies of important system and user tapes containing off-line stored materials.

A major disaster may never strike most computer systems, but other things, not as drastic, will certainly happen now and then. The computer room air conditioner may stop functioning or a power failure can take place. If the equipment is improperly turned off, or if it is left exposed to heat or power fluctuations, costly and delaying repairs may be needed. Brief and handy notes for the operators describing how to turn off the equipment should be available for this kind of emergency situation.

Leaving aside major or uncontrollable events which an operating system should be ready to deal with, we encounter other more routine issues that have to be considered as well when pursuing security and reliability. How those events are handled depends largely on the skill of the system maintenance team and, in addition, on the skill of a "devil's advocate" to direct the team's attention towards them. This paper attempts to play that role and, focusing on VMS, addresses some of those issues. One should bear in mind, however, that full security and reliability may be unachievable.

### "The system does not boot"

Sometimes it may happen that, if an I/O operation is in progress on the system pack at the time of a power failure, the pack will be improperly closed and will not be a good booting device any longer. The same applies to the console medium. Either a power failure or any other factor may produce a phone call to the system manager's office (or home) with the news that "the system does not boot" and "it does nothing". One of the reasons why the system does not boot may be problems with the console medium. An updated backup copy of the console medium should be available. Generally, the console medium is updated when software upgrades require it and when the system device changes. The booting files on the console medium are equipment dependent. They refer to the system configuration by hardware values. If an updated copy is not available, it may still be possible to boot the system by using an older copy and manually entering the appropriate console instructions. For that, familiarity with the equipment configuration and booting file instructions are needed. It may certainly help to list the booting files ahead of time to understand their instructions and, of course, to keep the list handy. It may also help to keep a copy of the current console medium on line in case it has to be reconstructed.

The console medium may be right, but the system may continue acting dead after all the "booting clicking noises" were made. Messages about the system pack being "improperly closed or formatted" may show up on the operator's console. There are several remedial alternatives for that situation. They depend on the system configuration and, of course, on some work done ahead of time.

If there is only one disk drive on the system, Stand-alone Backup may be the only alternative for a damaged pack. Operator instructions should be available describing the use of Stand-alone Backup. The notes should include samples of the Backup requests and how to answer them. Stand-alone Backup is a mini-system that loads on the computer from the console media. Its loading time may seem surprisingly long for any one who has not worked with it before. Timing information should be included in the notes.

The Stand-alone Backup kit depends on the system hardware configuration and on software updates. A new version has to be made each time it is required by the instal-

lation of a software update and when the system device or the tape drives are replaced by others of a different hardware type. An old version of Stand-Alone Backup may not be able to read tapes written by a newly released Backup utility, or, if it is, it may not be able to take full advantage of new features. The Stand-Alone Backup system should be tested whenever possible. A duplicate copy should be made for additional safety.

If there is a free removable pack device an old system pack can be mounted there. Booting the old system will allow the use of the Analyze utility to try to repair the damaged pack. On systems without removable drives a small version of a bootable system can be created ahead of time on one of the user file devices. This system will be much smaller than the current running system. It does not need compilers, help libraries, accounting files, etc. This system should contain only enough software to mount and dismount the system pack and to run the Analyze utility.

Should the system pack be irreparably damaged, updating the backup copy of the system pack can be a faster alternative to Stand-alone Backup. Incremental Backup dump tapes will be needed. If incremental dumps are not taken frequently some important data may be lost. Examples of that are the validation file, software developments, configuration descriptions, etc. Attention should be paid to what may be important to preserve between dumps. The "new changes" could be, for example, backed up on another pack or on tapes. A copy of the validation file can be placed at short time intervals on a different device.

### **The Backup utility**

Since a damaged system pack may eventually require to be rebuilt with the use of the Backup utility, we will address here some of the issues that are relevant when using this utility to take dumps for general file preservation. The frequency of the full dumps and of the incremental dumps may differ for each system, but weekly full dumps and daily incremental dumps can be ideal in most cases. At least two versions of old Backup tapes should always be available. A set of tapes should not be re-cycled until a new set is written. If there is a damaged spot on a tape the files from that area can always be recovered from an older tape. Some old Backup sets should also be kept for longer intervals of time. Keeping sets of tapes on a monthly, quarterly and/or yearly basis can help to recover files that have been intentionally but erroneously deleted. Incidentally, those monthly or quarterly backup tapes could also be used for off-site storage.

DCL procedures should be prepared for backing up and reloading user and system disks. The backup dumping and reloading procedures should specify when Backup qualifiers such as /IMAGE, /RECORD, /OWNER\_UIC, /SINCE, /INCREMENTAL, /OVERLAY, or /NEW\_VERSION should be used, and take care of the appropriate validations that may be needed for the job, such as BYPASS, EXQUOTA, and VOLPRO. The reloading procedure, for example, can ensure that no active data packs will be altered during the reloading process by mounting the target device right before the Backup command, request the loading of the incremental tapes and, if needed, rebuild the Quota file. When a pack of user data is damaged and needs to be rebuilt is not the best time to go searching for the right combination of Backup parameters and DCL statements!

### **"The system boots but..."**

The system may boot, but suddenly, the startup procedures may abort. There is a lot of activity in progress when these procedures run. Users' packs are mounted, devices are initialized, drivers are loaded, hardware is connected, system definitions are made, network communications are brought up, etc. The startup procedures may stop on any one of these tasks. This may happen when hardware components are not available or are functioning badly. The booting procedures should have some messages to inform the operator as the different booting steps are accomplished, such as "User packs mounted", or "Special drivers loaded". The messages should be informative but short. Long printouts from the startup procedures can produce paper jams if the system goes down and boots several times while unattended. A list of the startup procedures made in advance helps the operator to see where the problem may be. Should the problem not be easy to find, a conversational boot will be needed. Here again, the console media has to be ready for it. It has to contain files for the current system device. And, by the way, was the minimum configuration choice ever tested?

### **Human factors: system users**

Certainly, the ability to recover a damaged operating system may be the minimal safety required for its proper operation. That is not enough, however, to make the system as safe and reliable as one would like it to be. There is more organizational planning-ahead and managerial day-to-day work to do. This work deals with human factors. On an operating system, protection has to exist for the system against the users, the users against each other and the users against themselves. The same applies for system programmers and managers. VMS has very good features for protection against malicious users such as hashed passwords, file protection, validation restrictions and captive accounts. Even so, the user's capability to break into the system should never be underestimated. Users should be informed of the illegality of misusing accounts and asked for cooperation to inform the system manager of any possible system weakness. Caution should be exercised when granting to any computer user more than the standard system access validation. Usually, for most application packages the standard authorization parameters suffices.

User access to system software should be permitted only in execute and, where necessary, read mode. If possible, the system software should reside on a pack by itself, where a quota allowance is given only to system accounts. Users' root directories should be created, for example, with write permission granted only to the users and/or the group. Groups should be defined for people working on common projects, or for administrative reasons. Users should learn how to use the VMS file protection and be aware of group validation features when they are in groups where some of the members are entitled to privileges that can exercise control over the members. A quota value should be enforced on all multi-user disks. Users should accomplish maintenance tasks for their own protection, such as to observe and clean up their file's directories, to check their quota values and to set up their job's time limit. They should keep track of their sessions by saving accounting records at the end of each session as permanent files and checking them at login time. Passwords, of course, should be changed frequently.

## Human factors: system programmers

System programmers are supposed to work for the well-being of the operational system and its users, but it may not always happen that way. We will leave aside from this paper considerations about internal sabotage. Those are psychological and legal issues to be dealt with by the overall site management and not by an operating system alone. We should assume that system programmers will only create unintentional harm and see how their work and the work of the system manager should be organized to minimize that harm.

For example, system programmers need to use accounts with special privileges to access and alter system files. Those accounts should be restricted to acquire special privileges only as they are needed, and to do it within captive DCL procedures. Such accounts should enforce a secondary password and be allowed to log in only on specific (e.g., hard-wired) terminals.

Several members of the system programming team may need to work simultaneously altering software across many system directories. The system manager should design the organization of the system directories. The organization should be suitable for easy searches, off-line backups and for the tracing of changes to procedures and source files. A well-planned design for system software directories becomes more and more valuable as time passes and the software production becomes larger. Some rules should apply to the internal structure of the system directories and subdirectories. For example, each directory and subdirectory could contain a file with a brief description of the function of the other files that reside there. Another file should contain a description of all the changes made in the directory. Scratch intermediate files should not be left in the system directories. After a change, the altered directories should contain only two versions of files, the new versions and the versions that were used prior to the change. System software should be developed in a location separate from the production system directories. Logical names could be used to reference the location of all the files that are used by those procedures and programs. This makes the software easily transportable for testing.

Great coordination is required on the system team to schedule and organize system upgrades and modifications in a satisfactory manner. Changes and additions should be consistent in style and quality. Not all system programmers have all the same good qualities that a system manager would like for every one of the team's members. The system manager should compensate for that by assigning work, when possible, to more than one person. Each person can act as a moderator for the others. Frequently, bright programmers turn in a large and prompt production of undocumented software. Those products suit their purposes well until someone else has to make a change. More conservative and not so impressively bright programmers tend to be consistent in style and neatness, mainly for their own sake. Grouping people with complementary work characteristics can produce software that will be easily understood by others. Software changes have to be tested at appropriate times and with proper user notification. Changes which affect users and operators should be clearly documented ahead of time and the date of the change announced properly. System notes and a directory of system documents and user notices can be used for users' information.

## Human factors: system manager day to day

The system activities should be monitored in a routine manner. Daily maintenance runs should produce snapshots of changes in system data, such as quota files, system parameters, validation, free disk space, hardware configuration, startup procedures, login files, error reports, etc. The daily printouts should be brief, so they can be checked fast and at once. Weekly reports should be made of important system data and statistics. Lists of startup files, quota files, brief accounting information, protection and size of user root directories and operator procedures, should be printed or placed on tape or microfiche at a certain frequency. Snapshots of data on the "live" system status should be included. The reports can help to trace back the reason of many obscure problems that may eventually appear.

When the system crashes, the system dump should be analyzed at the next booting time. A printout of the dump can be included in the startup procedures. The printout should be made on a printer, not on the console. The error log should be monitored for sudden increases in size. If that happens, it should be analyzed immediately. It may possibly contain information on a partially failing device. Detected in time, the problem can be taken care of before it brings the system down, or a proper shutdown, if needed, can be scheduled. Maintenance procedures, such as accounting runs and system reports, should be carefully written. The procedures should take care of cleaning up temporary files and old versions of reports. They should also take care of protecting any information which should not be made public.

The installation of new software should be carefully documented. It will help, later on, if the software has to be removed. New software may affect several directories and sections of libraries and documentation. It may happen that, eventually, those sections may be shared by other products. Keeping track of installation changes will help if the software has to be deleted in the future. Images to be installed with special privileges such as KERNEL or PHY\_IO should be carefully screened if they are not provided from a reliable source. Additions to startup and system login files may include symbol definitions that can overlap with existing system symbols. Those procedures should be carefully screened before they are used. Changes in non dynamic parameters should be tested immediately. Some Sysgen parameters are relation dependent on each other. A change in one may require changes in others. Any change in non-dynamic parameters should be followed by a test boot operation.

A description of other possible system manager mini-disaster traps could continue for several pages or forever. Every reader may have additions based on inquisitive test runs or on bitter experience, but the information here should be enough to direct system managers' attention to observe their own systems' needs for reliability and security.

Smile! Your system may be running...and well.



Richard H. Warner  
Measurex Corporation  
Cupertino, Ca 95014

This session will describe how DCL can be used as a high level programming tool. As an example, I will describe a sophisticated set of DCL procedures which were written to enable S/W integrators to build Process Control S/W Systems.

The procedures are designed so that all high CPU usage is done in batch jobs at lower priority. The interactive procedures check fully for file existence, and dates. The integrators have an opportunity to correct most errors before the batch jobs are submitted.

The set of procedures consists of:  
(12) interactive procedures, averaging 300 lines each.  
(6) batch procedures, averaging 200 lines each.  
(10) utility procedures, which are used 150 times.

The batch procedures utilize MAC, and TKB Utilities, and (2) user written FORTRAN programs.

The session notes will include examples of:  
. effective standards for all procedures  
. formatted screen outputs  
. often used utility procedures

Welcome to the session on Advanced DCL Programming!

It has been a very challenging task, and a very satisfying experience getting prepared for this presentation.

Everyone here should think very strongly about presenting a paper at DECUS. Nearly everyone should have some topic which would be of interest to DECUS members.

I had been thinking about presenting a paper, but not very seriously. At the Fall meeting in Anaheim, a question from the audience triggered my decision to do it.

The question as I remember it was about like this: "Does a person who writes DCL command procedures qualify as a programmer?" Implicit in that was another question: "Is DCL a programming language?"

My answer to both is a resounding "Yes!"

I have been writing DCL command procedures for over 5 years, and I have extensive experience in FORTRAN and assembly language programming. I find that DCL is truly capable of performing powerful programming tasks.

There are areas where it lacks "bells and whistles", and in some cases it requires more programming skill than FORTRAN.

Here is how we use it at Measurex.

Some background on what Measurex does

Measurex builds computer process control systems for manufacturing facilities. For the purposes of this presentation, the system will be a system based on a DEC 11/23 processor for control of a Vinyl Calendar mill.

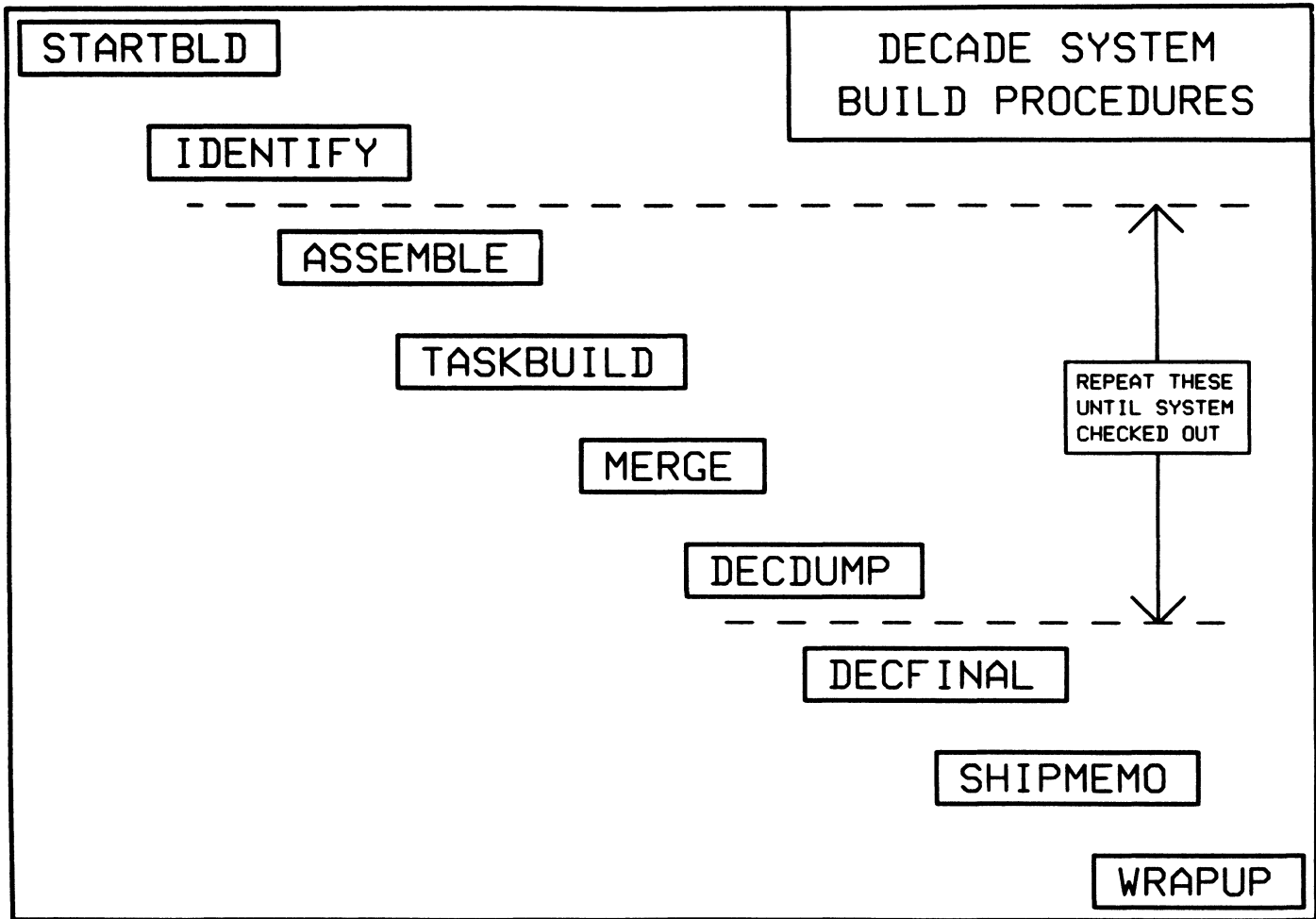
The Measurex hardware system consists of a set of scanning gauges which pass over the Vinyl sheet as it is being produced. When one of the gauges detect an abnormal or out-of-target condition, the control system adjusts the input variables to return the Vinyl to the proper specifications.

The Measurex software system consists of 2 parts.  
. system programs which are constant for a product line  
. databases and control functions specific to a user's facility

The system integrators adapt a conditional file to meet the customer's requirements. This conditional file is used by the DCL Command procedures to create database listings, system Maps, and an executable file on a floppy diskette.

The procedures assemble modules using RSX MAC, task build using RSX TKB, then merge the .EXE files into a total system .EXE file. The total system .EXE is dumped to 1 or more floppy diskettes.





How the Measurex system software is built:

There are 9 basic procedures which the integrators use, very briefly this is what they do:

STARTBLD - sets up logical assignments in users' LOGIN.COM

IDENTIFY - creates a .DAT file which is sent to a S/W shipping clerk. This alerts the clerk of an upcoming system.

ASSEMBLE - submits batch jobs to run MCR MAC on desired .MAC files.

TASKBUILD - submits batch jobs to run MCR TKB on .OBJ command files.

MERGE - submits a batch job to merge all of the .EXE files into a full system.

DECDUMP - dumps the system .EXE onto 1 or more floppy diskettes.

Note: the above 4 procedures are repeated until the system is checked out.

DECFINAL - rebuilds the entire system, creates multiple floppy copies, listings, and creates a microfiche tape.

SHIPMEMO - allows the S/W integrator to document information specific to this system.

WRAPUP - puts the completed software onto archive tapes and removes all files from working directories.

These processes are CPU intensive and are run in Batch mode. The batch processing requires good Command Procedures to be effective.

The interactive part of the command procedure set contains an enormous amount of file and error checking. If there is anything missing or in error, the Batch job is not submitted. The user has an opportunity to fix the problems now, not hours or days later when the batch job failed. In addition, the extensive checking means that re-runs of batch jobs are reduced. The net effect is more efficient use of the computer resources.

The S/W integrators soon get into a nice production groove. Find a S/W bug, edit the affected source file, send a batch job off to re-assemble the file. Find another bug, fix it, submit another batch job, etc. After all the known bugs are found, then they re-assemble everything that needs it.

The extensive use of batch processing means that the user terminals are not tied up with jobs which are CPU intensive. In fact, normally the S/W integrator's terminals are "OFF".

## SAMPLE ASSEMBLE INTERACTIVE SESSION

- - ASSEMBLE - -

THIS PROCEDURE WILL ASSEMBLE SOURCE (.MAC) FILES,  
AND CREATE .OBJ AND .LIS FILES.

AFTER YOUR BATCH JOB, YOU MAY CONTINUE WITH TASKBUILD, ETC.

YOU MAY SELECT INFORMATION ABOUT THE FOLLOWING:

- |                               |                                       |
|-------------------------------|---------------------------------------|
| (1) DECxxA.CNF AND .ASM FILES | (6) BATCH JOB .LOG FILES              |
| (2) ASSEMBLE OPTIONS          | (7) RULES FOR "NEED TO BE ASSEMBLED"  |
| (3) PROCEDURE DEFAULTS        | (8) ASSEMBLE OF NON STANDARD FILES    |
| (4) BATCH PROCESSING          | (9) USE OF ALTERNATE SOURCE DIRECTORY |
| (5) CHAINING BATCH JOBS       |                                       |

Enter Information Number, or Zero to End Information: 0

Do you want to ASSEMBLE individual .ASM files[YES/NO]?: NO

Why our command procedures are effective:

The command procedures have been designed to be very user friendly. They feature such things as screen clearing often, so that as information is presented it writes down instead of scrolling up. All questions and presentations are standard in format. Error and control\_y handling has been included in all procedures so that they cannot abnormally terminate leaving a mess of open files, incomplete data, or submit a batch job which will fail anyway.

Some of the command procedures can be run 3 ways:

1. Normal mode: a preamble about the procedure is put on screen. Before each question, enough information is presented to enable the S/W integrators to make the correct decisions.
2. Novice mode: the S/W integrators have an option to display up to 10 screens of information documenting the use and features of the procedure.
3. Expert mode: screen clearing is eliminated, questions are not preceded by information, and other features are eliminated to allow the user to get through as quickly as possible. All error and file checking, however, is still done.

The command procedures have been written to be very general. We have to support over a dozen different industries, each with different system programs and databases. In order to accomplish this, all of the procedures are file-driven.

The entire system build has been modularized into small files and libraries which are then combined into 1 assemble task (MCR MAC). The object modules and libraries are combined into a list for use by the task builder (MCR TKB @xxx).

For each industry, there is a set of configurator files. These define the names of all files needed to be assembled, task built, and merged to create a complete system.

The assemble and taskbuild requirements are defined by files which are then unique to an industry.

By using the file-driven concept in the writing of the procedures, the procedures now became general purpose, and they can handle all of our different industry requirements.

The other feature of the general purpose procedures is the use of logical names pointing to specific industry master files. When a system is started, a procedure is executed which defines the master directories for the industry. For example:

```
. MX$$ = Sources - MDSA: [MXVINYL.SOURCE.B202]
. MX$0 = Objects - MDSA: [MXDECADE.OBJECT.B201]
. MX$d = Drivers - MDSA: [MXVINYL.DRIVER.B201]
```

The use of the logicals makes the procedures much simpler and allows 1 set of procedures to be used for all industries.

The procedures are designed so that the user can chain them together. When the user selects files to be assembled, the user may elect to have the batch job submit another batch job to do the next step, ie, task build. Similarly, the user may elect to merge the .MAP files, and dump the system .EXE to a floppy. The batch procedures continue chaining unless a fatal error occurs.

Since the procedures are batch oriented, the .LOG files become very important. The batch procedures are run with SET NOVERIFY so that they are short. Whenever an important event occurs, it is carefully documented in the users' .LOG file. The batch job and .LOG file names use the system number for identification. The computer operators track jobs with these IDs.

YOU MAY ASSEMBLE:

ANY OF THE FILES SHOWN BELOW, ALLOW THE PROCEDURE TO SELECT ALL THAT NEED TO BE ASSEMBLED, OR ASSEMBLE ALL OF THEM IN ONE BATCH JOB.

| # | FILE NAME | DESCRIPTION            | #  | FILE NAME | DESCRIPTION         |
|---|-----------|------------------------|----|-----------|---------------------|
| 1 | COMMV4    | COMMON                 | 8  | KRNLV4    | KERNEL DATABASE     |
| 2 | CTLV4P    | CONTROL B/C WINDOW     | 9  | LOGV4C    | LOGIC ORDINAL TABLE |
| 3 | DIGV4P    | DIAGNOSTICS B/C WINDOW | 10 | MISV4P    | MIS B/C WINDOW      |
| 4 | FRMV4P    | FRM B/C WINDOW         | 11 | USERV4    | USER COMMON         |
| 5 | FXV4DB    | USER FEX DATABASE      | 12 | ZZKRZE    | KERNAL PATCH FILE   |
| 6 | GAGV4P    | GAUGING B/C WINDOW     | 13 | ZZCOZE    | COMMON PATCH FILE   |
| 7 | GDRV4P    | GDR B/C WINDOW         |    |           |                     |

YOU NOW           1 - Assemble individual files from list  
HAVE 3            2 - Assemble all files from list which need it  
OPTIONS           3 - Assemble all files in list

Enter Number of Option[1:3]:

The ASSEMBLE and TASKBUILD procedures have been designed to give the S/W integrators 3 options. For example, when assembling files they may:

- . assemble one or more files by entering specific file names
- . assemble all files required by the system
- . assemble all files which need it (ie, source or a component file edited)

The concept of assemble or task build "everything that needs it" has three very important features:

- . the S/W integrators do not have to keep track of changes to various modules
- . by using "everything that needs it", the CPU usage is reduced because less files are processed each time
- . it is "safe" and always produces a correct system every time

\* GENERAL STANDARDS FOR ALL PROCEDURES \*

START PROCEDURE WITH NAME, AUTHOR, REVISOR, AND DATES  
PUT A 1 OR 2 LINE DESCRIPTION AT BEGINNING  
DOCUMENT PROCEDURES REALISTICALLY - IT MAY BE ALL THERE IS  
DO NOT OVER-DOCUMENT, TOO MUCH WILL MASK MEANINGFUL COMMENTS  
IF INPUT PARAMETERS ARE USED, DOCUMENT THEM AT BEGINNING  
USE MEANINGFUL SYMBOLS FOR VARIABLES AND LABELS  
AVOID OBSCURE COMMAND STRING EQUIVALENCES, SPELL IT OUT  
DO NOT ABBREVIATE COMMANDS OR QUALIFIERS  
COMMENTS AT BEGINNING OF LOGICAL PROCEDURE SEGMENTS  
DO NOT PUT COMMENTS ON A DCL COMMAND LINE  
DO NOT COMMENT WITHIN HIGH USAGE LOOPS

For the following sections, I have an outline of the coding "standards" which I have adopted. These have proven to be very effective for my command procedures.

Following the "header" are actual examples from my procedures. There is a little problem in synchronizing the two, so some flipping back and forth may be necessary.

```

$! R$:ASSEMBLE.COM R.H.WARNER 27 FEB 85
$!
$! COMMAND FILE TO ASSEMBLE BLOCK CHAIN WINDOW OR COMMON FILES.
$! THE OUTPUT IS ALWAYS A .LIS AND A .OBJ FILE FOR EACH INPUT.
$!
$! IF P1 IS "EX" (FOR EXPERT MODE) THEN CUT OUT SOME SCREEN PROMPTS
$!
$ OCS:VERIFYUSR
$ IF '$STATUS .EQ. 3 THEN EXIT
$ ON CONTROL Y THEN GOTO ABORT
$ USRDIR := 'F$LOGICAL("SYS$DISK")' 'F$DIRECTORY()'
$ END := ""
$ ERRORS = 0
$ OPTION := "INDIVIDUAL_"
$ QUEUE := "MX_ASMB"
$ SYS_NUM := 'F$LOGICAL("MX$SYSNUM")
$ IF "'_SYS_NUM'" .EQS. "" THEN GOTO NUM_ERR
$GOT NUM:
$ OPEN/READ/ERROR=NO_LBLMSTR TEMP LBL'_SYS_NUM'.MAC
$ CLOSE TEMP
$CONTINUE:
$ EXPERT := "NO"
$ IF P1 .EQS. "" THEN GOTO NO_P1
$ IF ("'"F$EXTRACT(0,1,P1)'"') .NES. "E" THEN GOTO NO_P1
$ EXPERT := "YES"
$ GOTO START
$NO P1:
$ RUN ES:CLEARSCRN
$START:

```

The following are items that I feel are important, and highly recommended are:

Always start the procedure with procedure name, modifier's name, current revision date. If it was written by someone else, add a line telling who the original author was, date, etc.

Put a 1 or 2 line description of the procedure at the beginning.

Document your procedures realistically. Very seldom are our command procedures documented from a programming point of view. We tell the users how to use them, etc, but the "maintenance" programmers are out of luck. Therefore, any documentation that you put into a procedure is invaluable.

Do not over-document! Too many comments may mask the important and significant comments. It is like crying "Wolf!".

If input parameters are used, document these at the beginning.

Variables and labels should have meaningful names. I have found that names between 4 and 8 characters long work best.

Initialize variables at the beginning of the procedure. This is very valuable when adding commands later which could be out of the original sequence.

Labels should be left justified. Be generous in label names, they should be meaningful. Use underscores freely.

Command lines should be indented 2 spaces after the \$ sign. Continued lines should be "tabbed" in 8 spaces.

Variable equivalences using "=" and ":=" are more easily read when they are surrounded by blanks.

At the start of a procedure, clear the screen, and present a screen full of information about the use of the procedure.

I also clear the screen before presenting information to the user. It is easier to read text from the top down, than to read text which is scrolling up.

I avoid putting comments on a valid command line. Usually this makes a procedure hard to follow logically. I think that this clutters the listings and is not that important if meaningful names are used.

Avoid creating obscure command string equivalences. For example:  
 \$PHDC2QL:= "PRINT/HEADER/DELETE/COPIE=2/QUEUE=LONG"  
 is very obscure and misleading when it looks like  
 \$ PHDC2QL OUTFILE as a command.

Note: if it really bothers you to have redundant characters in a command line, find a nice loop which is executed many, many times and abbreviate as much as possible. Then when you have it out of your system, go back to full, meaningful names.

When you are searching through someone else's procedure to find a bug, or to make a change, then you will appreciate the redundancy.

```

$ IF EXPRT .EQS. "YES" THEN GOTO INPUT_LOOP
$ TYPE SYSS$INPUT

ENTER INDIVIDUAL FILE NAMES DESIRED AS: AAAAA.ASM

AFTER LAST FILE, THEN ENTER /END, OR /NOOBJ

(NOTE: /NOOBJ OVERRIDES DEFAULT OF .OBJ CREATED)

*** FILE CHECKING WILL TAKE TIME . . PLEASE BE PATIENT ***
$! -----
$! - - - USERS INPUT INDIVIDUAL FILE NAMES - - - - - INPUT_LOOP:
$! -----
$INPUT_LOOP:
$ ON WARNING THEN CONTINUE
$ WRITE SYSS$ERROR " "
$ INQUIRE FILE_NAME " 'NEXT'File name"
$ IF FILE_NAME .EQS. "" THEN GOTO INPUT_LOOP
$ LENGTH = '$LENGTH(FILE_NAME)
$ LOC_SLASH = '$LOCATE("7",FILE_NAME)
$ IF LOC_SLASH .EQ. LENGTH THEN GOTO CHK_BLANK
$ FILE_NAME := '$EXTRACT(0,LOC_SLASH,FILE_NAME)
$CHK_BLANK:

```

I find it very convenient to use TYPE SYSS\$INPUT followed by the text. In EDT editor keypad mode, the text duplicates the screen.

Use comment lines at the start of logical procedure segments. This makes it easier to scan through a long procedure and locate segments of interest.

I use "WRITE SYSS\$ERROR" for single line output. It must be used when symbol equates are used in a text line. I avoid SYSS\$OUTPUT because it may have been assigned to a file, or null device.

There are times when comments are harmful. In a high usage loop, for instance, it greatly increases the procedure overhead.

\* GENERAL STANDARDS FOR ALL PROCEDURES (Continued) \*

```

INITIALIZE VARIABLES AT BEGINNING, DO NOT SCATTER AROUND
LEFT JUSTIFY LABELS, USE UNDERSCORES FREELY
INDENT DCL COMANDS 2 SPACES AFTER DOLLAR SIGN
INDENT CONTINUATION LINES AT LEAST 8 SPACES
SURROUND "=" AND ":=" WITH A BLANK FOR READABILITY
CLEAR SCREEN BEFORE PRESENTING DATA TO USER
MAKE ALL PROMPTS AND QUESTIONS UNIFORM
USE "TYPE SYSS$INPUT" FOR DISPLAYING LONG TEXT
USE "WRITE SYSS$ERROR" WHEN SYMBOLS ARE TO BE OUTPUT
ALWAYS EXIT GRACEFULLY, ESPECIALLY CLOSE OPEN FILES
HANDLE ALL ERROR CONDITIONS AT END OF PROCEDURE

```

More general standards for all procedures. Some of the examples are shown best on previous slides.

As procedures get more complex, and much larger, it is very useful to break up the procedure into separate procedures.

```

$! -----
$! - - - ERROR HANDLING -----
$! -----
$NUM ERR:
$ WRITE SYS$ERROR -
 "MX$SYSNUM not assigned in LOGIN.COM, a temporary System Number needed"
$ QCS:GETNUMBER " Enter a temporary system number" 1111 9999
$ IF '$STATUS .EQ. 3 THEN GOTO ABORT1
$ _SYS_NUM := ' NUMBER'
$ GOTO GOT_NUM
$NO LBLMSTR:
$ WRITE SYS$ERROR -
 " * * * COULD NOT FIND LBL''_SYS_NUM'.MAC * * *"
$ QCS:YESNO -
 "Do you want to continue"
$ IF '$STATUS .EQ. 3 THEN GOTO ABORT
$ IF _YESNO .EQS. "N" THEN GOTO ABORT
$ GOTO CONTINUE
$ABORT1:
$ ON WARNING THEN CONTINUE
$ CLOSE CNF
$ABORT:
$ TYPE SYS$INPUT

 *** THE JOB WAS ABORTED, RE-RUN THE PROCEDURE ***

$ EXIT

```

Check for error conditions. Handle all error conditions at the end of the procedure, out of the logical flow. Always exit gracefully.

Remember to close any files which were opened. If they are not, then the next run of the procedure uses the file at its previous position at time of abort.

\* STANDARDS APPROPRIATE TO MODULAR PROCEDURES \*

```

PUT DUPLICATED CODE INTO A SEPARATE SUB-PROCEDURE
PUT INDEPENDENT SECTIONS INTO SEPARATE PROCEDURES
USE GENERAL PURPOSE UTILITY PROCEDURES WHERE POSSIBLE
RETURN VALUES WITH STATUS CODES OR GLOBAL VARIABLES

```

\* STANDARDS USED WHEN SUBMITTING BATCH PROCEDURES \*

```

CHECK ALL FILES USED IN A BATCH JOB FOR EXISTANCE
ALLOW USERS TO RUN THE BATCH JOB LATER OR OVERNIGHT
DISPLAY PARAMETERS OF BATCH JOB BEFORE SUBMITTING JOB
ALLOW THE USER TO CHANGE THEIR MIND, EXIT GRACEFULLY
SHOW STATUS OF QUEUE WHERE JOB WAS SUBMITTED

```

If a section of commands is used in more than one place, split it out to another procedure. Note that the new sub-procedure can have 8 parameters passed to it to handle variations. For example, I have a checking procedure which is called from several places in 3 separate procedures. By passing 5 parameters, the sub-procedure handles all of the calling procedure's requirements.

Independent sections lend themselves to separate procedures. I use an information section for novice users, and put it into its own procedure. This keeps it out of the logical flow, reduces the size of the original, and makes it easier to edit either one.

```

$ SAVE VERIFY = F$VERIFY() ! C$:YESNO.COM 14 SEP 83
$ SET NOVERIFY ! GET ANSWER TO QUESTION R.H.WARNER
$ ON CONTROL_Y THEN GOTO STOP
$REPEAT:
$ WRITE SYS$ERROR " "
$ INQUIRE YESNO "'P1' [YES/NO]?"
$! ! ! IF "YES", "YE", "Y", "NO", OR "N" IS NOT ENTERED THEN PROMPT AGAIN
$ IF (YESNO.NES."YES".AND.YESNO.NES."YE".AND.YESNO.NES."Y" -
.AND.YESNO.NES."NO".AND.YESNO.NES."N") THEN GOTO REPEAT
$! ! ! IF THE PROCEDURE REACHES HERE THEN A CORRECT RESPONSE WAS GIVEN
$ YESNO := 'F$EXTRACT(0,1,YESNO)
$ YESNO := 'YESNO'
$ IF SAVE_VERIFY THEN SET VERIFY
$ EXIT 1
$STOP:
$ IF SAVE_VERIFY THEN SET VERIFY
$ EXIT 3

```

Write general purpose command procedures to handle common tasks. For example, I have a procedure named YESNO.COM which is really great! There is a copy in the session notes. This procedure alone is worth your price of admission.

YESNO.COM is a really fine utility. It is easy to use, saves time, labels, and keeps the DCL code neat. As you can see, it is short, handles all normal and error conditions, and exits gracefully on control\_y. One of the things that I noticed while using it, is the ease of standardizing the questions presented to the users.

```

$ SAVE VERIFY = F$VERIFY() ! C$:GETNUMBER.COM 29 JAN 85
$ SET NOVERIFY ! PROMPT FOR NUMBER R.H.WARNER
$ ON CONTROL_Y THEN GOTO STOP
$REPEAT:
$ WRITE SYS$ERROR " "
$ INQUIRE NUMBER "'P1' ['P2':'P3']"
$ ON WARNING THEN GOTO REPEAT
$ CH1 := 'F$EXTRACT(0,1,NUMBER)
$ IF (CH1 .LTS. "0" .OR. CH1 .GTS. "9") THEN GOTO REPEAT
$ IF (NUMBER .LT. P2 .OR. NUMBER .GT. P3) THEN GOTO REPEAT
$ NUMBER == NUMBER
$ IF SAVE_VERIFY THEN SET VERIFY
$ EXIT 1
$STOP:
$ IF SAVE_VERIFY THEN SET VERIFY
$ EXIT 3
$! * * * * * DOCUMENTATION SECTION * * * * *
$!
$! THIS IS A GENERAL PROCEDURE TO GET NUMBER FROM THE TERMINAL
$!
$! THE TEXT TO BE PROMPTED IS PASSED BY PARAMETER P1
$! THE LOW RANGE OF THE NUMBER IS PASSED BY PARAMETER P2
$! THE HIGH RANGE OF THE NUMBER IS PASSED BY PARAMETER P3
$!
$! IT WILL REPEAT THE PROMPT UNTIL THE NUMBER IS ENTERED IS VALID
$!
$! _NUMBER IS RETURNED AS A GLOBAL SYMBOL
$!
$! IF CONTROL_Y ENTERED, A VALUE OF 3 IS RETURNED IN $STATUS

```

Other very useful procedures are GETNUMBER.COM, and VERIFYUSR.COM. Copies of these are also in the session notes.

Note that GETNUMBER handles all conditions, out-of-range, illegal alpha entries, and control y. This is another great time-saver.

Some key points about utility procedures: Values may be returned by using global symbols, or small odd integer values may be returned by EXIT status. I use this for handling control y interrupts.

If you use Global Symbols to communicate with other procedures, precede the name with a unique symbol such as "under score". One of the hardest things to find in debugging a procedure is a variable which is both global and local with different values.

If you have a lot of documentation, it is better to put it at the end of the procedure. The use of a separate page (form-feed), works well.

```

$ SAVE_VERIFY = F$VERIFY() ! C$:VERIFYUSR.COM 14 SEP 83
$ SET NOVERIFY ! CHECK USER WRITE PRIV R.H.WARNER
$ OPEN/WRITE/ERROR=WRONG DEF XX DIRECTORY.BAD
$ CLOSE XX
$ DELETE DIRECTORY.BAD;*
$ IF SAVE_VERIFY THEN SET VERIFY
$ EXIT 1 !NORMAL EXIT
$WRONG DEF:
$ WRITE SYS$ERROR " "
$ WRITE SYS$ERROR -
" *** YOU MUST BE IN YOUR OWN DIRECTORY TO RUN THIS PROCEDURE ***"
$ WRITE SYS$ERROR " "
$ IF SAVE_VERIFY THEN SET VERIFY
$ EXIT 3
$!
$! * * * * * DOCUMENTATION SECTION * * * * *
$!
$! MAKE SURE THAT USER HAS WRITE PRIVILEGE FOR PRESENT DIRECTORY
$!
$! IF DIRECTORY IS OK, EXIT WITH $STATUS = 1
$!
$! IF NOT, EXIT WITH $STATUS = 3

```

This little procedure comes in very handy in my procedures.

Our users often roam all around in various other

directories, especially in masters. If they forget to return to their own directory, they could get into trouble. This procedure will not allow them to write into other users directories.

```

$! - - - - -
$! - - - FINISHED WITH SUBMIT OPTIONS, SUBMIT JOB - - - - - SUBMIT1:
$! - - - - -
$SUBMIT1:
$ IF TIME .NES. "" THEN GOTO SUBMIT2
$ QCS: YESNO "Can the Batch Job be run after 8 PM"
$ IF '$STATUS .EQ. 3 THEN GOTO ABORT3
$ IF YESNO .EQS. "Y" THEN TIME := "/AFTER=20:00"
$SUBMIT2:
$ WRITE SYS$ERROR -
" ***"
$ WRITE SYS$ERROR -
" *** JOB 'SYS_NUM'ASM WILL BE SUBMITTED TO THE ***"
$ WRITE SYS$ERROR -
" *** 'QUEUE' QUEUE TO ASSEMBLE YOUR FILES ***"
$ IF TIME .NES. "" THEN WRITE SYS$ERROR -
" *** 'TIME' ***"
$ WRITE SYS$ERROR -
" ***"
$SUBMIT3:
$ WRITE SYS$ERROR " "
$ INQUIRE CR -
"THE PROCEDURE HAS PAUSED; HIT RETURN TO SUBMIT JOB; CONTROL_Y TO ABORT"
$ SUBMIT/NOTIFY/KEEP/QUE='QUEUE'TIME'-
/LOG FILE="''USRDIR'' SYS_NUM'ASM/NAME=' SYS_NUM'ASM-
RS:ASSMBA/PARAMETERS=(' SYS_NUM', ''USRDIR'', -
''TEMP FILE', 'DEFAULT', ''PRT'', 'TASK', 'MERGE', 'FLPY_LABEL')
$ SHOW QUEUE/ACL 'QUEUE'
$ EXIT

```

Procedures which submit batch jobs require special features. Some of my coding "standards" appropriate to these are as follows.

Check to make sure that all files used by batch jobs exist. If there is a missing file, the user can resolve the problem before submitting the job. Do everything you can to reduce failures.

At the end of procedure, allow the user to elect to run the job later, or at night. This allows the batch job load to be spread out more evenly.

Display parameters before submitting jobs. Show users what will happen, and then give them the option of not submitting the job. Make sure that the procedure exits gracefully at this time.

Do not abbreviate commands or qualifiers. If a default qualifier is important to the function of the procedure, include it. This protects against a user overriding defaults, future changes, and also helps to document the function.

Do a SHOW/QUEUE/ALL queue to show the status of the batch queue. This is a real convenience to the user.



\* STANDARDS APPROPRIATE TO BATCH PROCEDURES \*

-----

EXIT IF PROCEDURE IS NOT TO BE RUN INTERACTIVELY  
 SET NOVERIFY TO KEEP .LOG FILE AS SHORT AS POSSIBLE  
 PRINT OUT PARAMETERS USED TO THE .LOG FILE  
 ALLOW USERS TO RUN BATCH JOB USING SUB-DIRECTORIES  
 IF DEFAULT IS CHANGED, SHOW WORKING DEFAULT DIRECTORY  
 PRINT OUT ANY SPECIAL LOGICAL ASSIGNMENTS USED  
 PRINT OUT ANY SIGNIFICANT EVENTS OF INTEREST TO USER  
 IF PROCEDURE SUBMITS ANOTHER JOB, DESCRIBE THE EVENT  
 HANDLE ERROR CONDITIONS CAREFULLY, EXPLAIN TO USERS

```

$ IF "'F$MODE()'" .EQS."INTERACTIVE" THEN GOTO START
$ SET VERIFY
$! R$:ASSMBLBA.COM R.H.WARNER 21 FEB 85
$!
$! ASSEMBLE LIST OF DEC SOURCE FILES FROM .TMP FILE
$!
$ SET NOVERIFY
$ WRITE SYS$ERROR -
" * * PARAMETER LIST * *"
$ WRITE SYS$ERROR -
" P1 = 'P1' : SYSTEM NUMBER"
$ WRITE SYS$ERROR -
" P2 = 'P2' : DEFAULT DIRECTORY"
$ WRITE SYS$ERROR -
" P3 = 'P3' : LIST OF FILES TO BE ASSEMBLED"
$ WRITE SYS$ERROR -
" P4 = 'P4' : .LIS, .OBJ OPTIONS: [BOTH], [NONE], [LIS], OR [OBJ]"
$ WRITE SYS$ERROR -
" P5 = 'P5' : SPECIFIES PRINT QUEUE: [MX_NOTE], [NONE], [MX_TOME]"
$ WRITE SYS$ERROR -
" P6 = 'P6' : [NO], OR -DECxxT.CNF- TO TASK BUILD IF NO ERRORS"
$ WRITE SYS$ERROR -
" P7 = 'P7' : [NO], OR -DECxxM.CNF- TO RUN MERGE IF NO ERRORS"
$ WRITE SYS$ERROR -
" P8 = 'P8' : [NONE], OR FLOPPY LABEL FOR DUMP"
$ WRITE SYS$ERROR -
" * * END OF LIST * *"
$ SET DEFAULT 'P2'
$ SHOW DEFAULT
$ OCS:NEWLOGIN 'P2'
$START:

```

Many procedures are designed for batch use only. If this is true, exit with a message when used interactively.

If, however, a "batch" procedure can also be run interactively, then do not print parameters.

It is important to keep the .LOG files as short and concise as possible. SET NOVERIFY at the beginning, after the name.

Print out parameters in the .LOG file in an orderly list. This turns out to be good documentation for the procedure.

```

$! ASSMBLBA.COM R.H.WARNER 12 JUL 84
$!
$! ASSEMBLE LIST OF DEC SOURCE FILES FROM .TMP FILE
$!
$ SET NOVERIFY
 ** PARAMETER LIST **
P1 = 1428 : SYSTEM NUMBER
P2 = UDF1:[R1428] : DEFAULT DIRECTORY
P3 = 114ASM.TMP : LIST OF FILES TO BE ASSEMBLED
P4 = BOTH : .LIS, .OBJ OPTIONS: [BOTH], [LIS], [OBJ], OR [NONE]
P5 = SYS$PRINT : SPECIFIES PRINT QUEUE: [MX NOTE], [MX TOME], [NONE]
P6 = DEC4V4T.CNF : [NO], OR -DECxxT.CNF- TO TASK BUILD IF NO ERRORS
P7 = DEC4V4M.CNF : [NO], OR -DECxxM.CNF- TO RUN MERGE IF NO ERRORS
P8 = NONE : [NONE], OR FLOPPY LABEL FOR DUMP
 ** END OF LIST **
UDF1:[R1428]
MX$D = "UDD1:[MANT.CANDA.COM]" (process)
MX$$ = "MDSA:[MXCOATER.COATERB2]" (process)
MX$Z = "MDSA:[MXDECADE.B2DB]" (process)
24-JUL-1984 10:14:18
A = CONV4C.MAC
$ MCR MAC CONV4C,CONV4C.LIS/LI:TTM/-SP/CR=A
ERRORS DETECTED: 418
CONV4C,CONV4C.LIS/LI:TTM/-SP/CR=A
 COMPLETION STATUS WAS - ERROR -
$ MCR CRF CONV4C.LIS

SUBMITTED JOB TO MX QUICK BATCH QUEUE TO RUN THE ERROR
PROCEDURE. THE ERROR LIST WILL BE PRINTED ACCORDING TO P5.

Job 5951 entered on queue MXQ FAST
Job 5952 entered on queue SYS$PRINT

```

This a portion of a typical .LOG file. This one has been edited for brevity on the slide. Note the clear, concise information presented for the user.

The Parameter list is of great value in evaluating users' problems. It is a true history of what they submitted. The parameters are used to their fullest by allowing several options for each of them.

If any special logical assignments are used by the batch procedure, print them out for reference.

Since NOVERIFY is on, the procedure must print out any significant events. It is also important to SHOW TIME when starting a CPU intensive task, such as MCR TKB.

One of the best features of the procedures is the ability to "chain" batch jobs. The ASSEMBLE batch job can submit a TASKBUILD batch job, which can submit a MERGE batch job, which finally can submit a batch job to dump the floppies. Each time that a new batch job is submitted, the event is described in the .LOG file.

The batch procedures are in nice modules so that it is a simple task to "chain" them. The DECFINAL, for example, is mostly housekeeping, and run the already available procedures, plus a batched job to create a final microfiche tape.

We have computer operators staff to the Print, Tape, and Floppy requests for the users. The Tape and Floppy jobs each have their own Batch Queue. These queues run at a higher priority than batch to expedite them through operations.

It is very important to check for errors, and handle the error conditions carefully. Try to continue processing as much as possible to get the most information from the batch job.

In this particular ASSEMBLE batch job, there was an error found. When an error is found in this job, a batch job is submitted to help locate the errors.

If errors are found, always explain the conditions to the user and suggest possible solutions.

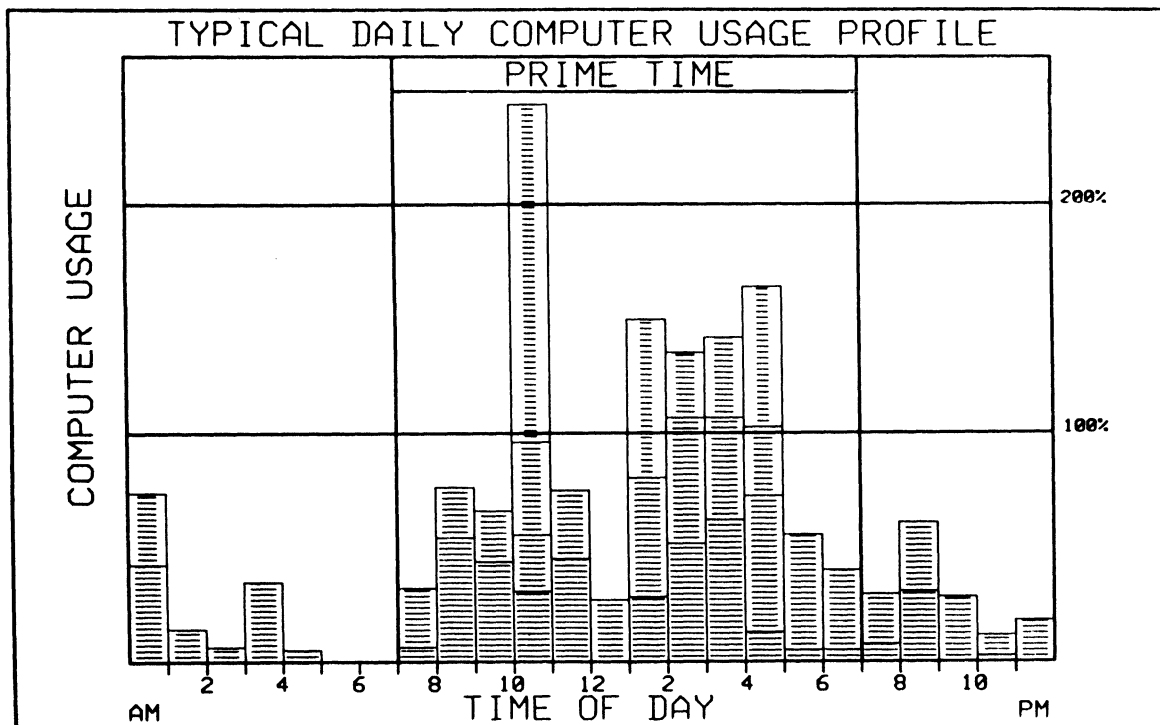
#### Why use Batch?

The integrators have two uses for the computer, EDIT files, and run Command procedures.

If all integrators did all of their work interactively at top priority, the computer usage would look like this typical day.

The VMS operating system handles an overload problem like this by effectively reducing the available CPU time proportional to its overload. The immediate result is poor response time for interactive tasks such as editing, searching, directory, etc.

Any one here have an overloaded VAX?? When you run EDT in key pad mode you get these lovely pauses ... maybe I didn't hit the back arrow .. then blast!! .. way past where you wanted to go...

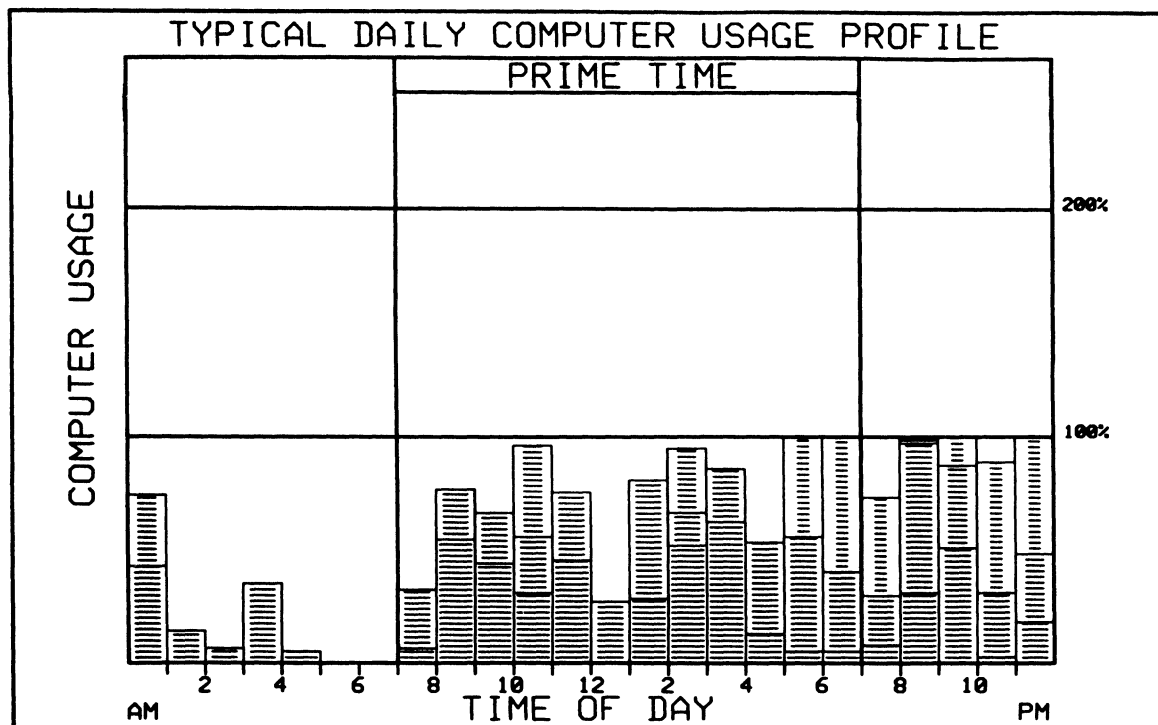


Our solution to the problem was to do all high CPU intensive tasks in Batch mode at lower priorities. Short jobs such as 1 to 3 assemblies run at a priority of 3. Longer jobs run at a priority of 2, and the longest jobs run at a priority of 1, or are forced to run after 6 PM.

When this was done, the computer usage had a nice pattern to it as shown here. Plenty of time for interactive jobs; short jobs come out of batch in about an hour or so, longer jobs require about 4.

Another feature is that the integrators have the option of delaying any batch job until after 8 PM. It is surprising how well the integrators have adapted to this feature. If it is late in the day, or they have a meeting to attend, they just elect to have their job run later at night.

The high usage of Batch processing is the reason for the extensive command procedures. In order to make it all work well, the procedures have considerable on-line checking, and the procedures then became very large.



\* ADVANTAGES TO USING DCL COMMAND PROCEDURES \*

- Allow efficient use of batch processing
- Can be used to create effective tools
- Use a very familiar and friendly language, DCL
- Can be used as a high-level language
- Can run other high-level language programs
- Do not require compile and link processes:
  - . easier to test and debug modules
  - . easier to modify and maintain

I have been criticized very severely at times because my procedures are too large and too inefficient.

It is true that my procedures are large. My ASSEMBLE.COM procedure and its associated sub-procedures contain about 1800 lines of code. It submits the ASSMBLBA.COM procedure which totals about 1200 lines.

It is also true that if the procedures were written in FORTRAN they would be more efficient. It is difficult to determine quantitatively how much would be gained by using programs.

We have had excellent results with our procedures. We made an excellent choice in using DCL as our programming language for building our systems.

These are some of the tradeoffs:

- . procedures and programs are equally easy/hard to write
- . procedures are easier to test and modify because they are interpretive and do not require a LINK process when making a minor change
- . procedures are easier to maintain for the same reasons
- . programs are best for CPU intensive applications
- . procedures may run FORTRAN modules which are CPU intensive
- . programs are best for clearly defined functions
- . procedures are best for constantly changing requirements

I clearly recommend DCL procedures over high-level programming. The advantages are clear.



Recovery of Lost Files from VAX/VMS Disk Structures.  
A Case Study.

Larry W. Ebinger \*  
Sandia National Laboratories  
Instrumentation Development Division I  
Albuquerque, New Mexico

ABSTRACT

Having a corrupted system disk can be a scary thing. This paper will describe the somewhat humorous events leading up to the destruction of the file structure on the only good copy of our VMS system disk. This disk unfortunately also contained all of the user files. The method used for file recovery will be detailed. Copies of the programs developed to recover the files will be available. Other recovery programs will also be noted.

HOW WE GOT STARTED

The whole thing started when our new VAX minicomputer arrived. Although we had spent over 10 years with numerous PDP-11s this was our first VAX. We told everyone we didn't know what we were doing because we were still inexperienced.

The master plan was to upgrade the minicomputer we call the Laboratory System from a PDP-11/50 to a VAX-11/751. The VAX-11/751 is an ugly rack mount version of the VAX-11/750. The current system had to remain operational because of several prior commitments. We devised a plan where one or both of the RPO6 disk drives and one or both RLO2 disk drives could be on either system. The tape drives could also be on either system. This could be accomplished by simply moving cables even when one or both systems were being used. Obviously we had a very flexible setup.

Things were going along just fine. The users were gaining experience and confidence. We could use both systems effectively and we were learning about the system management of our new VAX. We had done system updates to the operating system and had installed three layered products. Our users were basically happy with the system even though they were told we were not yet treating it as operational. They were told to be careful about which kind of files they kept on the system because the system disk was not being backed up at regular intervals.

HOW WE GOT INTO TROUBLE

Two of our users became concerned because they had some files on the system which they didn't want to lose. We reminded them that they were not to do that. We also decided it was well past time to do another backup of the system disk. One of the RPO6 disk

drives was then being used as the system disk and the RA81s were used as scratch drives. We also decided it was time to switch over to the RA81s and use one of them as the system disk. We were unsure of using the BACKUP utility on line to save the system disk because we always received these strange messages about files still open by another user; we knew we were the only user on the system at the time. We were concerned that maybe we were not making good copies of our disk. Anyway, we brought the system down and booted a copy of the stand-alone BACKUP from an RLO2. We then typed in a command to copy the data from the RPO6 disk pack into the RA81 disk drive. We checked our command and then write protected the RPO6 so that even if something went wrong we would not destroy our original system disk.

We were very unlucky that day. We were accustomed to the IAS/RSX Monitor Console Routine which generally requires the destination or new file to be specified first and the source or old file last. You guessed it, the VMS BACKUP utility wants them in the reverse order. Actually, BACKUP will prompt you for the destination and source. We tried to take advantage of the feature but the stand-alone version of BACKUP doesn't work that way. It requires the entire command to be on a single line.

So far we have had three errors. First is the inability of stand-alone BACKUP to be user friendly and the second is a syntax error in typing the command and the third is there were to be no critical files on the system. The fourth error came when we write protected the RPO6 drive 0. Yes, we did indeed write protect an RPO6 drive 0 and yes, our system disk was in the RPO6 drive 0. Unfortunately we had one RPO6 disk drive on each of the systems and they were both drive 0. We had write protected the wrong drive. Our BACKUP was making a copy of the

scratch disk on top of our only good copy of the system disk. When we realized what was happening we halted the CPU but it was too late. To prevent making things worse than they already were we write protected the correct, but now corrupt system disk because we didn't want to make things worse than they were already.

We have found that if you are careful it takes at least two errors to really foul up. In this case we had at least four.

Our next step was to find out what was wrong with the disk. We could mount it but it now had the volume ID of the scratch disk. The directories and file names within the directories were also the same as those of the scratch disk. Most of the files however, were not actually there because we did get the operation stopped before the majority of the transfers were completed.

#### HOW BAD WAS IT?

Here we are. We have two users glaring at us and yelling "What have you done to us?!" and we are responding "What did you do to yourself?". After things calmed down a bit we found out the first user had been programming in FORTRAN and had listings of all he had done. He estimated it would take two to three days to get back to where he was before the "backup". We told him to hold off a while as all was not lost and there was still a possibility of recovering his files. He didn't believe us and elected to reenter his programs back into the VAX.

The second user was a different story. He had been working on a large document which was the conglomeration of status from about ten different people working on the same project. When the information was entered into the VAX the original status sheets were discarded. This represents a sixth error as he had lost the ability to easily reconstruct the data. He was willing to wait to see if we could recover his file for him. All of the dozen or so other users said any files they had on the system were either backed up elsewhere or they were just "learning" files and were unimportant.

When you have a disk disaster there are usually only a very few files which need to be recovered. Most of the others are on old save sets or are not very important at all.

Next I started calling people for help. I talked to some of the expert friends of mine but they were of little help. I called the VAX Local User's Group's president and librarian to see if there was anything they knew about from DECUS which might help. Neither of them knew of anything relevant to our particular situation. I then called a couple of experts from DIGITAL. The only help I got from them is they told me where to find the definition of the VAX/VMS disk structure. I called some other users in

Albuquerque with no further luck.

#### WHAT TO DO NOW

By now a few days had passed and we hadn't recovered a thing. I figured if it was going to get done I had better get busy. I looked through the file `SYS$$SYSROOT:[SYSLIB]STARLET.REQ` (see Figure 1) which is 986 blocks long. There I found the file header definitions for the Files-11 Structure Level 2. What I thought I needed was within about five pages. The only problem was that it is written in BLISS and I don't know how to read BLISS code. After some study of this section I thought I understood what it was saying. It contains the offset, size and bit fields within the file header.

Using DUMP I printed out the header of several files from a good VMS disk (see Figure 2 and 3). I was able to confirm what the BLISS file was trying to tell me about the locations of the items within the file headers. What I found was that in the first byte of the file header was the word offset to the "ident" area and it looked like it was always 38. The second byte was the map area offset in words and its value was always 65. The file name was always in ASCII and started at byte 76 and was about 20 characters long.

Now I was ready to start looking for the badly needed file. I coded a program called READHEAD to read every block on the now defunct system disk (see Figure 4). I checked to see if those two bytes were correct and if they were I printed the physical block number and the name of the possible file. This gave me a list of over 2000 file names. We found the correct file name among the others (see Figure 5).

Next I used DUMP with the `/FILE/BLOCKS=(START:100572,COUNT:1)` switches and indeed we had found the correct file header. It said the file was 104 blocks long (see Figure 6). Things were starting to look up.

Now a routine was written called FORMHEAD (see Figure 7) to read a header block from the clobbered disk and output the block number, the file name, the starting block number of the actual data and the length of the data in blocks. It then continued to list out additional data blocks if the file happened to not be contiguous. This program turned out to be unnecessary as the DUMP utility will supply that information. (See the bottom of Figure 6.)

We now have all we need to go get the lost file. READFILE was the next program written to read a block of data from the clobbered disk and write it out to a good disk (see Figure 8). It requires the clobbered disk name, the new file name, the starting block number and the number of blocks. If the file was not contiguous this program needed to be run once for each of the data blocks. The individual files could later be appended

with the VMS COPY or APPEND command. At this point we had the file on a good disk but it was still an unformatted file. Next I wrote a program called REFORM (see Figure 9) to convert the unformatted fixed length file into a variable sequential file with implied carriage control. It turned out this program was also unnecessary. All that was needed was to edit the file with the EDIT editor and it would automatically convert the file on exit.

Now we had the file recovered and of course we were heroes. Others now decided some of their files were actually worth recovering.

#### OTHER RECOVERY PROGRAMS

There is a nice package on the 1984 Spring DECUS US VAX SIG Tape. Fortunately, I have yet to need to use it so I really don't know what it does or how well it works.

In the VMS package there is a program called ANALYZE/DISK\_STRUCTURE which will find files which are in the master index file but not in any directory. This is good for if some reason the directory file goes bad you may recover your files.

In the VMS package is a routine which corrects disks which were unloaded before the file system was finished with them. This could happen because of a power failure

or some other error. When you remount the disk, the system automatically rebuilds it for you.

#### SUMMARY

This paper was not to show my expertise in solving lost file problems but to show how simple something can really be if it is broken down into manageable pieces. The coding was left as it was when I completed the recovery of this one file. All the programming was done in FORTRAN and I used no tricks. No one would ever present this code to impress someone. Each program is so simple that any novice programmer should be able to understand them. They could easily be combined into one or two programs and be made to run automatically or semi-automatically. Some of the problems with the current routines is that they rely on finding the file header but the data may be there even when the header is not. They are inconvenient to run and the offsets are fixed so a new release of the file system could break the code. But for now I'm satisfied. The code recovered our files and all is well.

If you have a disaster you might try this method of recovery. You may be able to recover from someone deleting a file they didn't want to delete or even from a head crash.

\* This work performed at Sandia National Laboratories supported by the U.S. Department of Energy under contract number DE-AC-76DPO0789.



Figure 1

Part of SYS\$SYSROOT:[SYSLIB]LIB.REQ

```

!+
! File header definitions for Files-11 Structure Level 2
!-
!...$FH2DEF ! Header area

MACRO FH2$B_IDOFFSET = 0,0,8,0%; ! ident area offset in words
MACRO FH2$B_MPOFFSET = 1,0,8,0%; ! map area offset in words
MACRO FH2$B_ACOFFSET = 2,0,8,0%; ! access control list offset in words
MACRO FH2$B_RSOFFSET = 3,0,8,0%; ! reserved area offset in words
MACRO FH2$W_SEG_NUM = 4,0,16,0%; ! file segment number
MACRO FH2$W_STRUCLEV = 6,0,16,0%; ! file structure level
MACRO FH2$B_STRUCVER = 6,0,8,0%; ! file structure version
MACRO FH2$B_STRUCLEV = 7,0,8,0%; ! principal file structure level
LITERAL FH2$C_LEVEL1 = 257; ! 401 octal = structure level 1
LITERAL FH2$C_LEVEL2 = 512; ! 1000 octal = structure level 2
MACRO FH2$W_FID = 8,0,0,0%; ! file ID
LITERAL FH2$S_FID = 6;
MACRO FH2$W_FID_NUM = 8,0,16,0%; ! file number
MACRO FH2$W_FID_SEQ = 10,0,16,0%; ! file sequence number
MACRO FH2$W_FID_RVN = 12,0,16,0%; ! relative volume number
MACRO FH2$B_FID_RVN = 12,0,8,0%; ! alternate format RVN
MACRO FH2$B_FID_NMX = 13,0,8,0%; ! alternate format file number extension
MACRO FH2$W_EXT_FID = 14,0,0,0%; ! extension file ID
LITERAL FH2$S_EXT_FID = 6;
MACRO FH2$W_EX_FIDNUM = 14,0,16,0%; ! extension file number
MACRO FH2$W_EX_FIDSEQ = 16,0,16,0%; ! extension file sequence number
MACRO FH2$W_EX_FIDRVN = 18,0,16,0%; ! extension relative volume number
MACRO FH2$B_EX_FIDRVN = 18,0,8,0%; ! alternate format extension RVN
MACRO FH2$B_EX_FIDNMX = 19,0,8,0%; ! alternate format extension file number
MACRO FH2$W_RECATTR = 20,0,0,0%; ! file record attributes
LITERAL FH2$S_RECATTR = 32;
MACRO FH2$L_FILECHAR = 52,0,32,0%; ! file characteristics

 ! reserved
MACRO FH2$V_NOBACKUP = 52,1,1,0%; ! file is not to be backed up
LITERAL FH2$M_NOBACKUP = 1^2 - 1^1;
MACRO FH2$V_WRITEBACK = 52,2,1,0%; ! file may be write-back cached
LITERAL FH2$M_WRITEBACK = 1^3 - 1^2;
MACRO FH2$V_READCHECK = 52,3,1,0%; ! verify all read operations

!...$FI2DEF ! Ident area

MACRO FI2$T_FILENAME = 0,0,0,0%; ! file name, type, and version (ASCII)
LITERAL FI2$S_FILENAME = 20;
MACRO FI2$W_REVISION = 20,0,16,0%; ! revision number (binary)
MACRO FI2$Q_CREDATE = 22,0,0,0%; ! creation date and time

```

Figure 2

Dump of a good file header

Dump of file SYS\$SYSDEVICE:[EBINGER.MEMO]CPUMAIN.T.RNO;5 on 13-FEB-1984 10:32:25  
File ID (1647,10,0) End of file block 9 / Allocated 9

File Header

|        |        |        |        |        |        |        |        |               |        |
|--------|--------|--------|--------|--------|--------|--------|--------|---------------|--------|
| 000000 | 000000 | 000012 | 003157 | 001001 | 000000 | 177777 | 040446 | &A.....o..... | 000000 |
| 000011 | 000000 | 000011 | 000000 | 000113 | 001002 | 000000 | 000000 | .....K.....   | 000020 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000134 | \.....        | 000040 |
| 000002 | 000002 | 000002 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000060 |
| 046525 | 050103 | 000000 | 000000 | 000000 | 000001 | 001632 | 175000 | .....CPUM     | 000100 |
| 020040 | 020040 | 020040 | 032473 | 047516 | 051056 | 052116 | 044501 | AIN.T.RNO;5   | 000120 |
| 144342 | 104637 | 130300 | 000214 | 144342 | 104560 | 061540 | 000001 | ..`cp.....    | 000140 |
| 042542 | 034417 | 003100 | 000000 | 000000 | 000000 | 000000 | 000214 | .....@..9bE   | 000160 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 004347 | 043410 | 000215 | ...G.....     | 000200 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000220 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000240 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000260 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000300 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000320 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000340 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000360 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000400 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000420 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000440 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000460 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000500 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000520 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000540 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000560 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000600 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000620 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000640 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000660 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000700 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000720 |
| 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....         | 000740 |
| 045562 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | .....rK       | 000760 |

Figure 3

Dump of a good file header (formatted)

Dump of file SYS\$SYSDEVICE:[EBINGER.MEMO]CPUMAIN.T.RNO;5 on 13-FEB-1984  
File ID (1647,10,0) End of file block 9 / Allocated 9

File Header

Header area

|                                |                          |
|--------------------------------|--------------------------|
| Identification area offset:    | 38                       |
| Map area offset:               | 65                       |
| Access control area offset:    | 255                      |
| Reserved area offset:          | 255                      |
| Extension segment number:      | 0                        |
| Structure level and version:   | 2, 1                     |
| File identification:           | (1647,10,0)              |
| Extension file identification: | (0,0,0)                  |
| VAX-11 RMS attributes          |                          |
| Record type:                   | Variable                 |
| File organization:             | Sequential               |
| Record attributes:             | Implied carriage control |
| Record size:                   | 75                       |
| Highest block:                 | 9                        |
| End of file block:             | 9                        |
| End of file byte:              | 92                       |
| Bucket size:                   | 0                        |
| Fixed control area size:       | 0                        |
| Maximum record size:           | 0                        |
| Default extension size:        | 0                        |
| Global buffer count:           | 0                        |
| Directory version limit:       | 0                        |
| File characteristics:          | <none specified>         |
| Map area words in use:         | 2                        |
| Access mode:                   | 0                        |
| File owner UIC:                | [000002,000002]          |
| File protection:               | S:RWED, O:RWED, G:RE, W: |
| Back link file identification: | (922,1,0)                |

Identification area

|                  |                         |
|------------------|-------------------------|
| File name:       | CPUMAIN.T.RNO;5         |
| Revision number: | 1                       |
| Creation date:   | 14-JUN-1984 00:20:03.35 |
| Revision date:   | 14-JUN-1984 00:20:03.66 |
| Expiration date: | <none specified>        |
| Backup date:     | 19-NOV-1984 10:46:28.26 |

Map area

|                    |   |             |
|--------------------|---|-------------|
| Retrieval pointers |   |             |
| Count:             | 9 | LBN: 461031 |

|           |       |
|-----------|-------|
| Checksum: | 19314 |
|-----------|-------|

Figure 4

READHEAD

```

C READHEAD.FOR
C
C THIS PROGRAM READS THE ASKED FOR DISK AND PRINTS OUT THE BLOCK
C NUMBER AND FILENAME OF THE BLOCK.
C
C INTEGER*4 J
C BYTE B(512), FILEN(24), DISK(24)
C EQUIVALENCE (B(76), FILEN(1))
C
C WRITE (5,1002)
C READ (5,1003, END=10) (DISK(I), I=1,24)
100 OPEN (UNIT=1, FILE=DISK, STATUS='OLD', FORM='UNFORMATTED')
C OPEN (UNIT=6, FILE='SYS$PRINT', STATUS='NEW')
C
C DO 100 J=1,1000000 !LARGE NUMBER
C READ (1, IOSTAT=IOST) B !READ A BLOCK
C WRITE (5,1000) ((I-1, B(I), B(I+1), B(I+2), B(I+3),
C 1 B(I+4), B(I+5), B(I+6), B(I+7),
C 2 B(I+8), B(I+9), B(I+10), B(I+11),
C 3 B(I+12), B(I+13), B(I+14), B(I+15)), I=1,596,16)
C IF (IOST .NE. 0) THEN !CHECK FOR ERROR
C WRITE (5,1004) J-1, J-1 !PRINT OUT..
C WRITE (6,1004) J-1, J-1 ! NUMBER READ
C STOP !AND EXIT
C END IF
C
C IF (B(1) .EQ. 38 .AND. B(2) .EQ. 65) THEN !CHECK FOR HEADER
C WRITE (6,1001) J-1, J-1, (FILEN(I), I=1,20) !PRINT OUT HEADER
C WRITE (5,1001) J-1 !TYPE " "
C END IF
100 CONTINUE !END OF LOOP
C
1000 FORMAT (1H , 1704)
1001 FORMAT (1H , 010, I10, 5X, 20A1)
1002 FORMAT (30H$ ENTER THE DISK DEVICE NAME)
1003 FORMAT (45A1)
1004 FORMAT (1H0, 010, I10, 5X, 11HBLOCKS READ)
END

```

Figure 5

Small portion of the File Name List

```
o
o
o
o
304330 100568 PCOFD.FOR;1
304331 100569 OPERATOR.LOG;88
304332 100570 STATUS.MEM;2
304333 100571 PVALD.OBJ;1
304334 100572 STATUS.RNO;10
304335 100573 CNVRT.OBJ;1
304336 100574 CONFIG.OBJ;2
304337 100575 CONFIG.EXE;2
304340 100576 BUFFER.CMN;1
304341 100577 CONFIG.FOR;2
304342 100578 OPERATOR.LOG;70
304343 100579 CONFIG.FOR;2
304344 100580 OPERATOR.LOG;90
304345 100581 CNVRT.FOR;1
o
o
o
o
```

Figure 6

Dump of Lost File Header

Dump of device DBAO: on 13-FEB-1984 12:47:50.46

Logical block number 100572 (000188DC), 512 (0200) bytes

Header area

|                                |                          |
|--------------------------------|--------------------------|
| Identification area offset:    | 38                       |
| Map area offset:               | 65                       |
| Access control area offset:    | 255                      |
| Reserved area offset:          | 255                      |
| Extension segment number:      | 0                        |
| Structure level and version:   | 2, 1                     |
| File identification:           | (1434,9,0)               |
| Extension file identification: | (0,0,0)                  |
| VAX-11 RMS attributes          |                          |
| Record type:                   | Variable                 |
| File organization:             | Sequential               |
| Record attributes:             | Implied carriage control |
| Record size:                   | 79                       |
| Highest block:                 | 104                      |
| End of file block:             | 104                      |
| End of file byte:              | 184                      |
| Bucket size:                   | 0                        |
| Fixed control area size:       | 0                        |
| Maximum record size:           | 0                        |
| Default extension size:        | 0                        |
| Global buffer count:           | 0                        |
| Directory version limit:       | 0                        |
| File characteristics:          | <none specified>         |
| Map area words in use:         | 4                        |
| Access mode:                   | 0                        |
| File owner UIC:                | [000030,000011]          |
| File protection:               | S:RWED, O:RWED, G:RE, W: |
| Back link file identification: | (1221,1,0)               |

Identification area

|                  |                         |
|------------------|-------------------------|
| File name:       | STATUS.RNO;10           |
| Revision number: | 1                       |
| Creation date:   | 15-FEB-1984 16:02:18.96 |
| Revision date:   | 15-FEB-1984 16:02:23.51 |
| Expiration date: | <none specified>        |
| Backup date:     | <none specified>        |

Map area

|                    |    |      |        |
|--------------------|----|------|--------|
| Retrieval pointers |    |      |        |
| Count:             | 77 | LBN: | 100119 |
| Count:             | 27 | LBN: | 100198 |

Checksum: 39282

Figure 7

```

FORMHEAD
FORMHEAD.FOR
THIS PROGRAM READS A DISK BLOCK TRIES TO FIGURE OUT IF IT IS A
HEADER BLOCK. IT THEN OUTPUTS A LISTING OF WHAT IT THINKS THE
FILE IS AND WHERE ITS BLOCKS ARE LOCATED.

INTEGER*2 TEMP, ISIZEN
INTEGER*4 J, IBLOCK, LBNN
BYTE B(512), FILEN(24), DISK(24), FILE(24), LBN(4), MAP(4)
BYTE ISIZE(4), TEMP1(2)
EQUIVALENCE (B(77), FILEN(1))
EQUIVALENCE (LBNN, LBN(1))
EQUIVALENCE (ISIZEN, ISIZE(1))

C
C GET AND OPEN THE DISK IN QUESTION
WRITE (5,1000)
READ (5,1001, END=10) (DISK(I), I=1,24)
10 OPEN (UNIT=1, FILE=DISK, STATUS='OLD', FORM='UNFORMATTED',
1 ACCESS='DIRECT')

C
C FIND OUT WHAT BLOCK NUMBER THEY WANT AND GO GET IT
WRITE (5,1002)
READ (5,1003) IBLOCK
READ (1, REC=IBLOCK) B !READ A BLOCK

C
C DECODE AND LIST HEADER BLOCK
WRITE (5,1008) !OUTPUT HEADER MESSAGE
WRITE (5,1004) IBLOCK, IBLOCK !OUTPUT BLOCK NUMBER
WRITE (5,1005) (FILEN(I), I=1,20) !FILENAME
INDEX=130
20 LBN(1)=B(INDEX+3) !GET THE PHYSICAL
LBN(2)=B(INDEX+4) ! BLOCK NUMBER OF THE DATA
LBN(3)=B(INDEX+2) !GET HIGH ORDER NUMBER
IF (LBN(3) .GE. 64) LBN(3)=LBN(3)-64 !CLEAR OFF JUNK
LBN(4)=0 !FINISH UP LONGWORD
WRITE (5,1006) LBNN, LBNN !OUTPUT START OF DATA

C
ISIZE(1)=B(INDEX+1) !GET THE NUMBER OF BLOCKS
ISIZE(2)=0 ! CAN'T BE THAT BIG
ISIZEN=ISIZEN+1 !ADD IN THE EXTRA BLOCK
WRITE (5,1007) ISIZEN, ISIZEN !OUTPUT NUMBER OF DATA BLOCKS
INDEX=INDEX+4 !MOVE POINTER TO NEXT MAP
IF (B(INDEX+4) .NE. 0) GOTO 20 !SEE IF THERE'S ANOTHER

1000 FORMAT (30H$ ENTER DISK NAME)
1001 FORMAT (24A1)
1002 FORMAT (30H$ ENTER BLOCK NUMBER IN OCTAL)
1003 FORMAT (010)
1004 FORMAT (30HOLIST OF BLOCK , I10, 010)
1005 FORMAT (15HOFILNAME = , 20A1)
1006 FORMAT (30HOSTART OF DATA BLOCK , I10, 010)
1007 FORMAT (30H LENGTH OF DATA , I10, 010)
1008 FORMAT (1H0,29X, 20H DECIMAL OCTAL)
END

```

Figure 8

READFILE

```

C READFILE.FOR
C
C THIS PROGRAM READS THE DISK AND PUTS THOSE BLOCKS IN A FIXED
C LENGTH FILE FOR MORE RECOVERY LATER
C
 INTEGER*4 J, IBLOCK, ISIZE
 BYTE B(512), FILEN(24), DISK(24), FILE(24)
 EQUIVALENCE (B(76), FILEN(1))
C
 WRITE (5,1104)
 READ (5,1105, END=10) (DISK(I), I=1,24)
10 OPEN (UNIT=1, FILE=DISK, STATUS='OLD', FORM='UNFORMATTED',
 1 ACCESS='DIRECT')
C
 WRITE (5,1106)
 READ (5,1105, END=20) (FILE(I), I=1,24)
20 OPEN (UNIT=2, STATUS='NEW', FILE=FILE,
 1 FORM='UNFORMATTED', ACCESS='DIRECT',
 2 RECORDTYPE='FIXED', RECORDSIZE=128)
C
 WRITE (5,1100)
 READ (5,1101) IBLOCK
 WRITE (5,1102)
 READ (5,1101) ISIZE
 DO 100 J=1,ISIZE
 READ (1, REC=IBLOCK, ERR=90) B !READ A BLOCK
 WRITE (2, REC=J) B !WRITE A BLOCK
90 IBLOCK=IBLOCK+1
100 CONTINUE
 CLOSE (UNIT=2, DISPOSE='KEEP')
1000 FORMAT (1H , I4, 5X, 16A1)
1001 FORMAT (1H , 010, 5X, 20A1)
1100 FORMAT (30H$ ENTER STARTING BLOCK NUMBER)
1101 FORMAT (I20)
1102 FORMAT (30H$ ENTER SIZE OF FILE)
1103 FORMAT (1H , 2I20)
1104 FORMAT (30H$ ENTER CLOBBERED DISK NAME)
1105 FORMAT (24A1)
1106 FORMAT (30H$ ENTER NEW FILE NAME)
 END

```



Figure 9

REFORM

```

C REFORM.FOR
C THIS PROGRAM TAKES THE FIXED LENGTH BLOCK MODE FILE AND
C MAKES IT INTO THE VARIABLE LENGTH FILE
C
 INTEGER*4 J, IBLOCK, ISIZE
 BYTE B(1024), FILEI(24), FILEO(24), STRING(133)
 BYTE B1(512), B2(512)
 EQUIVALENCE (B(1), B1(1))
 EQUIVALENCE (B(513), B2(1))
C
C GET AND OPEN INPUT FILE
 WRITE (5,1100)
 READ (5,1101, END=10) (FILEI(I), I=1,24)
10 OPEN (UNIT=1, FILE=FILEI, STATUS='OLD', FORM='UNFORMATTED',
 1 ACCESS='DIRECT')
C
C GET AND OPEN OUTPUT FILE
 WRITE (5,1102)
 READ (5,1101, END=20) (FILEO(I), I=1,24)
20 OPEN (UNIT=2, FILE=FILEO, STATUS='NEW',
 1 FORM='FORMATTED', ACCESS='SEQUENTIAL',
 2 RECORDTYPE='VARIABLE',
 3 CARRIAGECONTROL='LIST')
C
 ISTART=3 !RECORD START
 ILOC=1 !RECORD SIZE LOCATION
 IBLOCK=1 !START WITH THE FIRST BLOCK
30 READ (1, REC=IBLOCK, ERR=100) B1 !READ A BLOCK TO B1
 READ (1, REC=IBLOCK+1, ERR=35) B2 !READ NEXT BLOCK
35 IBLOCK=IBLOCK+1 !POINT TO NEXT BLOCK
40 ISIZE=B(IBLOCK) !GET RECORD SIZE IN BYTES
 WRITE (2,1002) (B(I), I=ISTART,ISTART+ISIZE-1) !OUTPUT RECORD
C WRITE (5,1003) (B(I), I=ISTART,ISTART+ISIZE-1) !OUTPUT RECORD
 ILOC=ILOC+ISIZE+2 !POINT TO NEXT BYTE COUNT
 I=(ILOC/2)*2 !SEE IF WE ARE EVEN
 IF (I .EQ. ILOC) ILOC=ILOC+1 ! AND CORRECT IF SO
 ISTART=ILOC+2 !ADJUST RECORD START POSITION
 IF (ILOC .LT. 512) GOTO 40 !GO WRITE NEXT RECORD
 ILOC=ILOC-512 !BACK POINTER TO FIRST BLOCK
 ISTART=ILOC+2 !ADJUST RECORD START POSITION
 GOTO 30 !GO READ NEXT DATA BLOCK
100 CLOSE (1) !CLOSE INPUT FILE
 CLOSE (2, DISPOSE='KEEP') !CLOSE OUTPUT FILE
1000 FORMAT (133A1)
1001 FORMAT (1H , 133A1)
1002 FORMAT (133A1)
1003 FORMAT (1H , 133A1)
1100 FORMAT (30H$ ENTER FILENAME TO OPEN)
1101 FORMAT (24A1)
1102 FORMAT (30H$ ENTER NEW FILE)
9000 END

```

**TUNING RMS FILES  
A CASE STUDY ON VMS INDEXED FILES**

John W. Beyer  
Database Development Manager  
MarketVision Corporation  
40 Rector Street  
New York, New York 10006

**ABSTRACT**

RMS (Record Management Services) is a comprehensive file system that provides MACRO and higher level language access to sequential, indexed and relative record file organizations. An RMS file can be created with an OPEN statement in a language like FORTRAN; however, the resultant file will usually be far from optimal, especially in the case of indexed files. For example, indexed files will occupy a single area. The index records will be dispersed among the data records, resulting in unnecessary disk head movements when the index is traversed. Critical parameters, such as bucket size, will default to less than optimal values.

DEC provides a utility, the FDL (File Definition Language) Editor, which assists the user in making judicious selections for the values of these parameters. An interactive session with this editor results in the creation of an FDL file, which is used with the CREATE utility to create an optimized file structure. For indexed files, the first thing that FDL does is to separate index and data areas. This, by itself, is a significant improvement over FORTRAN. Further, you are prompted to select values for the critical parameters. Judicious selections can easily result in gains of up to 90% or more in I/O efficiency.

MarketVision's application is I/O intensive. After the first blind stab at tuning an indexed file with FDL, it was amazing to see 50% less I/O to the file. Since that time, considerable research and experimentation have resulted in a 90% reduction in I/O and an increase in user capacity by a factor of 6.

**SCOPE**

The purpose of this paper is twofold: 1) to describe the critical factors which must be considered when tuning files with FDL; and 2) to present and analyze the results of an extensive series of experiments in which files were tuned using a large variety of file parameters and the subsequent performance. The discussion encompasses the following:

1. VAX-11 RMS under VMS version 3.7
2. ISAM files
3. Pseudorandom access
4. Single key files
5. High level language access

Of the three file organizations supported, the most interesting, from a tuning standpoint, is the indexed sequential file. ISAM was originally an IBM acronym for their Indexed Sequential Access Method, but the term is now commonly used to refer to any ISAM-type file (in much the same manner as Coke refers to any cola-type soft drink). Sequential files are the simplest and don't require extensive tuning. They are also the most efficient, if your access requirements are strictly sequential. Relative files provide direct access by record number, but no means for key-sequenced access. ISAM files are the most common, since they provide for both random and sequential access.

Access methods figure prominently in a number of tuning decisions. ISAM files can

be accessed sequentially, in key sequence, or randomly by key. A third method is known as pseudorandom access, that is, sequential access from some starting point for some number of records or until a control break occurs. The results described below apply to this access method, but the analysis applies to random and sequential access as well.

All performance-critical files in the application were single key files. Performance declines drastically with each additional key, since multiple indices must be updated and pointers must be followed from the data level of the alternate index to the data level of the primary index during retrieval. Nevertheless, the analysis can be extended to tune these files as well, with less dramatic results.

The language used in the application was FORTRAN, but the same results can be expected using any high level language.

This paper is aimed at experienced users familiar with RMS and the FDL Editor. The scope is too detailed to provide a

tutorial overview. Additional information can be found in Volume 7 of VAX/VMS documentation (VAX/VMS Version 3.0), although the information in these manuals is somewhat incomplete.

## ANALYSIS OF FILE TUNING PARAMETERS

### Prologue Versions

Each RMS file contains a prologue, which contains information such as file attributes, key descriptors, etc. There are three versions of the prologue: Prologue 1, Prologue 2 and Prologue 3. Since there are no user-discernable differences between Prologues 1 and 2 (a file whose keys are all STRINGS will be constructed as a Prologue 1 file), the choice of Prologues is narrowed down to two: Prologue 1/2 or Prologue 3. For this discussion, Prologue 2 will refer to either Prologue 1 or Prologue 2. The default under VMS Version 3 is the most current version, Prologue 3. This version differs significantly from the previous two. The major differences are summarized below.

#### Prologue 2

1. no compression
2. reorganization requires a full CONVERT (rewrite) of the file
3. variety of key data types supported
4. alternate keys supported

#### Prologue 3

key compression, index and data record compression  
empty buckets can be RECLAIMed without rewriting the file  
keys must be of STRING data type  
primary key only

The decision as to which Prologue to use is based on an analysis of each of these features with respect to the application.

### Compression

The value of compression to the application must first be assessed. Compression can result in considerable space savings (in MarketVision's application, up to 50%). If disk space is the critical resource, compression may well be worth the cost. In our application, disk space is not critical, and, in most cases, compression does not improve performance. Compression can also produce a "flatter" index (fewer levels, with more information at each level) for a given bucket size, since more records will fit in each bucket. Furthermore, areas can be compressed selectively.

The savings realized through compression are largely data dependent. In

some cases, it will be profitable to compress the index records and not the data records, or vice-versa, depending on the pattern of repeating characters. Certain key distributions lend themselves well to key compression while others do not. The only way to determine this is to experiment. Turn the compression switches on through the FDL Editor, CONVERT the file, and use the ANALYZE/RMS utility to see the percent of compression realized. If the file is compressed by less than 10%, the savings realized will probably not outweigh the cost.

In order to compress the file several extra bytes are added to each record. If there are a small number of repeating characters, the cost of the extra bytes may be significant (and may even result in a negative value for compression). The most significant cost is the CPU time necessary to reconstruct the data on retrieval. The results of the experiments show that CPU time increases significantly when compression is turned on. This can be a

high price to pay in a CPU intensive environment.

If it can be determined that compression offers a significant advantage, then Prologue 3 must be used. If compression offers no advantage, then either Prologue 3 or Prologue 2 can be used without compression. However, a case can be made in favor of Prologue 2. The ANALYZE/RMS/INTERACTIVE utility is a useful means of examining the structure of a file. The index structure can be traversed interactively by using a few simple commands. At each level, the index records can be displayed, as well as the value of the bucket pointer to the next level. The DUMP command can be used to display the actual contents of the block in which the record is contained. With Prologue 2, the bucket pointer associated with each record can be examined when the block is DUMPed. With Prologue 3, the keys are at the beginning of the bucket, and the associated bucket pointers are at the end, making it more difficult to examine the bucket pointers. Additionally, this situation is not likely to improve performance. In fact, there is additional CPU overhead at run time with Prologue 3, even with compression off. Referring to figure 1, it can be seen that, with all other factors being equal, the CPU time for Prologue 3 files is about 5-10% greater than for Prologue 2 files. In other words, if compression is not used, it is more advantageous to use Prologue 2 than Prologue 3.

To use the compression feature, the procedure is to use the FDL Editor iteratively. On the first pass, using the DESIGN script, the user is prompted to enter the percentage of key compression and index and data compression (the percentage of compression is the reduction in file size due to compression divided by the size of the uncompressed file). But these values are not known at this point! The purpose of the first pass is to turn on the compression switches, so that the file can be CONVERTed and these values obtained using the ANALYZE/RMS utility for subsequent input to the FDL Editor using the OPTIMIZE script. The surface plot obtained from the first pass is meaningless, since the number of index levels depends on the amount of compression (Version 4 simply asks whether or not you want compression).

There is nothing magic about the OPTIMIZE script. It simply eliminates a few prompts by automatically taking the values obtained from the .FDL file created by ANALYZE/RMS. The same information can be entered manually using the DESIGN script after obtaining the .FDL file.

The compression switches are turned off by entering a value of zero and are turned on by entering a non-zero value. Incidentally, RMS automatically turns off the data compression switch if the non-key part of the record is below a certain minimum size. For example, data records can not be compressed if they are 15 bytes long and the keys are 10 bytes.

#### CONVERT vs. CONVERT/RECLAIM

It is characteristic of ISAM files that are subject to frequent additions/deletions that they need to be periodically reorganized. This is because adding a record to a bucket that is full causes a bucket split to occur (half of the bucket is moved to a new bucket, leaving behind a pointer to the new bucket). Bucket splits increase I/O overhead by causing additional disk head movement. Additionally, the process of splitting a bucket involves CPU overhead and a considerable amount of bucket and record locking. When a deletion occurs, the space occupied by the deleted record is not made available for reuse. RMS provides the CONVERT utility, which rewrites the file in an optimal manner, creating a new index structure and reclaiming all deleted space. The problem is that CONVERT is time consuming, especially for large files. The fact that space must be allocated for a new version of the file further complicates matters if space is at a premium.

For Prologue 3 files, RMS provides the /RECLAIM qualifier for indexed files. CONVERT/RECLAIM makes available for reuse those buckets that have been completely emptied by record deletions. The reclaiming is done in place, so no additional space is necessary. If there are severe space and time constraints, this is a useful feature, since the file size is kept to manageable levels. However, this results in negligible performance improvement, since bucket splits are not cleaned up. This is substantiated by some performance measurements on a file (more about the nature of the tests later) with the following results:

|                 | ELAPSED TIME<br>IMPROVEMENT | CPU TIME<br>IMPROVEMENT | I/O<br>IMPROVEMENT |
|-----------------|-----------------------------|-------------------------|--------------------|
| CONVERT/RECLAIM | 2.1%                        | 1.8%                    | 3.0%               |
| CONVERT         | 38.0%                       | 4.2%                    | 13.5%              |

If performance is a critical issue, then a full CONVERT should be performed as often as possible.

Another recourse available to improve the performance of frequently updated files is to adjust the fill factor. This topic will be covered later.

Since Prologue 3 does not support alternate keys or keys that are not STRINGS, files that have these properties must use Prologue 2.

#### Bucket Size

The bucket is the basic unit of I/O in RMS ISAM files. The bucket size is specified in terms of blocks (512 bytes), ranging from a minimum of 1 to a maximum of 32 (in Version 4, the maximum has been raised to 63). It is the bucket size, more than anything else, which determines the shape and size of the index. A small bucket size will result in an index with many levels, while a large bucket size will result in a flatter index. Since each level will result in an additional disk access for each I/O, it is desirable to make the index as flat as practical. The flattest index will result when the bucket size is 32 blocks. However, there are tradeoffs which will make such a large bucket size impractical. Since the intent is to minimize the amount of time spent doing I/O, it is worthwhile to examine the components of disk access.

#### Components of I/O Access

The time required to access a record on disk is composed of the following:

$$\text{I/O TIME} = \text{SEEK} + \text{LATENCY} + \text{DATA TRANSFER} + \text{BUCKET SEARCH}$$

Seek time is the average time required for the disk head to be positioned over the desired track (30 ms for an RM05). Latency is the average time required for the desired record to pass under the disk head after it has been positioned. This is about the time required for one half rotation of the platter (8.3 ms for an RM05). The data transfer time is the number of bytes to be transferred (512 times the bucket size) divided by the transfer rate (1.2 Mbytes/sec for an RM05). Finally, the bucket search time is the time required for the CPU to search through the bucket for the desired record once it is in memory.

Most of the I/O time is consumed by mechanical motion (seek + latency). This is a property of the hardware and cannot be changed. The four components of I/O are present for every I/O operation (except, in some cases, bucket search time).

Therefore, a major objective of tuning is to reduce the number of seeks required, since this is the largest single component of I/O.

#### Bucket Size Considerations

There are four factors which may cause a large bucket size to adversely affect performance:

1. data transfer time
2. bucket search time
3. memory constraints
4. bucket locking

Data transfer time is directly proportional to the bucket size. A quick calculation for the RM05 yields the following table:

| Bucket Size | Seek Time | Latency | Transfer Time | Total   | %  |
|-------------|-----------|---------|---------------|---------|----|
| 1           | 30 ms     | 8.3 ms  | .4 ms         | 38.7 ms | 1  |
| 10          | 30 ms     | 8.3 ms  | 4.3 ms        | 42.6 ms | 10 |
| 20          | 30 ms     | 8.3 ms  | 8.6 ms        | 46.9 ms | 18 |
| 32          | 30 ms     | 8.3 ms  | 13.7 ms       | 52.0 ms | 26 |

The last column is the percentage of the total time which is represented by data transfer. For small buckets, this is insignificant. However, as the bucket size is increased, the time spent actually transferring the data becomes more and more significant. For sequential and pseudorandom access, the higher data transfer time may be balanced by a smaller number of direct I/O's. In this case, the high data transfer time per I/O is not important because most of the bytes that are transferred will be used by the CPU, whether in one large access or many smaller ones. For random access, on the other hand, only one record in the bucket will be accessed (unless there are multiple buffers set up for caching). Therefore, most of the data transfer work is pure overhead. As a general rule of thumb, it is wise to limit the data transfer time to 10-15% of the total (in other words, keep the bucket size below about 15).

Bucket search time is only significant in random access applications. Sequential access will cause every record in the bucket to be read, while with pseudorandom access, only the first record in a group of records will have to be located in the bucket. From that point on, it will look like sequential access. For large bucket sizes, decreased I/O resulting from a flatter index must be balanced against the time required to search through the buffer for the desired record. The bucket search time is a function of the CPU type and record size.

Another consideration is memory constraints. For each file, one or more buffers are allocated in memory. One bucket of data from the file is transferred to a buffer for each I/O. The size of the buffer is therefore the same as the size of the bucket. For ISAM files, the minimum number of buffers is two, so for large bucket sizes, a large amount of memory must be allocated. For example, if the bucket size is 10, 10K bytes of memory must be available. The use of global buffers can reduce the total amount of buffer space required (this topic will be addressed later).

In a shared file environment, an additional tradeoff becomes important. When a file is opened for read/write access, RMS performs locking at two levels. While the record is initially being accessed, the entire bucket in which the record is contained is locked. Afterwards, the lock on the bucket is released and the lock on the record is retained until another operation takes place. While the time required to initially access a record may be short, a large bucket size means that you will be locking out large portions of the file for some period of time. If there is a great deal of contention, these bucket locks may adversely affect performance.

These considerations yield tradeoffs which must be evaluated in the context of the application before a choice for bucket size can be made. If the application is I/O bound, the amount of I/O can be reduced at the expense of a greater load on the CPU. If the application is CPU bound, the load on the CPU can be reduced at the expense of I/O. If the application is both I/O and CPU intensive, the tradeoffs must be carefully studied and a compromise reached. In the final analysis, it may be necessary to experiment to determine the optimal bucket size.

As a further note on bucket size, this parameter can be set to a different value for each area. Index areas do not need as large a bucket size as do data areas for optimal performance. However, the FDL editor takes the bucket size selected and applies it to all areas. Select the MODIFY option of the editor and change the bucket size for each individual area before exiting.

The FDL editor assists the user in selecting a bucket size by drawing a line or surface plot that shows the resulting number of index levels for the parameters selected. The documentation advises selecting a value between the slashes. It also indicates that increasing the bucket size generally will not improve performance unless the number of index levels decreases. This is true for random access, but not for sequential and pseudorandom access, where performance depends on more than just the number of index levels (as

noted above).

#### Fill Factor

To reduce the number of bucket splits which will occur with a file that is subject to frequent insertions, RMS allows you to reserve a certain amount of free space in each bucket when the file is initially loaded. This is done by selecting a fill factor between 50 and 100%, and is another tradeoff decision. A low fill factor will increase file size and may adversely affect performance in the short term. It is only after a significant number of bucket splits have been avoided that any advantage will be realized. Furthermore, if the insertions tend to cluster in certain areas of the file, then bucket splits will occur anyway. If the insertions are expected to be distributed evenly over the entire file, and the system time constraints are such that you cannot afford to CONVERT frequently, then, by all means, experiment with low fill factors. However, in most cases, optimal performance will result from setting the fill factor to 100% and CONVERTing the file as frequently as possible (this will depend on the expected number of updates).

Another consideration is the relative frequency of inserts versus retrievals. A low fill factor will improve performance if the frequency of insertions is relatively high. However, it is important to consider the relationship between record size and bucket size. If the record is large compared to the bucket, then reserving even a large amount of free space in the bucket will not accommodate many insertions. On the other hand, with small records and large buckets, even a moderate amount of free space can reduce the number of bucket splits without significantly increasing the size of the file.

#### Fixed vs. Variable Length Records

Variable length records contain two additional bytes of record length information per record. If there are many records in the file that are sparsely filled, then the overhead bytes may buy considerable space savings (and a smaller index for a given bucket size). Otherwise, fixed length records will be processed more efficiently. It is necessary to experiment to determine the average record size, including overhead bytes, of variable length records versus fixed length. The mean data length, in bytes, can be found in the ANALYSIS\_OF\_KEY section in an FDL file created by the ANALYZE/RMS utility. Unless the mean data length of the variable length record is significantly less than the fixed length, performance will be better with fixed length records.

## Buffers

There are two reasons for using buffers for disk I/O. For sequential processing, buffers are used to minimize the disparity between CPU speed and I/O. That is, several buffers are loaded initially. While the CPU is processing from one buffer, another buffer can be loading with the next group of records in anticipation of their need. When the CPU is finished, the buffers can be switched, so that there is a minimum amount of time that the CPU is waiting for I/O. This is called anticipatory double buffering, and is used with sequential files (read ahead/write behind processing). For this purpose, two buffers are usually sufficient.

The second use of buffers is for caching. For random access applications, the idea is to cache the index buckets and some of the data buckets. After the first access, one bucket from each level is retained in a buffer. On the next access, it is possible that the record is already in the buffer, obviating the need to perform actual I/O for that record. For truly random access, it is not likely that successive records accessed will be in the same bucket. However, it is likely that one or more index levels, especially the root level, will be cached. When access tends to cluster in areas of a file, a significant amount of I/O can be eliminated (the documentation points out that the caching algorithm favors the higher index levels).

There are two types of buffers under VMS: local process I/O buffers and global buffers. Local buffers are allocated by the process and only that process can access them. You allocate local buffers by the SET RMS\_DEFAULT command. Experiments with local buffers with pseudorandom access have shown, as would be expected, no performance advantage. Consequently, it is wise to default to the minimum of two in this case. However, if your application is random access, you should try using more than the minimum.

Global buffers offer advantages in a shared file environment. The idea here is that if you require access to a particular record, it is possible that someone else has cached one or more of the levels required. Again, the caching algorithm tries to keep at least the root level in memory at all times. We have not implemented global buffers in our application. Experiments with them on a single user basis have shown that their mere presence contributes a significant amount of CPU overhead. What that suggests is that global buffers should only be used if there are a large number of users that tend to access the same area of a file on a frequent basis. Also, global buffers will offer no advantage in a sequential or pseudorandom application.

Global buffers are a system-wide resource and their allocation is limited by certain system parameters. The documentation states that you may be able to reduce your total buffer requirements by allocating a single large global cache rather than many small local process caches. However, this cache must be large enough to cache at least the entire index in order for the ratio of "hits" to "misses" to be acceptably high. The FDL editor assists the user in determining the number of global buffers to allocate. However, this number will usually be far from optimal. Version 4 provides a more intelligent calculation of the number of pages required to cache the index, but the bottom line is the hit/miss ratio. This information can be accessed in the global buffer header by using an undocumented command in the System Dump Analyzer as follows:

```
ANALYZE/SYSTEM (requires CMKRNL privilege)
SDA> SHOW PROCESS/RMS=GBH
```

This will display the contents of the global buffer header, which includes the current number of hits and misses. If the number of hits is not very large compared with the number of misses, then either an insufficient number of global buffers have been allocated for the file or file access patterns are such that global buffers provide no advantage for the application.

## Contiguity

To minimize disk head movement, a file should be maintained as contiguous as possible. It is possible to specify, in the FDL file, that a file be created contiguous. However, if there is insufficient space to allocate the file contiguously, an error will occur. It is better to specify BEST\_TRY\_CONTIGUOUS. In that case, an attempt will be made to allocate contiguous space. If there is insufficient space, the file will still be allocated, but non-contiguously. Each individual area should also be allocated BEST\_TRY\_CONTIGUOUS.

## Allocation

RMS will calculate a reasonable initial allocation based on the answers provided in the FDL Editor regarding the number of records, record size, etc. An extension size of about 10% of the initial allocation is also calculated. This value can be overridden if necessary. The object is to prevent the file from fragmenting into non-contiguous extents, if the file has to be extended due to record insertions. If the extension size is too small, the file may have to be extended many times. With too large an extension size, there may not be enough contiguous space available. If the estimates of initial record count and number of

additional records are correct, extension size will not be an important issue. At any rate, the file header should be periodically examined and a contiguous copy of the file should be made if the number of extents becomes excessive.

## CASE STUDIES

All experiments were performed on a VAX 11/750 with 8 Mbytes of memory. The disk used was an RM05 256 Mbyte removable disk. Test runs were submitted as batch jobs in the middle of the night, when the system was relatively quiescent, to eliminate contention. Test programs were constructed which accessed the files in a pseudorandom manner. Data collection was effected by using the library routines, LIB\$INITTIMER and LIB\$SHOWTIMER. They were set up in the programs so as to isolate the I/O (the non-I/O CPU time was found to be negligible). These routines provided elapsed time, CPU time and direct I/O count. A DCL procedure was set up which, for each set of file parameters, performed the following:

1. the original, untuned file was CONVERTed, using an FDL file which was previously created with Edit/FDL
2. the FORTRAN test program was run 10 times, with logical assignments to the optimized file
3. ANALYZE/RMS/FDL/STATISTICS was run against the optimized file to collect further data (index depth, compression percentages, etc.)

The results were averaged over the 10 runs and assembled into a tables.

File 1

|                       |          |
|-----------------------|----------|
| Record size:          | 72 bytes |
| Key size:             | 10 bytes |
| Key data type:        | STRING   |
| Initial record count: | 261,632  |

The test program performed an initial keyed read for 16 randomly selected keys, followed by 150 sequential reads per key. The results for File 1 are presented in graphical form in Figures 1 and 2 and in tabular form in Table 1.

Since our application is both CPU and I/O intensive, it is required to minimize the total elapsed time for I/O (which includes both CPU and I/O components). The fact that there was no contention for either CPU time or I/O when the tests were run made the variance in the measurement of elapsed time reasonably small. In Figure 1, the number of direct I/O's are plotted against a reference number which, when used as an index into Table 1, yields the file

attributes.

Figure 2 shows the elapsed and CPU times for the same set of reference numbers. They are plotted in increasing order of elapsed time, with the shaded portion of each bar being the CPU time.

Several points are immediately apparent.

1. All of the files that were tuned with FDL resulted in significantly improved elapsed time and dramatic improvements in direct I/O count (as high as 89%).
2. Prologue 3 files with compression on actually resulted in an increase in CPU time, although the elapsed time and I/O count improved.
3. Prologue 2 files performed better than the corresponding Prologue 3 files with compression turned off.
4. Prologue 2 files, in general, provided the greatest improvement in both elapsed time and CPU time.

The object of the tests was to determine the optimal set of file parameters for our application, that is, those parameters which resulted in the best combination of elapsed time, CPU time and I/O count. To that end, the FDL file represented by the first bar in Figure 1 was selected, with the following results:

|                           |     |
|---------------------------|-----|
| Elapsed time improvement: | 29% |
| CPU time improvement:     | 18% |
| Direct I/O improvement:   | 86% |

File 2

|                       |           |
|-----------------------|-----------|
| Record size:          | 496 bytes |
| Key size:             | 13 bytes  |
| Key data type:        | STRING    |
| Initial record count: | 4293      |

The test program performed a keyed read for 11 randomly selected keys, followed by sequential reads until a control break occurred, which averaged 39 records per key. The tests were run in the same manner as for File 1. The results for File 2 are presented in Table 2.

The first entry in Table 2 is for the unoptimized file. The FDL file represented by reference number 30 was selected, with the following results:

|                           |     |
|---------------------------|-----|
| Elapsed time improvement: | 61% |
| CPU time improvement:     | 38% |
| Direct I/O improvement:   | 85% |



## CONCLUSIONS

1. Determine if compression can provide advantages for the application.
2. Determine the expected frequency of insertions/deletions.
3. Determine if time and space constraints permit a full CONVERT on a periodic basis rather than CONVERT/RECLAIM.
4. Select a Prologue version based the above, keeping in mind that Prologue 2 offers a performance advantage. (If alternate keys or non-STRING keys are required, use Prologue 2.)
5. Use the line or surface plot in the FDL Editor to select a range of bucket sizes with which to experiment. Select a bucket size for each area that yields the best experimental results. Keep in mind that a large bucket size can adversely affect performance in a shared file environment.
6. Determine if the mean data length can be reduced by using variable length records.
7. Select a fill factor with consideration given to conclusion number 2 (above). If possible, use a fill factor of 100% and CONVERT frequently.
8. Set the local buffer count based on the predominant access method.
9. Determine if the application can benefit from global buffers. If so, experiment with allocating global buffers and observe the effects on performance.
10. Try to allocate the files contiguously and maintain contiguity as much as possible.
- 11.

Use the ANALYZE/RMS utility periodically and adjust the FDL parameters if the index increases in depth.

## BIBLIOGRAPHY

1. Introduction to VAX-11 Record Management Services.
2. VAX-11 Record Management Services Tuning Guide.

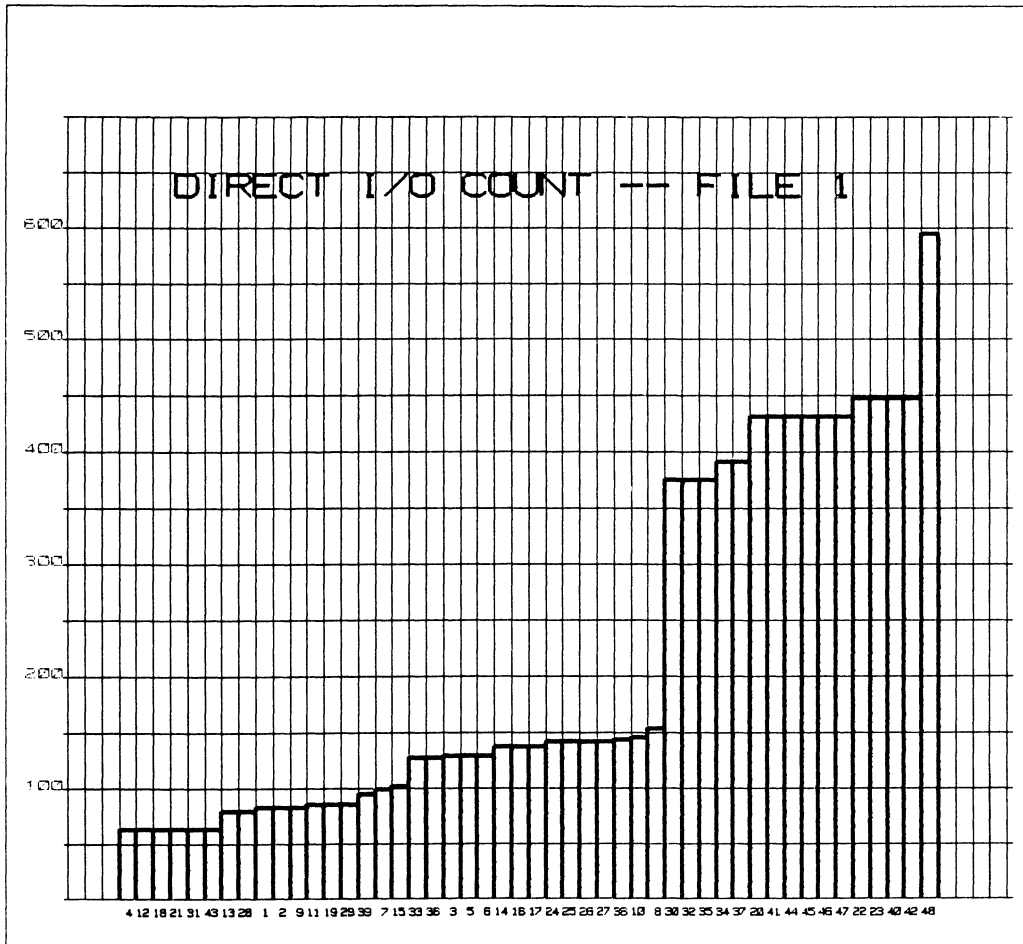
3. VAX-11 Record Management Services Utilities Reference Manual.
4. VAX-11 Record Management Services Reference Manual.

The above references are found in the VAX/VMS Documentation, Volume 7, Digital Equipment Corporation, Maynard, Massachusetts, May, 1982.

## ACKNOWLEDGEMENTS

The graphs in the Figures were produced on an ORCA 3000 CAD/CAM Workstation with the assistance of Bill Waters, MarketVision. The hard copy was produced on a SEIKO color printer.

Thanks are in order to Bill Adiletta, MarketVision, for his editorial comments.



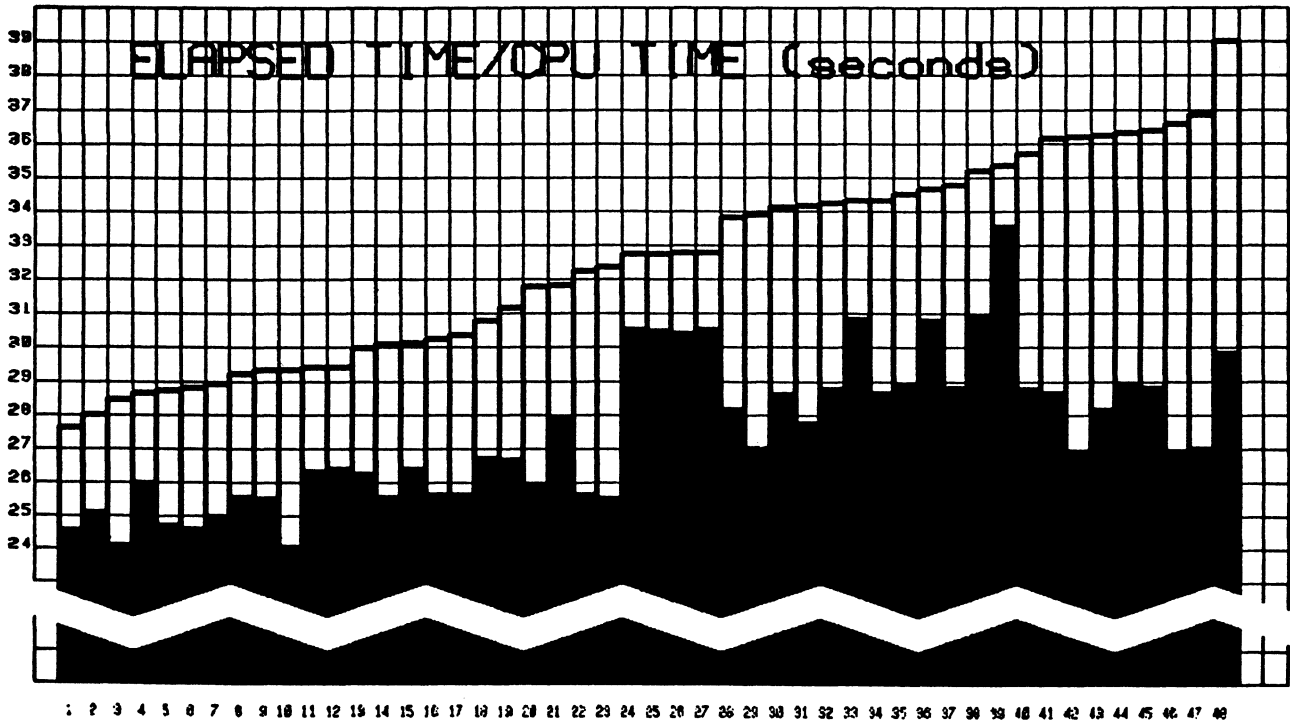


FIGURE 2 - ELAPSED TIME/CPU TIME FOR FILE 1

(refer to the text for a description)

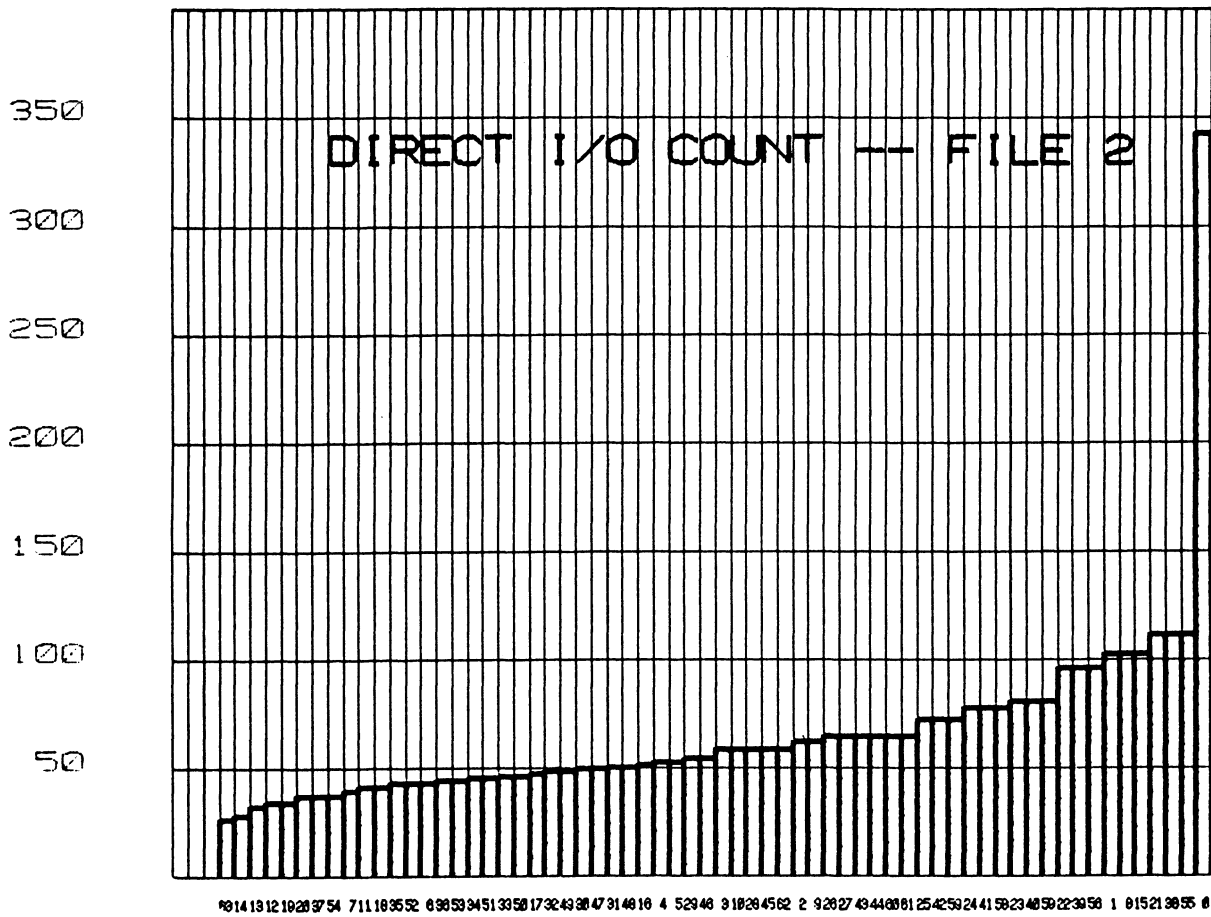


FIGURE 3 - DIRECT I/O COUNT FOR FILE 2

TABLE 1 - PERFORMANCE ANALYSIS RESULTS

| REF<br>NUMBER | INDEX<br>BUCKET<br>SIZE | DATA<br>BUCKET<br>SIZE | PROLOG | KEY<br>COMPR<br>(Y/N) | INDEX<br>COMPR<br>(Y/N) | DATA<br>COMPR<br>(Y/N) | EL.<br>TIME<br>(SECS) | CPU<br>TIME<br>(SECS) | DIR.<br>I/O<br>COUNT | INDEX<br>DEPTH |
|---------------|-------------------------|------------------------|--------|-----------------------|-------------------------|------------------------|-----------------------|-----------------------|----------------------|----------------|
| 1             | 4                       | 8                      | 2      | -                     | -                       | -                      | 27.63                 | 24.66                 | 84                   | 2              |
| 2             | 8                       | 8                      | 2      | -                     | -                       | -                      | 28.04                 | 25.16                 | 84                   | 2              |
| 3             | 4                       | 4                      | 2      | -                     | -                       | -                      | 28.47                 | 24.18                 | 131                  | 2              |
| 4             | 4                       | 13                     | 2      | -                     | -                       | -                      | 28.68                 | 26.06                 | 65                   | 2              |
| 5             | 13                      | 4                      | 2      | -                     | -                       | -                      | 28.74                 | 24.80                 | 131                  | 2              |
| 6             | 8                       | 4                      | 2      | -                     | -                       | -                      | 28.85                 | 24.68                 | 131                  | 2              |
| 7             | 1                       | 8                      | 2      | -                     | -                       | -                      | 28.94                 | 25.03                 | 100                  | 2              |
| 8             | 1                       | 4                      | 3      | NO                    | NO                      | NO                     | 29.26                 | 25.64                 | 155                  | 3              |
| 9             | 13                      | 8                      | 2      | -                     | -                       | -                      | 29.37                 | 25.59                 | 84                   | 2              |
| 10            | 1                       | 4                      | 2      | -                     | -                       | -                      | 29.38                 | 24.16                 | 147                  | 3              |
| 11            | 13                      | 8                      | 3      | NO                    | NO                      | NO                     | 29.43                 | 26.40                 | 87                   | 2              |
| 12            | 8                       | 13                     | 2      | -                     | -                       | -                      | 29.46                 | 26.68                 | 65                   | 2              |
| 13            | 1                       | 13                     | 2      | -                     | -                       | -                      | 30.00                 | 26.30                 | 81                   | 2              |
| 14            | 4                       | 4                      | 3      | NO                    | NO                      | NO                     | 30.13                 | 25.64                 | 139                  | 2              |
| 15            | 1                       | 8                      | 3      | NO                    | NO                      | NO                     | 30.16                 | 26.50                 | 103                  | 3              |
| 16            | 8                       | 4                      | 3      | NO                    | NO                      | NO                     | 30.29                 | 25.73                 | 139                  | 2              |
| 17            | 13                      | 4                      | 3      | NO                    | NO                      | NO                     | 30.41                 | 25.73                 | 139                  | 2              |
| 18            | 13                      | 13                     | 2      | -                     | -                       | -                      | 30.82                 | 26.77                 | 65                   | 2              |
| 19            | 4                       | 8                      | 3      | NO                    | NO                      | NO                     | 31.22                 | 26.72                 | 87                   | 2              |
| 20            | 13                      | 1                      | 2      | -                     | -                       | -                      | 31.82                 | 25.98                 | 433                  | 2              |
| 21            | 4                       | 13                     | 3      | NO                    | NO                      | NO                     | 31.84                 | 28.00                 | 65                   | 2              |
| 22            | 4                       | 1                      | 2      | -                     | -                       | -                      | 32.28                 | 25.69                 | 449                  | 2              |
| 23            | 1                       | 1                      | 2      | -                     | -                       | -                      | 32.40                 | 25.61                 | 449                  | 3              |
| 24            | 5                       | 3                      | 3      | YES                   | YES                     | YES                    | 32.78                 | 30.58                 | 143                  | 2              |
| 25            | 10                      | 3                      | 3      | YES                   | YES                     | YES                    | 32.79                 | 30.55                 | 143                  | 2              |
| 26            | 3                       | 3                      | 3      | YES                   | YES                     | YES                    | 32.82                 | 30.48                 | 143                  | 2              |
| 27            | 7                       | 3                      | 3      | YES                   | YES                     | YES                    | 32.82                 | 30.60                 | 143                  | 2              |
| 28            | 1                       | 13                     | 3      | NO                    | NO                      | NO                     | 33.83                 | 28.23                 | 81                   | 3              |
| 29            | 8                       | 8                      | 3      | NO                    | NO                      | NO                     | 33.91                 | 27.09                 | 87                   | 2              |
| 30            | 5                       | 1                      | 3      | YES                   | YES                     | YES                    | 34.17                 | 28.69                 | 376                  | 2              |
| 31            | 13                      | 13                     | 3      | NO                    | NO                      | NO                     | 34.20                 | 27.82                 | 65                   | 2              |
| 32            | 7                       | 1                      | 3      | YES                   | YES                     | YES                    | 34.25                 | 28.84                 | 376                  | 2              |
| 33            | 13                      | 4                      | 3      | YES                   | YES                     | NO                     | 34.34                 | 30.91                 | 129                  | 2              |
| 34            | 1                       | 1                      | 3      | YES                   | YES                     | YES                    | 34.35                 | 28.74                 | 392                  | 3              |
| 35            | 12                      | 1                      | 3      | YES                   | YES                     | YES                    | 34.52                 | 28.95                 | 376                  | 2              |
| 36            | 8                       | 4                      | 3      | YES                   | YES                     | NO                     | 34.67                 | 30.87                 | 129                  | 2              |
| 37            | 3                       | 1                      | 3      | YES                   | YES                     | YES                    | 34.82                 | 28.87                 | 392                  | 3              |
| 38            | 1                       | 4                      | 3      | YES                   | YES                     | NO                     | 35.24                 | 31.05                 | 145                  | 3              |
| 39            | 5                       | 5                      | 3      | YES                   | YES                     | YES                    | 35.39                 | 33.62                 | 96                   | 2              |
| 40            | 1                       | 1                      | 3      | YES                   | YES                     | NO                     | 35.71                 | 28.84                 | 449                  | 3              |
| 41            | 4                       | 1                      | 3      | YES                   | YES                     | NO                     | 36.17                 | 28.74                 | 433                  | 3              |
| 42            | 1                       | 1                      | 3      | NO                    | NO                      | NO                     | 36.20                 | 26.98                 | 449                  | 3              |
| 43            | 8                       | 13                     | 3      | NO                    | NO                      | NO                     | 36.25                 | 28.22                 | 65                   | 2              |
| 44            | 8                       | 1                      | 3      | YES                   | YES                     | NO                     | 36.33                 | 28.95                 | 433                  | 2              |
| 45            | 13                      | 1                      | 3      | YES                   | YES                     | NO                     | 36.40                 | 28.89                 | 433                  | 2              |
| 46            | 8                       | 1                      | 3      | NO                    | NO                      | NO                     | 36.59                 | 27.02                 | 433                  | 2              |
| 47            | 13                      | 1                      | 3      | NO                    | NO                      | NO                     | 36.87                 | 27.08                 | 433                  | 2              |
| 48            | 1                       | 1                      | 3      | NO                    | NO                      | NO                     | 39.05                 | 29.94                 | 596                  | 3              |

TABLE 2 - PERFORMANCE ANALYSIS RESULTS

| REF NUMBER | INDEX BUCKET SIZE | DATA BUCKET SIZE | PROLOG | KEY COMPR (Y/N) | INDEX COMPR (Y/N) | DATA COMPR (Y/N) | EL. TIME (SECS) | CPU TIME (SECS) | DIR. I/O COUNT | INDEX DEPTH |
|------------|-------------------|------------------|--------|-----------------|-------------------|------------------|-----------------|-----------------|----------------|-------------|
| 0          | 2                 | 2                | 3      | NO              | NO                | NO               | 13.45           | 6.96            | 343            | 2           |
| 1          | 2                 | 5                | 3      | YES             | YES               | YES              | 6.76            | 5.31            | 103            | 2           |
| 2          | 2                 | 10               | 3      | YES             | YES               | YES              | 6.29            | 5.18            | 63             | 2           |
| 3          | 2                 | 11               | 3      | YES             | YES               | YES              | 6.77            | 5.34            | 59             | 2           |
| 4          | 2                 | 13               | 3      | YES             | YES               | YES              | 6.34            | 5.27            | 53             | 2           |
| 5          | 2                 | 15               | 3      | YES             | YES               | YES              | 6.33            | 5.31            | 53             | 2           |
| 6          | 2                 | 17               | 3      | YES             | YES               | YES              | 6.50            | 5.51            | 44             | 2           |
| 7          | 2                 | 20               | 3      | YES             | YES               | YES              | 6.58            | 5.64            | 40             | 2           |
| 8          | 4                 | 5                | 3      | YES             | YES               | YES              | 7.04            | 5.29            | 103            | 2           |
| 9          | 4                 | 10               | 3      | YES             | YES               | YES              | 6.33            | 5.24            | 63             | 2           |
| 10         | 4                 | 11               | 3      | YES             | YES               | YES              | 6.46            | 5.32            | 59             | 2           |
| 11         | 4                 | 13               | 3      | YES             | YES               | YES              | 6.06            | 5.25            | 42             | 1           |
| 12         | 4                 | 15               | 3      | YES             | YES               | YES              | 6.17            | 5.39            | 35             | 1           |
| 13         | 4                 | 17               | 3      | YES             | YES               | YES              | 6.16            | 5.44            | 33             | 1           |
| 14         | 4                 | 20               | 3      | YES             | YES               | YES              | 6.31            | 5.57            | 29             | 1           |
| 15         | 6                 | 5                | 3      | YES             | YES               | YES              | 6.82            | 5.29            | 103            | 2           |
| 16         | 6                 | 10               | 3      | YES             | YES               | YES              | 6.06            | 5.17            | 52             | 1           |
| 17         | 6                 | 11               | 3      | YES             | YES               | YES              | 6.12            | 5.23            | 48             | 1           |
| 18         | 6                 | 13               | 3      | YES             | YES               | YES              | 6.06            | 5.23            | 42             | 1           |
| 19         | 6                 | 15               | 3      | YES             | YES               | YES              | 6.18            | 5.37            | 35             | 1           |
| 20         | 3                 | 25               | 2      | -               | -                 | -                | 6.77            | 4.44            | 38             | 2           |
| 21         | 4                 | 5                | 2      | -               | -                 | -                | 7.13            | 4.71            | 112            | 2           |
| 22         | 4                 | 6                | 2      | -               | -                 | -                | 7.01            | 4.69            | 97             | 2           |
| 23         | 4                 | 7                | 2      | -               | -                 | -                | 6.84            | 4.63            | 81             | 2           |
| 24         | 4                 | 8                | 2      | -               | -                 | -                | 6.46            | 4.52            | 78             | 2           |
| 25         | 4                 | 9                | 2      | -               | -                 | -                | 6.50            | 4.51            | 73             | 2           |
| 26         | 4                 | 10               | 2      | -               | -                 | -                | 5.47            | 4.40            | 65             | 2           |
| 27         | 4                 | 11               | 2      | -               | -                 | -                | 6.34            | 4.42            | 65             | 2           |
| 28         | 4                 | 12               | 2      | -               | -                 | -                | 6.35            | 4.51            | 59             | 2           |
| 29         | 4                 | 13               | 2      | -               | -                 | -                | 6.27            | 4.34            | 55             | 2           |
| * 30       | 4                 | 14               | 2      | -               | -                 | -                | 5.28            | 4.33            | 50             | 2           |
| 31         | 4                 | 15               | 2      | -               | -                 | -                | 6.25            | 4.45            | 51             | 2           |
| 32         | 4                 | 16               | 2      | -               | -                 | -                | 6.50            | 4.43            | 49             | 2           |
| 33         | 4                 | 17               | 2      | -               | -                 | -                | 6.38            | 4.46            | 47             | 2           |
| 34         | 4                 | 18               | 2      | -               | -                 | -                | 6.01            | 4.36            | 46             | 2           |
| 35         | 4                 | 19               | 2      | -               | -                 | -                | 5.87            | 4.36            | 44             | 2           |
| 36         | 4                 | 20               | 2      | -               | -                 | -                | 6.01            | 4.40            | 45             | 2           |
| 37         | 4                 | 25               | 2      | -               | -                 | -                | 6.28            | 4.43            | 38             | 2           |
| 38         | 5                 | 5                | 2      | -               | -                 | -                | 7.76            | 4.81            | 112            | 2           |
| 39         | 5                 | 6                | 2      | -               | -                 | -                | 6.21            | 4.69            | 97             | 2           |
| 40         | 5                 | 7                | 2      | -               | -                 | -                | 6.20            | 4.53            | 81             | 2           |
| 41         | 5                 | 8                | 2      | -               | -                 | -                | 6.00            | 4.49            | 78             | 2           |
| 42         | 5                 | 9                | 2      | -               | -                 | -                | 6.00            | 4.49            | 73             | 2           |
| 43         | 5                 | 10               | 2      | -               | -                 | -                | 5.54            | 4.45            | 65             | 2           |
| 44         | 5                 | 11               | 2      | -               | -                 | -                | 5.74            | 4.43            | 65             | 2           |
| 45         | 5                 | 12               | 2      | -               | -                 | -                | 5.55            | 4.37            | 59             | 2           |
| 46         | 5                 | 13               | 2      | -               | -                 | -                | 5.51            | 4.39            | 55             | 2           |
| 47         | 5                 | 14               | 2      | -               | -                 | -                | 5.31            | 4.32            | 50             | 2           |
| 48         | 5                 | 15               | 2      | -               | -                 | -                | 5.45            | 4.32            | 51             | 2           |
| 49         | 5                 | 16               | 2      | -               | -                 | -                | 5.40            | 4.32            | 49             | 2           |
| 50         | 5                 | 17               | 2      | -               | -                 | -                | 5.36            | 4.33            | 47             | 2           |
| 51         | 5                 | 18               | 2      | -               | -                 | -                | 5.44            | 4.35            | 46             | 2           |
| 52         | 5                 | 19               | 2      | -               | -                 | -                | 5.42            | 4.36            | 44             | 2           |
| 53         | 5                 | 20               | 2      | -               | -                 | -                | 5.47            | 4.39            | 45             | 2           |
| 54         | 5                 | 25               | 2      | -               | -                 | -                | 6.12            | 4.50            | 38             | 2           |
| 55         | 6                 | 5                | 2      | -               | -                 | -                | 6.96            | 4.82            | 112            | 2           |
| 56         | 6                 | 6                | 2      | -               | -                 | -                | 6.03            | 4.63            | 97             | 2           |
| 57         | 6                 | 7                | 2      | -               | -                 | -                | 6.38            | 4.52            | 81             | 2           |
| 58         | 6                 | 8                | 2      | -               | -                 | -                | 6.62            | 4.59            | 78             | 2           |
| 59         | 6                 | 9                | 2      | -               | -                 | -                | 7.11            | 4.52            | 73             | 2           |
| 60         | 6                 | 10               | 2      | -               | -                 | -                | 6.18            | 4.46            | 65             | 2           |
| 61         | 6                 | 11               | 2      | -               | -                 | -                | 6.11            | 4.45            | 65             | 2           |
| 62         | 6                 | 12               | 2      | -               | -                 | -                | 5.66            | 4.40            | 59             | 2           |
| 63         | 6                 | 25               | 2      | -               | -                 | -                | 5.36            | 4.36            | 27             | 2           |









RESULTS AND COMPARISONS IN MULTIPROCESSING  
USING VMS 4.0 AND MA780\*

Nancy E. Werner  
Lawrence Livermore National Laboratory  
P.O. Box 808, MS L306  
Livermore, CA 94550

Abstract

Experiments using different parallel processing techniques on selected parallel algorithms were performed. Relative performance of these techniques was observed. The hardware was 4 clustered Vax-780s with 14 to 16 Megabytes each of local memory and 4 Megabytes of shared memory (2 MA780s).

INTRODUCTION

Parallel processing is the ART of doing multitasking on more than one processor, where multitasking is the splitting up of a Job into many separate Tasks. Normally these Tasks need to communicate with each other in order to complete the Job. In a tightly coupled system, they will use Shared Memory for communication. In a loosely coupled system, they will send messages to each other via a common bus such as the CI Bus. With the present hardware, four clustered Vax-780s with 16 Megabytes, 14 Megabytes, 14 Megabytes local memory respectively and 4 Megabytes shared memory, either method of communication could be used. Only the tightly coupled method has been pursued so far and will be discussed here.

MOTIVATION

Lawrence Livermore National Laboratory (LLNL) also has on site a four processor CRAY computer, the XMP-48. It would be nice if users could become familiar with parallel processing on a cheaper, friendlier and more accessible environment than is yet offered on the CRAY. The Vax System's main purpose is for parallel processing research; there are no production jobs to worry about, and dynamic debugging tools are available.

Most potential users at LLNL are not familiar with VMS. In order to lure them onto the VAX System, it was necessary to imitate the environment of the CRAY as much as possible. CRAY users were using a set of primitives devised by Cray Research Inc. (CRI) which were referred to as the "CRI Multitasking Primitives"[1]. These are simply a library of routines which were designed to be used for implementing parallel processing algorithms. A similar library was implemented on the VAX System to be as consistent as possible with the CRI library [2]. Programs which run on the CRAY, with minor modifications, can also run on the VAX System, within memory size limitations. Programs have been debugged on the VAX System and then successfully moved onto the CRAY.

PARALLEL PROCESSING DEFINITIONS

There are some basic things one must do for parallel processing that are not necessary for sequential processing. It must be possible to define Tasks which can execute on the available processors. The consistency of the shared data must be assured; simultaneous updates to the same data must be avoided. A section of code that alters shared data must be executed by only one processor at a time; such a section of code is called a **Critical Section**. The Tasks often must synchronize their activities with each other. A place at which Tasks need to meet before proceeding with the computation is called a **Barrier**.

A **Logical Processor** is a process which has been initiated at Job submittal time and is scheduled by the Operating System on the VAX on which that process resides. A **TASK** is an instantiation by a **Logical Processor** of a subroutine call with shared memory arguments. When the **TASK** has completed (returned), the **Logical Processor** is free to activate another **TASK**.

To implement a **Critical Section**, a **LOCK** can be used. A **LOCK** is a resource protector; only one **TASK** at a time is allowed to have a specified **LOCK**. If a **LOCK** is gotten before entering a section of code, then anyone else attempting to get that **LOCK** must wait until it is released. When the **Critical Section** of code has been completely executed, the **TASK** should then release the **LOCK** to allow another **TASK** which has also requested this **LOCK** to proceed. The same **LOCK** should be used for related **Critical Sections** which affect the same data.

**Barriers** can be implemented using **EVENTs** and/or **LOCKS**. An **EVENT** is a system wide signal that can be set, tested and cleared by all **TASKs** working on this Job. There are many ways to implement a **Barrier**. One example is given in Appendix B.

PARALLEL PROCESSING LIBRARY

The library implemented for VMS contains not only those subroutines as defined by CRI, but also by

\* This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore under Contract No. W-7405-Eng-48.

necessity ,some special subroutines which must be used to map and unmap predefined areas of data to the shared memory. To facilitate this mapping, the user must place all his shared data into a common block named /SHAREDGLOBAL/. The library will also map some of its own data to shared memory. Other subroutines were added to provide more functionality.

#### DATA DEFINITIONS

The data was set up to be compatible with the CRI definitions as far as possible. Please note that an integer is 32 bits on the VAX and 64 bits on the CRAY.

**Taskarray:** 2-3 integers used to hold TASK information as follows:

count: number of integers in this array (set by user)

pointer: pointer to library data for this TASK (used only by library)

Taskvalue: optional value associated with this Task (set by user)

**Name:** subroutine name (entry point for TASK instantiation)

**List:** argument list for subroutine (addresses of arguments {must be in shared memory})

**Taskvalue:** user defined value (32 bits)

**Lockdata:** integer used to represent a LOCK (defined by user, manipulated by library)

**Eventdata:** integer used to represent an EVENT (defined by user, manipulated by library)

**First\_shr:** address of the first data item in shared memory  
( %loc(\*) where common/sharedglobal/\*.../ )

**Last\_shr:** address of the last data item in shared memory  
( %loc(\*) where common/sharedglobal/\*...\*/ )

**Dbug:** integer flag which will cause library to write tasking information to the log file if > 0

#### SPECIAL SUBROUTINES

**Task\_init** (First\_shr, Last\_shr, Dbug)

Subroutine which maps the shared data to shared memory. It must be called before any shared data is used. It also sets up the Task\_cleanup subroutine as an exit handler.

#### Task\_cleanup

Subroutine which unmaps the shared memory.

#### CRI COMPATIBLE SUBROUTINES

**Tskstart** (Taskarray, Name [,List])

Startup the TASK associated with Taskarray by calling subroutine Name with arguments List.

**TskWait** (Taskarray)

Wait for the TASK associated with Taskarray to complete.

Logical = **Tsktest** (Taskarray)

If the TASK associated with Taskarray exists, then set Logical = .true.

**TskValue** ( Taskvalue )

Retrieve the value for this TASK.

**Lockasgn** ( Lockdata )

Assign and initialize the LOCK associated with Lockdata.

**Lockrel** ( Lockdata )

If there are no waiters release the LOCK associated with Lockdata

otherwise set error condition.

**Lockon** ( Lockdata )

If the LOCK associated with Lockdata is busy then wait,

otherwise get the LOCK.

**Lockoff** ( Lockdata )

Relinquish the LOCK associated with Lockdata.

Logical = **Locktest** ( Lockdata )

If the LOCK associated with Lockdata was already on, set Logical = .true. and return

otherwise get the LOCK and set Logical = .false.

**Evasgn** ( Eventdata )

Assign and initialize the EVENT associated with Eventdata.

**Evrel** ( Eventdata )

If there are no waiters, release the EVENT associated with Eventdata

otherwise set error condition.

**Evpost** ( Eventdata )

Post the EVENT associated with Eventdata.

**Evclear** ( Eventdata )

Clear the EVENT associated with Eventdata.

**Evwait** ( Eventdata )

Wait for the EVENT associated with Eventdata to be posted.

Logical = **Evtest** ( Eventdata )

If the EVENT associated with Eventdata was posted, Logical = .true.

otherwise Logical = .false.

Another set of subroutines was implemented to facilitate dynamic partitioning of the Job's work among the TASKS. A unique set of mailboxes may be set up with specified message size with which TASKS may communicate. Dynamic partitioning is achieved by dividing the work up and then putting the pieces into a mailbox queue; each TASK that is doing that work can then retrieve pieces of the work until the work is completed and the mailbox is empty.

**Setup\_sr** (Mbx\_array, Count, Mbx\_size, Code)

Set up Count mailboxes whose message size is Mbx\_size and whose unique identifier will be Code. The channel number for each mailbox initialized will be placed in Mbx\_array(1 - Count).

**Send\_sr** ( Mbx\_array(I), Buffer, Msg\_size)

Send the message with size Msg\_size which had been placed in Buffer to the mailbox referred to by Mbx\_array(I).

**Receive\_sr** ( Mbx\_array(I), Buffer, Msg\_size)

Wait for a message from the mailbox referred to by Mbx\_array(I) to be read into Buffer, whose size is Msg\_size.

## UTILITIES

A set of utilities have been implemented to make Parallel Processing on the VAX system easier for the user. These utilities are really command files which have had symbols defined for them.

**Cricomplink** Program Compiler

This utility will compile and link Program using the compiler indicated by compiler, which may include optional parameters. It is helpful to use this utility because it handles special linking problems caused by shared memory access.

**Crisetup** Program Maxlog

This utility creates a set of command files for setting up the environment of this Program and also some debugging command files for use with the Parallel Debugger which assumes that the maximum number of logical processes used will be Maxlog. This needs to be executed only once for this Program unless the Parallel debugger is being used and Maxlog needs to be larger.

## ADDITIONAL SUBROUTINES

These subroutines can be used to obtain a chronological log of what is happening during a Job. If the Job is running on more than one VAX, the time is not a correct indicator since each VAX has a separate clock and they are not synchronized.

**Open\_shared** ( Unit, Filename, Record\_size)

The file Filename is opened as a shared relative file with maximum record size = Record\_size and associated with Unit. If being called by Fortran, Unit is also the unit number.

**Close\_shared** ( Unit )

The file associated with Unit is closed and reset.

**Get\_nxtrec** ( Unit, Record\_number, Count)

The next Count records are reserved on the file associated with Unit and the first of these record numbers is returned to Record\_number.

## **Crilogicals** Program

This utility defines the logical names necessary to map to the shared memory. If this is not executed, local memory will be used exclusively.

## **Crisubmit** Program Logcpu Physcpu [After\_time]

This utility starts up Program in Logcpu processes (logical cpus) on Physcpu processors (physical cpus) at time = After\_time if present, otherwise now. In VMS terms, a command file which was setup for this program when Crisetup was executed, which in turn executes Crilogicals and then runs Program, is submitted to a generic batch queue Logcpu times. The generic queue will alternate the submittal amongst queues on Physcpu VAXs. Thus there will be Logcpu processes running, divided evenly amongst Physcpu processors.

## **Cricleanup**

This utility needs to be executed only if there was an abnormal exit or the user wishes to abort the Job. Using a command file which was generated when Crisubmit was executed, it will remove any left over batch processes and delete shared memory access for the last Job submitted.

## **Cridebug** Program Logcpu Physcpu

This utility starts up Program in Logcpu processes (logical cpus) on Physcpu processors (physical cpus) with the Parallel Debugger enabled.

### IMPLEMENTATION

The implementation of the Parallel Processing Library on VMS was done using shared memory to store library information and interlocked instructions to update this information. Normally, temporary mailboxes in shared memory were used, which automatically go away when the Job completes. The shared memory must be mapped to a permanent global section and thus must be specifically deleted by the exit handler when the Job completes.

The root TASK of a parallel processing program is the program itself, all other TASKs are subroutines within that program. The same copy of the program is executed in all of the processes. The first process to execute the call to Task\_init becomes the root TASK. The root TASK creates the necessary shared memory global sections, creates the Tasking mailbox and associates an exit handler for Job cleanup and termination. After the return from Task\_init, it will continue executing the program. All other processes become slaves. A slave process

also executes the call to Task\_init, but after mapping to the shared memory global sections and to the Tasking mailbox which had been created by the root TASK, it will then perform a read on the Tasking mailbox and wait for a message. The slave processes will never proceed beyond the call to Task\_init except to make subroutine calls requested by the Tasking mailbox message. Whenever Tskstart is called, a message is placed in the Tasking mailbox which indicates the subroutine Name to be called and its arguments. One and only one of the slave processes will receive that message; if a TASK is to be started, it will set up a taskblock for that TASK in shared memory, mark it valid, and then generate a call to that subroutine with the appropriate arguments. Upon returning from that subroutine, it will mark the TASK done. In order to wake up any other TASKs which might be waiting for this TASK, it will create a unique mailbox associated with that TASK, write to it, and then delete it. Having finished the business of that TASK, it will read the Tasking mailbox again to look for another TASK to do. When the process receives a DONE message in the Tasking mailbox, it will pass the message on and then commit suicide. The root TASK will automatically place the initial DONE message in the Tasking mailbox when it is finished by automatically using the exit handler that was set up by the call to Task\_init.

The EVENT mechanism merely uses the VMS Common Event Flag clusters in shared memory.

The LOCK was implemented two ways. The first implementation did not care about the order in which the lock was granted. A LOCK was obtained by performing an interlocked decrement on the semaphore represented by Lockdata. If the LOCK was available, the process proceeded, otherwise it waited for an Event Flag which had been associated with that LOCK by the Lockasn call. Lockoff did an interlocked increment on this semaphore and then set the Event Flag associated with this LOCK. Whichever TASK reacted the fastest got the LOCK next, there was no fairness criteria. This implementation appeared to be sufficient for a while. Later, a program, which used Locking in its Barrier implementation and synchronized on Barriers frequently, displayed very erratic behavior when executing on all 4 processors. This behavior was finally traced to a semi "starvation" effect caused by the unfair Locking mechanism. Processes were waiting excessively long within the Barrier due to lack of fair access to the LOCK which was used in that Barrier implementation. The Barrier was rewritten without the use of LOCKs and the erratic behavior disappeared. However, because of the possible "starvation" problem, it was decided to re-implement the Locking mechanism using an interlocked first in, first out (FIFO) queue. In addition to a semaphore, an interlocked queue was associated with each LOCK, both were represented by Lockdata. If a LOCK is not available, the Pid of the process requesting the LOCK and the nodename of its Processor are placed in the queue. When the LOCK becomes available, an entry is removed from the queue for that LOCK; if the waiting process resides on the same Processor, it is awakened. If the waiting process is on a different Processor from the process relinquishing the LOCK, a message is placed in a permanent shared mailbox associated

with that Processor. A server process responds to this message and wakes up the appropriate process on its Processor. The Program was retried with the old Barrier implementation and the new Lock implementation. The previous erratic behavior was not observed.

In order to allow processes to record their behavior in a synchronous manner, a set of subroutines was implemented which allows the user to easily write ordered records to a shared relative file; which may then be printed or otherwise interrogated. The last record used is noted in a shared memory array indexed into by the Unit number for that file.

To make dynamic load balancing easier, a set of subroutines was implemented for setting up, and reading and writing to shared mailboxes. The user must determine what information is necessary to indicate the next work item and must put that information into a buffer of appropriate size. After the mailbox has been setup, subsequent writes and reads to/from this mailbox will enter and remove items to/from the work queue represented by this mailbox.

## RESULTS

Using the Parallel Processing Library described above, experiments were performed to investigate the benefits and the costs of parallel processing. For benchmarking purposes, two methods were used to implement Barriers ( see Appendix A). These methods were implemented in assembly language in order to make them as fast as possible. Another method was implemented and tested using EVENTS and the Parallel Processing Library (see Appendix B); while using the other Barrier method, the elapsed time did not vary significantly from the first two Barrier methods. The first Barrier method, Method E, relinquishes the CPU (Processor) when it must wait at a Barrier and waits for an Event Flag associated with this Barrier to be set. The second Barrier method, Method S, spins, testing the shared memory location associated with this Barrier until it is ready. The difference in CPU time used by the two methods is the time that is spent waiting for the other Tasks to reach the Barrier.

The cost of using Barriers can be broken into components. There is the cost of the extra computations necessary to implement the Barrier; there is the cost of waiting within the Barrier due to resource contention, and there is the cost of waiting for the other TASKs to reach the Barrier. The last component can be estimated by using both kinds of Barriers and comparing the CPU times used. The second component is negligible for Method S, since the only resource contention present is a single interlocked decrement that occurs for each TASK when it first reaches the Barrier. Method E will have more resource contention due to its use of event flags. The first and second components were estimated by timing 100 loops of 60 consecutive Barrier calls, for Method E and for Method S. This test was run with from 1 to 4 TASKs, each with its own Processor. Method S took approximately .0001

seconds per Barrier, per TASK, no matter how many TASKs were running simultaneously. Method E took longer due to its use of event flags. As more TASKs participated, this became worse due to the added resource contention; its time varied from .0006 seconds to .002 seconds depending on the number of TASKs participating.

Assuming a UNIT of COST to be the cost of a single +, -, \* floating point type operation, the COST of a Lockon followed by a Lockoff, the COST of an Evpost followed by an Evclear, and the COST of the Barriers S and E were measured, varying the number of participating TASKs, each with its own Processor, from 1 to 4. See Table 1 for complete results. The COST for Locking varied from 9 to 65 units, depending on the number of TASKs, due to LOCK contention. The COST of Events varied less, from 33 to 46 units. The COST of Barrier E varied from 37 to 106 units, but the COST of Barrier S remained fairly constant at 7 units. Even though Barrier S appears to be cheaper, further results showed that the first two components of cost of a Barrier, which this test measures, are not the most important. Also, if the Processors are being shared, Barrier S would be wasting CPU cycles that others could be using. Another interesting side result of this experiment was that Barrier S, which uses shared memory heavily, did not degrade the performance of the System. This would seem to indicate that a potential hardware problem, shared memory contention, was not a problem in these experiments.

A standard LLNL benchmarking code named Simple, a hydrodynamic calculation with heat conduction, was used for further investigations. A grid size of 80 x 100 was used for 100 time cycles. There were 14 Barrier synchronizations performed per time cycle. Both Barrier methods (S,E) were used. The number of TASKs (Processors) working on the problem varied from 1 to 4. The number of Logical Processors actually working on the problem was never greater than the number of Physical Processors. No other users were on the System during benchmarks. At each Barrier call, for each TASK, data was saved indicating when the Barrier was entered and when it was exited. Upon termination of the Job, this data was processed. All the Barrier delays were summed and averaged among the number of TASKs (WTave). Also, at each Barrier, the maximum delay amongst the participating Tasks was found, and these were summed (WTM). From the logs, the CPU usage for Barrier E was subtracted from the CPU usage for Barrier S and the difference was divided by the number of participating Tasks to give the average wait at a Barrier (Wave). With complete parallelism, if one TASK takes X seconds to complete the Job, then N TASKs should take X/N seconds. If T is the time that it actually took to run the Job with N TASKs, then let D be the Discrepancy, where  $D = (T - X/N)$ . The speedup is usually a measure of how much parallelism was actually achieved.  $Speedup(N) = \text{Elapsed time for one TASK} / \text{Elapsed time for N TASKs}$ .

The first experiments were done using fixed partitioning of the work load. The work was divided up equally amongst the Tasks before starting the Job. The time for 1 Task to complete the Job was 1600 seconds. A Job was run using 4 Tasks and bypassing the synchronization; the answers were wrong, but the Speedup was = 4! Using the Barrier synchronizations,  $Speedup(2) = 1.95$ ,  $Speedup(3) =$

2.85, Speedup(4) = 3.7. The Speedup did not vary significantly as a function of the Barrier implementation method used, including the one in Appendix B. For complete results, see Table 2. The total cost of the Barriers (first and second components) of this Job is approximately = cost of a single Barrier X 14 X 100. Therefore, Barrier E cost from .8 to 2.8 sec depending on the number of Tasks. Barrier S cost approximately .14 sec. In any case, the cost is < 1% of the total cost of the Job. Then, why isn't the Speedup better? It appears that the third component of the Barrier cost, the wait at the Barriers for the other TASKs, is the primary expense in Barrier synchronization for this Job. Even if this Job has exclusive use of the System, it still does not have exclusive use of the Processors. The Operating System must continue to do its work ( cluster management, accounting, error logging, etc). Bare in mind that there is not 1 Operating System, but rather 4 Operating Systems involved. If any of these Operating System uses CPU cycles, the TASK being run under that Operating System will be delayed, and all other TASKs will have to wait for the delayed TASK when a Barrier is encountered. Figure 1 shows a scatter plot showing the distribution of the sizes of the maximum waits at the Barriers, looking at all 14 barriers, but only 20 cycles worth of data. Figure 2 shows a plot of how the size of the maximum wait at a single Barrier varies, using the same 20 cycles worth of data.

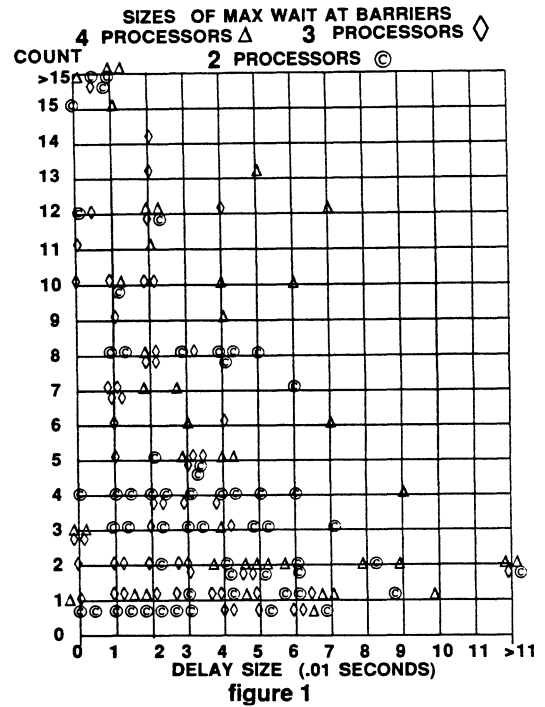


Table 1

COST OF PARALLEL PROCESSING

Definition: UNIT of COST = one \*,-,+ Operation

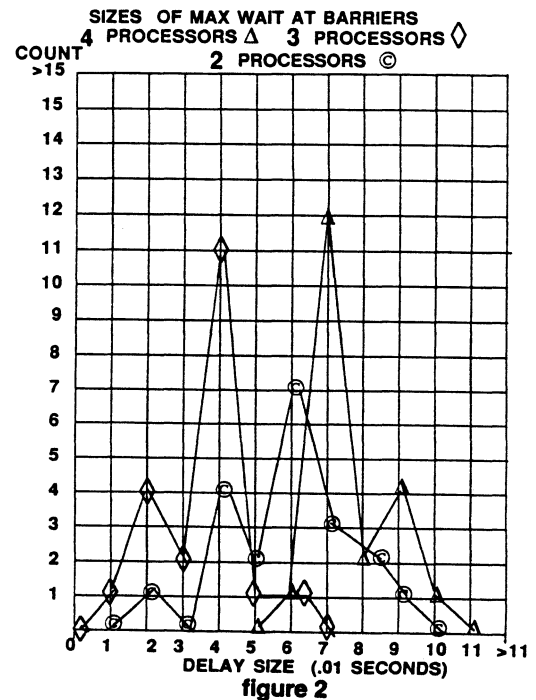
| FUNCTION           | COST (1 CPU) | COST (2 CPUS) | COST (3 CPUS) | COST (4 CPUS) |
|--------------------|--------------|---------------|---------------|---------------|
| LOCK (ON/OFF)      | 9            | 10            | 42            | 65            |
| EVENT (POST/CLEAR) | 33           | 43            | 46            | 44            |
| BARRIER (E)        | 37           | 78            | 92            | 106           |
| BARRIER (S)        | 6            | 7             | 7             | 7             |

If the Barriers are few, with a large amount of work being done in between, these delays might statistically even out amongst the TASKs, causing less delay at the Barriers due to waiting for each other. In other words, the larger the granularity of the problem between Barriers, the more efficient use the Job will make of the Processors.

Table 2

SIMPLE WITH FIXED EQUAL PARTITIONS

| #PROCESSORS | SPEEDUP | ELAPSED TIME       | D Wave | WTave | WTM |
|-------------|---------|--------------------|--------|-------|-----|
| 1           | 1.00    | 1600               | 00 00  | 00    | 00  |
| 2           | 1.95    | 820                | 20 15  | 20    | 38  |
| 3           | 2.85    | 560                | 27 13  | 15    | 30  |
| 4           | 3.70    | 430                | 30 16  | 21    | 45  |
| 4           | 4.00    | NO SYNCHRONIZATION |        |       |     |



Work partitioning was investigated next. The work was divided up into Work Queues with a fixed number of items in each queue. Each TASK is allowed to remove items from the queue until that work is completed, at which time a Barrier is usually encountered. As items are removed from one Work Queue and worked on, they are generally inserted into the next work queue. It was found that the overhead cost of using Work Queues was approximately .003 seconds or 120 COST UNITS per Work Item, per Barrier (see Table 3).

Table 3

SIMPLE WITH DYNAMIC PARTITIONING

OVERHEAD OF WORK QUEUES (1 PROCESSOR)

| # WORK ITEMS | ADDED ELAPSED TIME (seconds) |
|--------------|------------------------------|
| 01           | 000                          |
| 20           | 080                          |
| 40           | 170                          |
| 60           | 250                          |
| 80           | 350                          |

OVERHEAD approximately = 4 seconds per Work Item, per Job  
 = .003 seconds or 120 Cost UNITS per Work Item, per Barrier

If a Job takes 1600 seconds for 1 Processor to complete, the best that can be done with 4 Processors would be 400 seconds. The Overhead can be estimated to be the Actual CPU Usage per Processor - Best Possible CPU Usage per Processor. The Delay at the Barriers is again estimated by comparing the CPU usage of the two different Barrier implementations. As we can see from Figure 3, as the number of work items in the Work Queue increases, the delay at the Barriers tends to go down, but the overhead goes up. In fact, from Figure 4, it is evident that for this Job, the Fixed Partitioning Method is superior to the Dynamic Partitioning Method. The Overhead of Dynamic Partitioning exceeds any Delays caused by the work load imbalance of this Job.

**SIMPLE**  
**4 PROCESSORS**

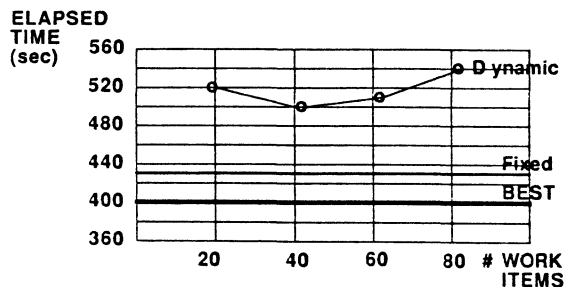


figure 4

The expected Elapsed Time for a Job can be estimated to be the average CPU used (per Processor) plus the average Delay at the Barriers. We can then guess the CPU utilization for this Job when it is executing to be the Estimated Elapsed Time divided by the Actual Elapsed Time. These results are shown in Table 4. The Job was monitored during its execution and the estimated CPU utilization numbers were actually observed to be true. It is assumed that not only was the Job doing more work when using Work Queues, but the Operating System was also doing more work. Even though Work Queues were not the most efficient implementation of this Job in an exclusive environment, that does not mean that they won't be for another Job which has to deal with greater load imbalance problems or even, perhaps, has to share the system with "other" users!

Table 4

SIMPLE, FIXED EQUAL PARTITIONING VERSUS DYNAMIC PARTITIONING

| BARRIER METHOD | # WORK ITEMS | ACTUAL ELAPSED TIME | CPU AVE | OVERHEAD | Wave | WTave | WTM | EST. ELAPSED TIME | %CPU UTI. |
|----------------|--------------|---------------------|---------|----------|------|-------|-----|-------------------|-----------|
| FIXED          |              | 430                 | 405     | 05       | 16   | 21    | 45  | 421               | 98        |
| DYNAMIC        | 20           | 520                 | 425     | 25       | 80   | 80    | 200 | 505               | 97        |
|                | 40           | 500                 | 445     | 45       | 40   | 40    | 115 | 485               | 97        |
|                | 60           | 510                 | 465     | 65       | 25   | 25    | 55  | 490               | 96        |
|                | 80           | 540                 | 485     | 85       | 30   | 30    | 100 | 515               | 95        |

SUMMARY

Many things were learned from these parallel processing experiments. Mostly, it was learned that parallel processing is not easy. The difficulties in constructing a "correct" program and knowing that it is indeed "correct" were not even discussed. Some factors that might influence implementation techniques were explored. The efficiency of these

**SIMPLE**  
**4 PROCESSORS**

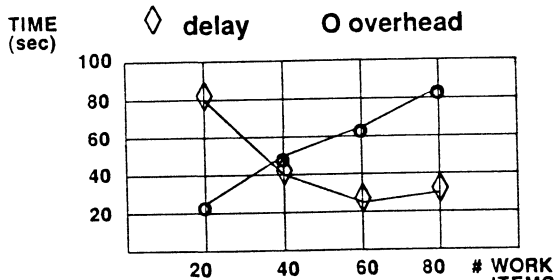


figure 3



techniques depends not only on the problem being solved, but on the architecture of the computer being used to solve it. To avoid unnecessary overhead and delays, synchronization should be minimized whenever possible. New mathematical algorithms need to be designed with this in mind. General schemes for solving parallel processing problems will need to be modified to suit each parallel processing environment. At present, the programmer is almost totally responsible for finding and explicitly declaring the parallel processing capabilities of his job. Eventually compilers will assist, if not relieve, the programmer of that responsibility. There are still many unknowns concerning the suitability of computer architectures, computer algorithms, and computer software for solving the problems inherent in parallel processing. Experimenting with parallel processing will give some useful insights into the problem.

#### REFERENCES

- [1] "Multitasking User's Guide", Cray Research, Inc., Mendota Heights, MN Sn-0222
- [2] Werner, N.E., Van Matre, S.W., "Parallel Processing on the Livermore VAX 11/780-4 Parallel Processor System with Compatibility to Cray Research, Inc. (CRI) Multitasking", Version 1, UCRL-92624, May 1985

#### DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

## BARRIER METHOD E

## DATA:

3 pairs of integers, ordered,  
 <THIS,NEXT,LAST>  
     Synchronization\_variable  
     Event\_flag\_number

Number\_of\_TASKs = number of TASKs  
                   synchronizing

## INITIALIZATION:

Set Synchronization\_variable(s) to  
                                   Number\_of\_TASKs  
 Clear event flags = Event\_flag\_number(s)  
 Initialize the order of the data items  
     THIS = now  
     NEXT = next to be used  
     LAST = last one used before now

## SYNCHRONIZE:

Decrement (interlocked)  
     THIS Synchronization\_variable

If THIS Synchronization\_variable not = 0,  
     then wait for THIS Event\_flag\_number

Otherwise Post THIS Event\_flag\_number

## RESET:

Set LAST Synchronization\_variable to  
                                   Number\_of\_TASKs  
 Clear LAST Event\_flag\_number  
 Rotate the order of the Data Items

THIS <-- NEXT <-- LAST  
 |\_\_\_\_\_↑

## DATA:

3 ordered integers, <THIS,NEXT,LAST>  
 Number\_of\_TASKs = number of TASKs  
                   synchronizing

## INITIALIZATION:

Set Synchronization\_variable(s) to  
                                   Number\_of\_TASKs  
 Initialize the order of the data items  
     THIS = now  
     NEXT = next to be used  
     LAST = last one used before now

## SYNCHRONIZE:

Decrement (interlocked) THIS  
                                   Synchronization\_variable

Test THIS Synchronization\_variable  
     until THIS Synchronization\_variable = 0

## RESET:

Set LAST Synchronization\_variable to  
                                   Number\_of\_TASKs  
 Rotate the order of the Data Items

THIS <-- NEXT <-- LAST  
 |\_\_\_\_\_↑

## Appendix B

### BARRIER METHOD USING THE PARALLEL PROCESSING LIBRARY

#### DATA:

```
C Array of 3 ordered event flag numbers for
C each participating TASK

Integer Event_numbers(3, Number_of_TASKs)

Integer THIS,NEXT,LAST

Integer Number_of_TASKs, Task_id
Number_of_TASKs = number of Tasks
 synchronizing

Task_id = This TASK's identification number
where 0 < Task_id < Number_of_TASKs + 1
```

#### INITIALIZATION:

```
C Clear this TASK's event flags

DO I = 1, 3

Call Evclear (Event_numbers(I, Task_id))

End do

C Initialize the order of the data items

 THIS = 0 ; use now
 NEXT = 1 ; next to be used
 LAST = 2 ; last one used before now
```

#### SYCHRONIZE:

```
C Signal that this Task is ready

Call Evpost (THIS, Task_id)

C Wait for all Tasks

Do I = 1, Number_of_TASKs
Call Evwait (THIS, I)
End Do
```

#### RESET:

```
C Reset appropriate last signal

Call Evclear (LAST, Task_id)

C Rotate Data Items

LAST = THIS
THIS = IMod (THIS + 1, 3)
NEXT = IMod (THIS + 1, 3)

Return
```

**MICROCOMPUTER EMULATION ON THE VAX  
IMPLEMENTATION AND MANAGEMENT OF A VIRTUAL MICROCOMPUTER SYSTEM**

**John J. Vasconcelos, Supervising Engineer  
and  
Ali T. Diba, Senior Software Engineer  
James M. Montgomery, Consulting Engineers, Inc.  
250 N. Madison Ave.  
Pasadena, California 91101**

**ABSTRACT**

Several third party vendors provide plug-in microprocessor boards and emulation software for the VAX which enable VAX users to run many of the popular CP/M and MS-DOS software packages from a terminal. For many VAX installations, this is a very cost-effective manner of obtaining microcomputer capability. This paper describes our experiences in implementing and managing one such system on our VAX network. Some of the problems we encountered and their solutions are also discussed. Advantages and disadvantages of microcomputer emulators are briefly discussed.

**INTRODUCTION**

The large base of user friendly microcomputer software can be accessed by VAX users at relatively low incremental cost by installation of plug-in microprocessor boards creating what amounts to a virtual microcomputer on a host VAX. The virtual machine has access to all the host machine's peripheral devices including large amounts of disk storage and has the benefit of regular back-up on the host machine.

James M. Montgomery, Consulting Engineers, Inc. (JMM), a consulting environmental engineering firm, has dual networked VAXes for running engineering, scientific, accounting/billing and data base management computing applications. Repeated requests from management and accounting personnel for microcomputers to run spreadsheet, project management, and other user friendly microcomputer software packages provided the impetus for investigating the implementation of virtual microcomputers on the VAX. The conclusion reached was that a virtual microcomputer system would be more cost effective than dedicated microcomputers for most routine applications.

**MICROCOMPUTING NEEDS**

The JMM VAX user community consists of three major groups. The engineering and scientific users which use Fortran or sophisticated scientific applications such as RS-1, the accounting users which use Cobol or Datatrieve, and a novice user group which has in the past used the computer primarily for sending and receiving electronic mail. This last group included a large percentage of managers and accounting personnel who expressed a need for spreadsheet, project management, and other financial/management software. Their computing needs could most easily be satisfied by the user friendly software packages available for microcomputers.

JMM presently has approximately a dozen microcomputers, the majority of which are located in domestic and foreign branch offices. The computer users in the Pasadena headquarters and several of the larger branch offices have direct access to the VAX computer network. For these users, the alternative of a microcomputer emulator on the VAX was considered the most cost effective method of providing microcomputing capabilities.

**REVIEW OF AVAILABLE EMULATORS**

There are presently three principal suppliers of microcomputer emulators for the VAX. They all provide both CP/M and MS-DOS emulation hardware and software. One of the first emulators on the market was a software emulator. However, software emulators have a slower response time which resulted in the development of hardware emulators in the form of microprocessor boards. Table 1 summarizes the attributes of CP/M emulators which are currently available.

Table 1  
CP/M Emulators

|                                         |
|-----------------------------------------|
| Software Emulators                      |
| Memory: 64 Kbytes                       |
| No. of Virtual Drives: 4                |
| Maximum Virtual Floppy Size: 4 Mbytes   |
| Hardware Emulators                      |
| Microprocessors: Z80 or Z80H            |
| Memory: 64 Kbytes                       |
| No. of Virtual Drives: 4-8              |
| Maximum Virtual Floppy Size: 4-8 Mbytes |

With the increasing popularity of MS-DOS and the widespread availability of MS-DOS software packages it was inevitable that MS-DOS emulators become available. At least one of these claims PC-DOS compatibility. Table 2 summarizes the attributes of the available 16 bit microcomputer emulators.

Table 2  
MS-DOS or CP/M-86 Emulators

Microprocessors: 8086 or 8088  
Memory: 128 to 768 Kbytes  
No. of Virtual Drives: 4 to 9  
Maximum Virtual Floppy Size: 4 MBytes

Fairly complete review articles on CP/M and MS-DOS emulators have appeared recently (1,2). For more details, the reader is referred to these articles.

#### ADVANTAGES AND DISADVANTAGES OF EMULATORS

Microcomputer emulators offer a number of potential advantages for an existing VAX installation, the most important of which is probably cost. Below are listed the most important advantages of virtual microcomputer emulators as compared to stand alone microcomputers.

##### Advantages of Microcomputer Emulators

- Lower cost per user
- Better file security
- Automatic backup of files
- Sharing of data facilitated by network
- Abundant mass storage

Our experience at JMM illustrates the potential savings which may be realized by implementation of a virtual microcomputer system. We presently have invested approximately \$25,000 in our virtual microcomputer system; \$16,000 in hardware and \$9,000 in software. Approximately 500 employees have accounts on our VAX computer system. Of these, approximately 50 use our virtual microcomputer system with some regularity with some 10 of these being heavy users. We presently have four CP/M and four MS-DOS (CPM-86) microprocessors installed with two additional MS-DOS microprocessors to be installed shortly. It is rare that more than two of the CP/M processors and three of the MS-DOS processors are in use simultaneously. For an investment equivalent to five stand alone microcomputers, the routine microcomputing needs of 50 users can be satisfied. Software costs are reduced since a single copy of a software package with a multiple user license can serve the needs of many users.

Since the microcomputer emulation runs as a process under VMS, it has the benefit of all the VMS utilities and features including password security, and regular backup. Virtual floppies are stored as RMS files in a VMS directory and as such can be further protected from access by unauthorized users. It is common for careless users of stand alone microcomputers to leave floppy disks with confidential data lying about but this is not possible with a virtual microcomputer system.

On the other hand, authorized users have rapid access to a central data base via VMS file transfer utilities and DECNET bypassing the communications headaches experienced by many heavy users of microcomputers. Furthermore, the Rainbow office work station can be easily integrated into the system via its automatic file backup features.

The virtual microcomputer has access to all the VAX peripherals including printers and Winchester disks. With virtual floppies as large as 8 Mbytes and VAX hard disks available, a microcomputer emulator has access to essentially unlimited mass storage by microcomputer standards.

There are, however, disadvantages to virtual microcomputers as compared to stand alone microcomputers related, in part, to the fact that they are operating in a mini/mainframe environment. The principal ones are summarized below.

##### Disadvantages of Microcomputer Emulators

- Slower execution times
- Inconvenient printout
- Limited software selection
- Excessive dependence on a central facility

Our experience at JMM has been that microcomputer emulators are noticeably slower than stand alone microcomputers. Our first software CP/M emulator was very slow and could be compared to modem communications with a microcomputer at 300 baud. Hardware emulators are significantly faster but our experience has been that performance is degraded on a loaded system. The reason for this is that all I/O is handled by VMS including terminal I/O and screen updates. A recent upgrade now allows logging directly into the microprocessor board which is said to improve performance noticeably.

For microcomputer users accustomed to an attached printer, the comparative inconvenience of printing out to a lineprinter queue can be disconcerting. Lineprinters have fewer options for formatting output than most microcomputer printers. Typically, print jobs are fed to the user's default printer queue by a print spooler. It is possible to print out to a graphics printer attached to a terminal with a printer port, but the procedure is more involved than printing to a microcomputer printer.

Initially, the selection of software available for microcomputer emulators was very limited but continues to grow. Generally, it is necessary to configure the software package for the emulator although one of the vendors claims that their emulator runs PS-DOS software. It is safest to purchase software pre-configured by the supplier of the emulator.

One of the big advantages of the distributed processing concept offered by microcomputers is the increased reliability that redundancy offers. Installing microprocessor boards on a central host computer puts all your microcomputing "eggs in one basket", so to speak. At JMM we have split our microprocessor boards between our two VAXes offering a measure of redundancy. A balanced system of microprocessor boards on central host computers networked with integrated microcomputers offers the most cost effective and reliable system.

#### SYSTEM IMPLEMENTATION

Implementation of a virtual microcomputer system involves more than installing the microprocessor boards and reading the software onto the system.

There are several distinct and important steps which can be identified. These can be categorized as follows:

- Hardware Installation
- Set-up Directory Structure
- Menu Development - DCL Command Files
- Documentation and Help Screens
- User Training

Installation of the microprocessor boards is a relatively straight forward process with vendor supplied installation documentation being fairly complete. Installation consists basically of plugging the board into an available slot, although one system comes on two boards and requires adjacent slots. All vendors provide hardware diagnostic routines which are used to check out the boards after installation is complete.

Once installation of the hardware is complete, public directories are created for storing the emulation software which is stored as executable images and the command files for running the emulation software. The MS-DOS and CP/M operating system and applications software is stored as RMS files which are structured as read only virtual floppies. User data storage virtual floppies are stored in CP/M or MS-DOS sub-directories of each user's home directory. These sub-directories are automatically created the first time the user invokes the emulator. Separate CP/M and MS-DOS directories were created to keep CP/M and MS-DOS virtual floppies separated.

To serve the needs of a largely novice user community, it was decided to develop a user friendly interface for invoking and running the microcomputer emulator. This was accomplished by developing a menu driven interface via DCL command files and a series of system logicals and symbols. For instance, to invoke the CP/M emulator, the user simply types the symbol \$ CPM <CR>. He is then presented with a series of menus to assist him in loading the desired microcomputer application package and data virtual floppy. Samples of menus developed for our system are appended as Figures 1 - 15.

One of the big challenges in providing user support was the issue of documentation. The software applications supplied by the vendors are provided with a multiple user license, but multiple copies of documentation are not supplied and copying of copyrighted documentation is not legal. A two pronged approach was adopted to meet this challenge. First, help screens were developed for the VAX HELP utility to provide general on-line help for the user. Next, brief summaries of the more commonly used commands were prepared for a number of software applications. Sample summaries entitled The JMM BRIDGE and TURBO PASCAL are appended.

Additional training for users in the form of in-house seminars on selected software applications has been provided. Classes on PASCAL and MULTIPLAN have been presented and have been well attended. A MICRO\_MANAGER has also been designated to which users with problems can send electronic mail describing any problems.

## PROBLEMS AND SOLUTIONS

In the nearly two years that JMM's virtual microcomputer system has been in operation, a number of problems have been experienced. Some of these problems are inherent in the implementation, and some are solvable.

### Software Configuration Problems

None of the microcomputer emulators presently available for the VAX are completely PC-DOS compatible although at least one claims to be. Occasionally, one of our users brings in a diskette and asks us to copy it on the system. Most of the time it does not run at all and as it usually turns out, it is written to run under PC-DOS. The safest course to take unless you are proficient in Z80 or 8088 assembly language is to purchase preconfigured applications from the software vendor or obtain generic MS-DOS or generic CP/M applications which usually come with a configuration module

### Complicated Printout Procedure

This issue was discussed in detail under emulator disadvantages. To simplify print out, a menu driven command procedure was developed and is invoked by the Getfile option on the Main Menu. Sample screens of a Getfile session are appended (Fig. 13 - 15).

### High I/O Overhead

All terminal input and screen updates are handled by VMS so, in essence, the VAX is serving as a terminal server for the microprocessor. This not only slows down the emulator as previously discussed, but results in high I/O costs. A recent upgrade allows logging directly into a port on the microprocessor board and should increase speed and reduce I/O overhead significantly

### X-on, X-off Protocol Problems

When X-on, X-off protocol is enabled, problems occur with some applications packages, an example being MULTIPLAN. In Multiplan, the Control-Q sequence moves the cursor to the home position. It also stops terminal scrolling. The alternatives are either to use the GoTo command as a work-around or disable X-on, X-off and accept the attendant risks.

### System Messages Trapped

During the early stages of system implementation, users occasionally lost data files during emergency system shut downs because system messages are trapped by the emulator. This problem was resolved by implementing a command procedure which displays currently logged in Z-board users by typing the symbol \$ ZUSERS, in a manner similar to the SHOW USERS command. This allows the operators to quickly determine emulator users who can then be called and asked to save open files and log off.

### Emulation Lock-up

Occasional lock-ups have been experienced by users of both the CP/M and MS-DOS emulators which have

resulted in the loss of data files. It has sometimes been necessary to stop the process which corrupts the data virtual floppy and results in a loss of all work done since the last daily backup. A recent upgrade for the MS-DOS emulator allows the user to abort the process without corrupting the virtual floppy with the result that only work done since the last file save is lost. We advise users to save files frequently when working in MS-DOS to minimize data loss if a lock-up occurs. For CP/M users, we recommend making personal back-ups of their current virtual floppy frequently when working on important projects. This is accomplished by exiting the CP/M application and making a duplicate virtual floppy under a different name using the VMS COPY command.

#### CONCLUSIONS

We at JMM have found our virtual microcomputer system a very cost effective method of providing routine microcomputing services with the additional advantages provided by the VAX host of better file security, automatic file backup, abundant mass storage, and ease of sharing data via the network.

There are, to be certain, disadvantages to the system including, slower speed of execution, printout inconveniences, limited software selection, and excessive dependence on a central computing facility. For these reasons as well as for special applications and for isolated branch offices, JMM has and will continue to acquire stand alone microcomputers.

#### REFERENCES

1. Scott, K. and Campbell, P., "Running CP/M Under VMS", Hardcopy, Vol. 13, No. 2, February, 1984, Pages 50-52.
2. Slaughter, P.H., "MS-DOS RIDES THE Q-BUS", Digital Review, Vol. 2, No. 4, January, 1985, Pages 78-82.

FIGURE 1

```
$ SET DEFAULT CC_17500:[JJV.CPM]
$ SHOW DEFAULT
 CC_17500:[JJV.CPM]
$ CPM
```

A>EXIT

```
$ SET DEFAULT CC_17500:[JJV.CPM86]
$ SHOW DEFAULT
 CC_17500:[JJV.CPM86]
$ MICRO CPM86
```

FIGURE 2

M...A...I...N.....M...E...N...U

Welcome to the JMM CPM Application Main Menu

This menu lets you choose between running CPM software or running CPM utilities.

Type H for help if you are a new user for more information on CPM.

- (R)un - Run CPM software
- (U)tilities - Run CPM utilities
- (H)elp - Obtain help
- (Q)uit - Exit

Enter your choice (R/U/H/Q): █



R....U....N.....M....E....N....U  
Specify the file to load in DRIVE A

- (M)aster - To use the Master system disk
- (B)lank - To copy and use a blank floppy
- (F)ile - To specify a file name
- (S)oftware - To see the available softwares

Enter your choice (M/B/F/S): S

FIGURE 4

Following Software products are available:

- BASIC
- FPLANNER
- INVENTORY
- MILESTONE
- MULTIPLAN
- POWERMP
- TALISMAN

Enter your choice: MULTIPLAN

FIGURE 5

R....U....N.....M....E....N....U  
Specify the file to load in DRIVE B

- (M)aster - To use the Master system disk
- (B)lank - To copy and use a blank floppy
- (F)ile - To see the available floppies
- (S)oftware - To see the available softwares

Enter your choice (M/B/F/S): F

FIGURE 6

Following virtual floppies are available:

BASPROG  
BUDGET  
CLASS  
FINREPTS  
LABMASTER  
MASTERS  
MILESTNE  
MPTEST  
MYMP  
PCBBUD  
PLANNER  
PLANTEST  
RLA1  
RLA2

Enter complete file name (e.g., MYFILE): MPTEST

FIGURE 8

```
A....T....T....A....C....H.....M....E....N....U
A>PUB_PUBROOT:[CPM.SOFTWARE.VT100]MULTIPLAN.VFL;
B>CC_17500:[JJV.CPM]MPTTEST.VFL;
```

- (S)oftware - Attach a software product
- (F)ile - Attach a file
- (R)eturn - Go back to CPM
- (M)ain - To go back to the main menu

Enter your choice (S/F/R/M): M

FIGURE 9

M....A....I....N.....M....E...N....U

Welcome to the JMM CPM Application Main Menu

This menu lets you choose between running CPM software or running CPM utilities.

Type H for help if you are a new user for more information on CPM.

(R)un - Run CPM software  
(U)tilities - Run CPM utilities  
(H)elp - Obtain help  
(Q)uit - Exit

Enter your choice (R/U/H/Q): Q

\$ █

FIGURE 10

\$ SET DEFAULT CC\_17500:[JJV.MSDOS]  
\$ SHOW DEFAULT  
CC\_17500:[JJV.MSDOS]  
\$ MICRO MSDOS █

A>HOST (OR LOGOUT) █

M....A....I....N.....M....E...N....U

Welcome to the JMM MSDOS Application Main Menu

This menu lets you choose between running MSDOS software or running MSDOS utilities.

Type H for help if you are a new user for more information on MSDOS.

- (R)un - Run software
- (U)tilities - Run utilities
- (H)elp - Obtain help
- (G)etfile - Extract/Print a file
- (Q)uit - Exit

Enter your choice (R/U/H/G/Q): U

FIGURE 12

U....T....I....L....I....T....I....E....S.....M....E....N....U

- (C)opy - Copy virtual floppies
- (F)ilelink -
  - View the directories of virtual floppies w/o loading MSDOS
  - Transfer files between the VAX files and the Virtual floppies
  - Create a new virtual floppy of any size
  - Write-protect or write-enable a virtual floppy
- (T)ranslate - Translate a file between the VAX and MSDOS formats
- (D)isklink - To transfer files between virtual and physical floppies
- (P)oly-XFR - To transfer files between a local computer and a host computer system
- (M)ain - To go back to the main menu

Enter your choice (C/F/T/D/P/M): M

M....A....I....N.....M....E...N....U

Welcome to the JMM MSDOS Application Main Menu

This menu lets you choose between running MSDOS software or running MSDOS utilities.

Type H for help if you are a new user for more information on MSDOS.

- (R)un - Run software
- (U)tilities - Run utilities
- (H)elp - Obtain help
- (G)etfile - Extract/Print a file
- (Q)uit - Exit

Enter your choice (R/U/H/G/Q): G

FIGURE 14

Enter the name of your virtual floppy: MPTEST  
 Enter the name of the MSDOS file in MPTEST: TEST.PRN  
 Extracting: TEST.PRN  
 TEST.PRN as TEST.PRN

Translating : TEST.PRN

Done .....

Enter (Y)es if you want to print the file: Y  
 Enter (Y)es if you want to save the file after printing: N  
 Job 5831 entered on queue TXA2  
 Enter (Y)es if you want to get another file: N

FIGURE 15

M...A...I...N.....M...E...N...U

Welcome to the JMM MSDOS Application Main Menu

This menu lets you choose between running MSDOS software or  
running MSDOS utilities.

Type H for help if you are a new user  
for more information on MSDOS.

(R)un - Run software  
(U)tilities - Run utilities  
(H)elp - Obtain help  
(G)etfile - Extract/Print a file  
(Q)uit - Exit

Enter your choice (R/U/H/G/Q): Q

\$

Print job 5831 TEST completed on 23-MAY-1985 10:58

\$

# RMS INDEXED FILE PERFORMANCE

**Harold T. Glaser, P.E.**  
Supervising Software Engineer

**Philip A. Naecker, P.E.**  
Manager of Information Services

**Pamela A. Valentine**  
Senior Software Engineer

**Gary Friedman**  
Associate Software Engineer

**James M. Montgomery, Consulting Engineers, Inc.**  
P.O. Box 7009  
Pasadena, California 91109-7009

## ABSTRACT

This paper presents information which can be used for optimization of VAX-11 RMS indexed files. Special attention will be paid to features available under VMS Version 4 for file design and tuning. Parameters which affect file performance will be discussed. Alternative approaches for file design and tuning are compared and evaluated to show what type of improvement in load time or retrieval performance might be anticipated for specific applications. Performance data will be presented for alternative file sizes and index structures in order to permit comparison to user applications.

## INTRODUCTION

There are a number of advantages in using indexed files under VAX-11 RMS. These advantages are particularly noteworthy when comparing indexed files to the more typical sequential file structure, especially where random access to records and data are desired.

### o Performance

Random access searches for records are significantly faster because of the availability of an index which utilizes a key for location of the desired record(s).

### o Write sharing

Applications can be constructed in which users can access an indexed file in a shared write mode, allowing record locking to control concurrency.

### o Automatic sorting of records

Because RMS maintains keyed index data structures of the file in sorted order, the records will already be sorted for the primary key.

### o Variety of datatypes

A variety of VMS datatypes are now available for indexed files which offer considerable flexibility in application development and use.

Certain disadvantages to the use of indexed files should be stated:

### o Disk space overhead

Because RMS must maintain an index data structure for the primary and any alternate keys, additional disk space is required. This creates an overhead which is not present in sequential files.

### o Disk access only

Indexed files are limited to disk storage only, as opposed to sequential files which may be stored on tape and disk media.

### o Complex programming

Programming and maintenance of indexed files is more complex than that required for sequential files. While the utilities and features provided in recent versions of VMS, RMS and certain layered products have reduced this complexity, one would have to argue that indexed files require more work than sequential files.

### o Organization not obvious

The organization of indexed files, especially those with multiple keys is not obvious to the novice user or application developer.



It has been our experience at JMM that the advantages of indexed files far outweigh the disadvantages, especially in an environment heavily oriented toward interactive applications. Indexed files are well-suited to the performance required by engineering, scientific and business users in conducting random searches, retrievals, on-line query and reporting. In addition, several tools are available for optimizing the performance of indexed files, including new features provided in Version 4.0 and later of VMS which significantly enhance application development and use.

The topics covered in this paper include:

- o A comparison of indexed and sequential file performance in random access searches.
- o A comparison of techniques for loading indexed files.
- o RMS file tuning considerations with an emphasis on the file bucket size and RMS multibuffer count parameters.
- o Key compression and the use of Prolog 3 files.
- o RMS Run-time Options available under Version 4 of VMS.

A special emphasis has been placed those features which are available under Version 4. Every attempt has been made to provide data which compares Version 4 performance with that under Version 3.

#### EXPERIMENTAL PROCEDURE

Two different databases were used for the experiments presented in this paper. The JMM personnel database was used for the first set of examples and benchmarks. The database for the large application discussed at the end of this paper is described in detail later. The employee database resides in an ISAM (Indexed Sequential Access Mode) file with the following attributes:

JMM Employee Database

- o Fixed Length - 110 byte records
- o 5 keys:
 

|            |                                  |
|------------|----------------------------------|
| Primary    | Employee Number                  |
| Alternates | Last Name                        |
|            | Computer Initials (VAX Username) |
|            | Supervisor Employee Number       |
|            | Cost Center                      |
- o 1961 Records Total

The experiments were conducted on a VAX-11/750 with 8 MB main memory, RA-81 disk drives and a UDA-50 controller. Results are shown for tests run under both Version 3.6 and Version 4.0 of VMS. VAX-11 Datatrieve was used to collect the benchmark data because it has convenient features for measuring performance and because of the overall convenience of working with an interactive query language. It

should be stressed that conclusions drawn for RMS file performance observations made using DTR are also generally valid for any procedural language (for example VAX-11 FORTRAN, COBOL, etc.) which utilizes RMS for record management. The following DTR procedure was developed for conducting benchmarks:

```
DTR> READY EMPLOYEES
DTR> FIND A IN EMPLOYEES SORTED BY
 LAST_NAME, FIRST_NAME
DTR> FN$INIT_TIMER
DTR> FOR B IN A PRINT NAME OF EMPLOYEES -
 WITH EMPNO EQ B.EMPNO ON NL:
DTR> FN$SHOW_TIMER
```

The objective of the procedure is to FIND a Datatrieve collection which has been inverted in some arbitrary manner with respect to the original keyed file. In this case, the employees' name is used, which is essentially random with respect to employee number, which is used in the retrieval, "PRINT NAME OF...". The procedure then loops through each record in the collection and references records via the key of interest by outputting a field to the null device. A schematic representation of the random access technique invoked by this procedure is shown in Figure 1.

The DTR FN\$INIT\_TIMER and FN\$SHOW\_TIMER functions are invoked to initialize and then display the totals for CPU Time, Elapsed Time, Buffered and Direct Input/Output and Page Faults. For most of the results in this paper, CPU and Elapsed Time and Direct I/O are reported. Rigorously speaking, CPU time and Direct I/O units are the most objective measures of performance. Elapsed Time figures are reported only where experiments were conducted on an unloaded system as a subjective, but useful, measure of response time.

It should also be noted that there is an overhead associated with the instructions necessary to perform the FOR loop and PRINT commands. This overhead is incurred regardless of whether the file is tuned or not and should be subtracted from the performance data because it tends to skew the observations. While we have not subtracted the overhead for the results in this paper, our tests show it is significant.

#### FILE DESIGN

One of the first considerations in file design is whether or not to index. Normally, it is reasonable to anticipate big performance gains in random record retrievals, even for relatively small files, by indexing. This should be qualified by stating that it is true for applications which access records in a random manner. In sequential files, the application must search the entire file each and every time an arbitrary record is requested based on the value of a field. In indexed files, an index is used to point to records by keyed fields which reduces the sequential search to the index.

The results presented here demonstrate the performance advantages of indexed files over sequential files for primary and alternate key access:

**Comparison of Indexed and Sequential  
File Performance  
Random Record Access  
Version 3.6 Data**

|                               | <b>Primary<br/>Key</b> | <b>Alternate<br/>Key</b> |
|-------------------------------|------------------------|--------------------------|
| CPU Time<br>(hh:mm:ss.cc)     |                        |                          |
| ISAM                          | 1:52.53                | 2:57.34                  |
| Sequential                    | 2:39:49.72             | 2:20:32.28               |
| Direct I/O<br>(I/O requests)  |                        |                          |
| ISAM                          | 5,341                  | 12,502                   |
| Sequential                    | 56,872                 | 58,900                   |
| Elapsed Time<br>(hh:mm:ss.cc) |                        |                          |
| ISAM                          | 3:43.76                | 7:50.67                  |
| Sequential                    | 3:32:20.58             | 3:44:24.71               |
| Storage Space<br>(blocks)     |                        |                          |
| ISAM                          | 716                    |                          |
| Sequential                    | 422                    |                          |

Primary key performance gains of 80:1 and 10:1 were observed for CPU Time and Direct I/O, respectively. The alternate key performance also showed significant gains. One might note, however, that the index file occupies more disk space than the sequential file because of the need to allocate space for the index structures.

Despite the small cost in space, this is convincing data on behalf of indexing where access is random. In the sequential file case, RMS had to search the entire file for each random access, whereas in the indexed case it was only necessary to access the desired records through the indexed key. It is important to note that record accesses on non-keyed fields of an ISAM file are implemented sequentially. This means there is no advantage in organizing a file using keys if the application does not reference those keys. (Datatrieve, incidentally, automatically uses keys when it can). Furthermore, it should be noted that the advantage of indexed files over sequential files improves substantially with increasing numbers of records in the file. This is because RMS will still be required to search the larger files one record at a time in the sequential file while the index structure is constructed in a multiple level fashion for large files which optimizes keyed searches.

In designing files, enough keys should be selected to ensure indexed access under most anticipated applications. Of course the additional indices cost you increased file storage space and/or CPU Time to build the indices when records are added, but often the trade-off in performance is worthwhile. In designing applications, it is difficult to anticipate which keys will be accessed most frequently. Because ISAM files can be restructured with new keys fairly easily, it is possible to drop seldom used keys or add new ones once the application is in use. Therefore, we recommend the designer continue to monitor the performance of files, once the application is put into use. It has been our experience that the clue

to a poorly performing application is often improper key selection or implementation of a new application on an existing file for which the keys are inappropriate.

**FILE LOADING**

An important consideration in the design of applications using ISAM files is the technique used for loading these files. Loading an indexed file with new records is required at the beginning of an application. It is also a process which may be called for periodically as a maintenance item during the life of the application. In this paper, two methods for loading indexed files will be evaluated. The first method utilizes the DTR RESTRUCTURE command and the second method utilizes VAX-11 RMS CONVERT.

The technical aspects of the DTR RESTRUCTURE command and CONVERT utility are covered in other papers<sup>1,2,3</sup> as well as the VAX-11 Datatrieve User Guide and VAX-11 Guide to File Applications. The results from a performance comparison test between DTR RESTRUCTURE and CONVERT for loading the same indexed file are presented in Figures 2-4 for Elapsed Time, CPU Time and Direct I/O for both Version 3 and Version 4 of VMS.

The first observation is that the performance for the CONVERT utility is superior to that of the DTR RESTRUCTURE command under both Version 3 and Version 4. This is because CONVERT utilizes block I/O and bypasses some of the layers of RMS. Performance is especially better for disk I/O which is significant on many VAXes which may be I/O bound due to a high load of information management applications.

Interestingly enough, performance comparisons between Version 3 and Version 4 do not show significant improvement in indexed file loading using CONVERT, except in Elapsed Time. In examining the data for Direct I/O, almost no change in performance is observable. This is a condition which will be noted throughout this paper and one in which application developers should be aware while working under Version 4. A change was implemented in Version 4 of VMS to eliminate Ancillary Control Processes (ACPs) for disk operations. They were replaced by the Extended Queue I/O Processor (XQP) which resides in each process' program space. As a result, many of the costs or resources consumed by the ACP under Version 3 now appear under the application's process in Version 4. Therefore, processes which do not appear to be performing better Version 4 may in fact be improved because of the lack of a total accounting of the resources consumed under Version 3.

For example, no attempt was made in the Version 3 tests run in these experiments to account for the resources utilized (CPU Time, Direct I/O) by the ACP in accessing the disk. It is our feeling that a true accounting of these costs would result in a more equitable comparison of Version 3 to Version 4 file performance. However, for the purposes of this paper, it should suffice to note that the condition exists and that the reader is cautioned

to keep in mind that for many tests where Version 4 performance is apparently equal to or only slightly better than Version 3, the results are biased toward Version 3 due to the incomplete accounting.

Because of the relative ease with which CONVERT can be run, we recommend that this be done frequently. File maintenance is important where there are frequent updates to the primary or alternate keys or where many records are added to the file. CONVERT will eliminate bucket splits and improve contiguity (if adequate contiguous disk space is available for the new file). Often CONVERT can be run at night or in batch mode with little or no impact on interactive users.

### TUNING CONSIDERATIONS

The next level of sophistication in optimizing performance is to tune the indexed file. A more detailed discussion of file tuning considerations is available in other papers<sup>2,3</sup> as well as the VAX-11 Guide to File Applications. However, a brief discussion of important file tuning parameters is presented here.

- o Areas

One can separate a file into physically separated areas on a disk or even different disks within the same volume set. The idea is to minimize disk head movement in accessing index and data structures. For larger files, the data levels of a file should be placed in an area on one disk volume and the index level in an area on another disk volume in the same volume set.

- o Initial Allocation

Enough space for the entire file and a reasonable estimate of additional space requirements should be allocated upon initial creation of the file in order to reduce the number of times RMS has to extend the file each time records are added. File extensions into noncontiguous areas reduce efficiency by introducing additional disk head movements each time the application has to access records in the extended area.

- o Contiguity

If your file is contiguous, it will require fewer disk head movements to traverse the file. If separate areas are used for the file or if the disk is not compressed regularly to consolidate noncontiguous space, then the BEST\_TRY\_CONTIGUOUS attribute should be selected, as a minimum.

- o Extension Size

As stated previously, file extensions result in some increase in inefficiency. Where extensions are necessary, however, it is best to extend the file by a reasonable size to minimize the effects.

- o Bucket Size

The file bucket size is one of two parameters for which optimization may have the largest beneficial impact on indexed file performance. In indexed file applications, the bucket size affects the number of index levels. Fewer index levels reduces the number of disk accesses in traversing the index structure. In most cases, if you can eliminate an index level by a small increase in bucket size, then you should use the larger bucket size.

- o Fill Factor

This attribute should be set based on the anticipated number of random insertions in the given application file. Normally, if the file is maintained by frequent use of the CONVERT utility then this factor should be set fairly high (buckets nearly full) to conserve disk space.

- o Number of RMS Buffers

Along with the bucket size, optimization of this parameter can also significantly improve performance of indexed files. In random access applications in files with multiple levels, buffer count is more important. This is because you should try to cache the buckets from similar key levels in memory as there is a chance the successive accesses will refer to the same higher level bucket.

Our initial attempts at tuning the employee database involved increasing the target fill factor to 100 percent and creating a flatter file. The fill factor was increased to 100 percent because new records are always added at the end of the employee database. This is because new hires are assigned employee numbers in increasing integer order. On the first day of work, each new employee receives an employee number one digit higher than the last new hire. Because no records are inserted in the primary index between existing records, it was felt that a fill factor of 100 percent would improve both primary index and data bucket utilization.

The second tuning criteria was to create a flatter file. The term "flatter file" means a file which has fewer index levels. Basically, levels are used because index records are placed in buckets, the same as data records. The last index record in a bucket always has the high key value for the bucket. This last key value is copied to an index record on the next higher level until all of the index records fit into one bucket. This level is then referred to as the Root Level.

In creating flatter files through tuning, you may find a trade-off between CPU time and direct I/O. This is because index structures are scanned sequentially within a given level. In using a file with fewer levels, you may see an increase in CPU time for the additional scanning because the buckets are larger. At the same time, direct I/O may decrease because there will be fewer levels to navigate.

In the first attempt, we increased the bucket size from 2 to 3 blocks which produced a file consisting only of one level. (The original file had two index levels). In addition, the RMS buffer count was changed from the default value of 1 buffer to an optimized value of 4 buffers. The primary key performance results for tuning the JMM employee database are presented in Figures 5-7 for Elapsed Time, CPU Time and Direct I/O, respectively. Alternate key results are presented in Figures 8-10, in the same order for Elapsed Time, CPU Time and Direct I/O. Version 3 and Version 4 results are shown on each graph.

As can be seen from the results, there are improvements in performance by file tuning with respect to CPU Time, Direct I/O and Elapsed Time. In this example, Direct I/O decreased because the application did not have to reference more than one index level even though CPU Time did not increase. In addition, there was improved caching of the index buckets due to the availability of more buffers. While the alternate key improvements are not as high as those for the primary key, they are still impressive gains.

Comparison of the results from Version 3 to Version 4 were interesting. Version 4 showed less Direct I/O and Elapsed Time performance at slightly higher CPU Time. Again, the reader should keep in mind the previous comments regarding the elimination of ACPs in Version 4. These results indicate that tuning guidelines which were valid under Version 3 are likely to be valid under Version 4. Also, the guidelines appear to be valid for both primary and alternate keys.

#### LARGE FILE APPLICATION

In order to test some of the tuning considerations on a larger scale, an application file was built with the following attributes:

- o Fixed length - 10 byte records
- o Primary Key Only
- o 100,000 Records Total
- o Prologue 3

The keys were generated using the FORTRAN RND() function in order to distribute them in some random order. The file was loaded with RMS CONVERT and benchmark procedures similar to the ones developed for the employee database tests were used. For the large file application, two variables were tuned: bucket size and RMS buffer count. Bucket size was varied at 2, 5 and 8 blocks with a RMS buffer count of 3 in Figures 11-13, for Elapsed Time, CPU Time and Direct I/O, respectively under Version 3 and Version 4. RMS buffer count was varied from 1 to 6 buffers with a constant bucket size of 2 blocks in Figures 14 and 15, for CPU Time and Direct I/O, respectively.

In the bucket size experiments, you can see the trade off between CPU Time and Direct I/O that was discussed earlier. As the bucket size was increased, the number of levels decreased. As a

result, the CPU Time increased and the Direct I/O decreased for a given random access. For this particular file, the Direct I/O performance was less sensitive to bucket size than CPU Time, so an optimum bucket size of 2 blocks was selected. Depending on the competition for resources such as CPU Time and I/O bandwidth for your system, you should select the file design which best utilizes those available resources.

The results for RMS buffer count show that CPU Time tends to be relatively insensitive to number of buffers for this particular file, but there is an interesting threshold affect in Direct I/O performance. In moving from 2 to 3 buffers, a dramatic improvement in performance is shown with respect to Direct I/O. One additional buffer, at 4 total, helps performance marginally, although additional buffers beyond 4 do not help at all. It is felt that for this file, 3-4 buffers were sufficient to cache a significant portion of the index structure in memory and after that point, additional buffers were ineffective because it was impossible to cache all of the data. As a result, an RMS buffer count of 4 was selected as the optimum for this parameter.

Figure 16 presents a summary of the comparison between Version 3 and Version 4 performance at the optimum bucket size of 3 blocks and the optimum RMS buffer count of 4. The results show improvements in CPU and Elapsed Time with little change in Direct I/O. However, again it should be pointed out that the Version 3 results are biased because the resources consumed by the ACPs have not been included. Based on this knowledge, it would be reasonable to assume that Direct I/O performance also improved from Version 3 to Version 4. Similarly to the JMM employee database application, the optimum conditions valid for Version 3 are still valid for Version 4.

#### PROLOG 3 FILES

With the advent of Version 3 of VMS, Prolog 3 files were made available which offered some advantages over Prolog 1 and 2 file structures. With Prolog 3 files, the application developer may select compression options for the file primary key or data. The advantages of compression are that the amount of disk space required for the file is reduced, I/O buffers theoretically contain more data (because of the compression, the buffers contain more information; they don't really contain more data) and fewer index levels were required. The principle disadvantage was that more CPU Time was required by the compression algorithm to resolve the lookups.

In Version 4, several extensions were made to the Prolog 3 file structure. Version 3 Prolog 3 files were permitted to have only one key, of string data type. In Version 4, Prolog 3 files allow multiple keys of all data types.

Two sets of experiments were performed to evaluate indexed file performance for Prolog 3 files under Version 4. In the first set of experiments, 1000 records were added to the 100,000 record random number database set up as three separate files. As

shown in Figure 17, these databases were set up as a Prolog 3 file with key compression turned on, Prolog 3 with key compression turned off and Prolog 2. In this experiment, the Prolog 3 files performed better than Prolog 2 files in Elapsed and CPU Time, and marginally better in Direct I/O. In the case of the second set of experiments, shown in Figure 18, a random access search for 1000 records were performed on the same three files. In this case, the Prolog 3 file without key compression clearly out performed the other two files.

Unfortunately, it is felt that these results are not conclusive because of the small record size in this file and the relatively limited opportunity for key and data compression. Future test results will present data and conclusions on Prolog 2 and 3 indexed files with larger record sizes and bigger keys.

### RMS RUN-TIME OPTIONS

Among the new features implemented in Version 4 are the RMS Run-time Options which allow the application developer access to the CONNECT options of the RMS record access block (RAB). These are now available through the File Definition Language which obviates the need to program in MACRO-32 or otherwise get direct access to the RAB. These options offer considerable flexibility in file performance tuning by allowing the developer to tailor parameters for specific applications. In the past, several of these options either were not available, were difficult to access or could only be tuned on a process by process basis.

Of particular interest to developers interested in performance tuning are:

- o Asynchronous record processing

This option allows the program to continue processing without waiting for I/O completion. The I/O request proceeds in parallel with the next program steps.

- o Deferred write

The file buffer is written only when needed by another application sharing the file or when the file is closed. Obviously, selecting this option may have an impact on file integrity in the event of a system crash if buffers have not been written, but the potential for performance gains is significant.

- o Multibuffer count

The multibuffer count may now be specified on a file by file basis for indexed files. In Version 3, this was specified as a system default or on a process level. It is now possible to tailor the buffer requirements to the optimum needs of each file without directly accessing the RAB.

- o Read-ahead/Write-behind (sequential)

While these do not apply to indexed files, they are mentioned anyways. Read-ahead and write-behind allow RMS to alternate between buffers on sequential read and write operations to improve performance.

- o Record retrieval options

Several record retrieval options are now available including "do not lock record," "lock read/write," "time out period," and "wait if locked." These allow the application developer to tailor lock management operations for files with extensive sharing of data among multiple users.

It should be mentioned that the advent of VAX clusters has complicated the process of managing record locks. The application developer should recognize that there are additional performance considerations when dealing with shared access, particularly shared write access to indexed files among cluster nodes. For example, the first node to open a file in a shared mode is designated the lock-mastering node. Locks requested applications running on the mastering node incur less cost than locks from lock-requesting nodes, but it should be remembered that the lock-mastering node is still required to process all of the other requests.

Single node file sharing requires no distributed locking and therefore incurs the same cost as if the file is opened on a non-clustered node. However, the application developer should be careful to avoid any unnecessary shared write access to indexed files because of the additional locking overhead cost of locking individual records instead of the whole file. This is valid even for shared files on a single node. A good rule of thumb is to open files with the minimum access and minimum sharing necessary to do the desired task.

While it was not possible to present the results from experiments performed on the RMS Run-time Options in time for this paper, limited test data obtained on a VAX 8600 on the deferred write option for indexed files showed a minor improvement in CPU Time and a whopping improvement by a factor of 60 or better in Direct I/O.

Application developers working with the RMS Run-time Options should be aware that many VAX languages or layered products accessing RMS files may treat options in unpredictable ways. For example, Datatrieve does not permit read-ahead and write-behind in any fashion on sequential files. Also, Datatrieve does not permit deferred write when the file is readied in a SHARED mode or when the file is designed with a 100 percent fill factor. Each layered product may respond differently and it is advised that application developers check each product's documentation for specific details.

### BIBLIOGRAPHY

1. Glaser, H.T., "Improving Performance of RMS ISAM Files," Wombat Examiner, Datatrieve SIG Newsletter, Volume 5, Number 3, January 1984.

2. Glaser, H.T. and P.A. Naecker, "Optimization of Indexed File Performance in the Data Management Environment," Proceedings of the Digital Equipment Computer Users Society, U.S.A. Spring 1984, Cincinnati, Ohio, June 1984.
3. Glaser, H.T. and P.A. Naecker, "Optimization of Indexed File Performance in the Data Management Environment," Proceedings of the Digital Equipment Computer Users Society, U.S.A. Fall 1984, Anaheim, California, December 1984.

# Random Access Procedure

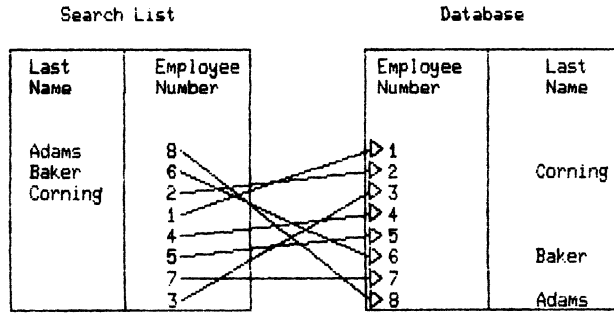


FIGURE 1

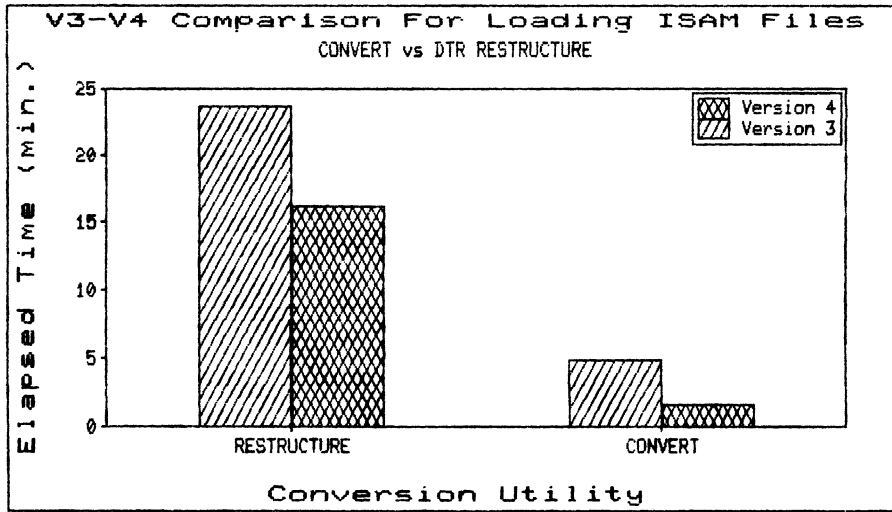


FIGURE 2

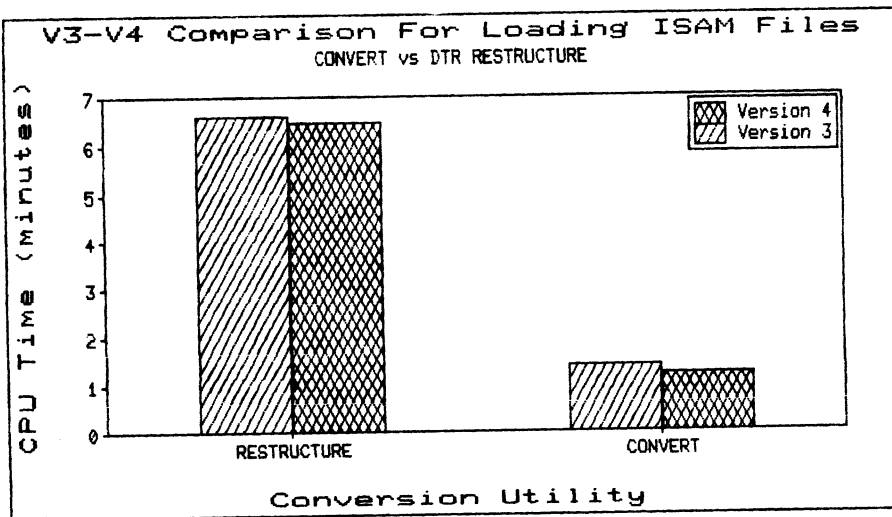


FIGURE 3

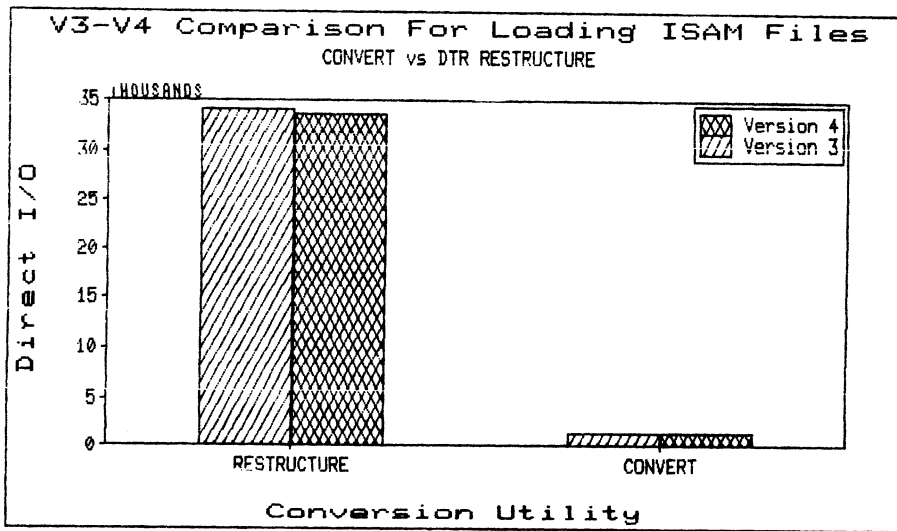


FIGURE 4

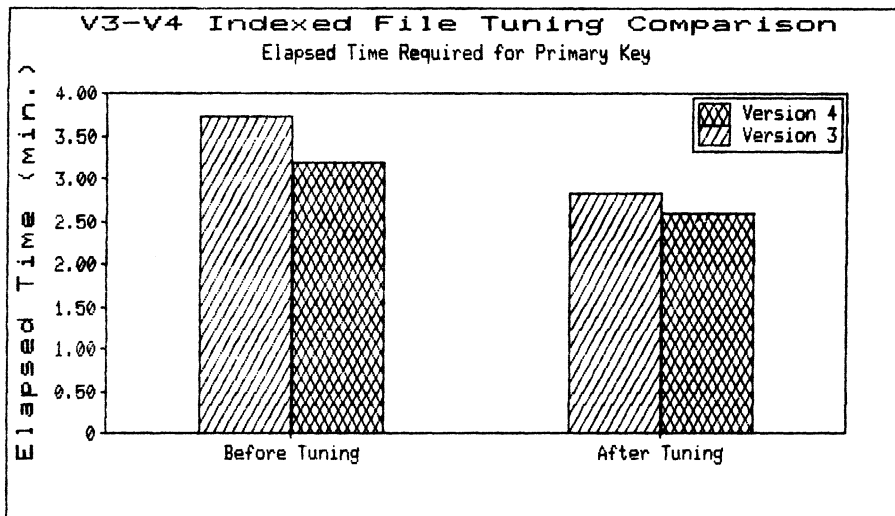


FIGURE 5

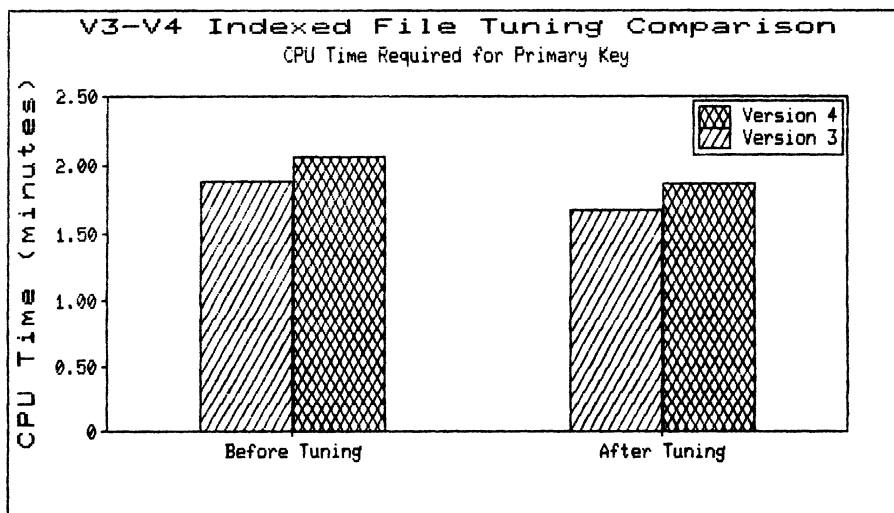


FIGURE 6



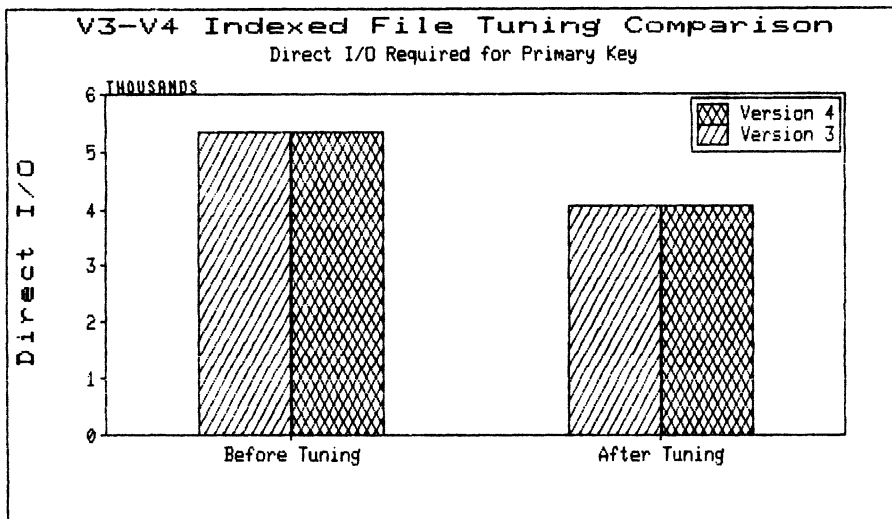


FIGURE 7

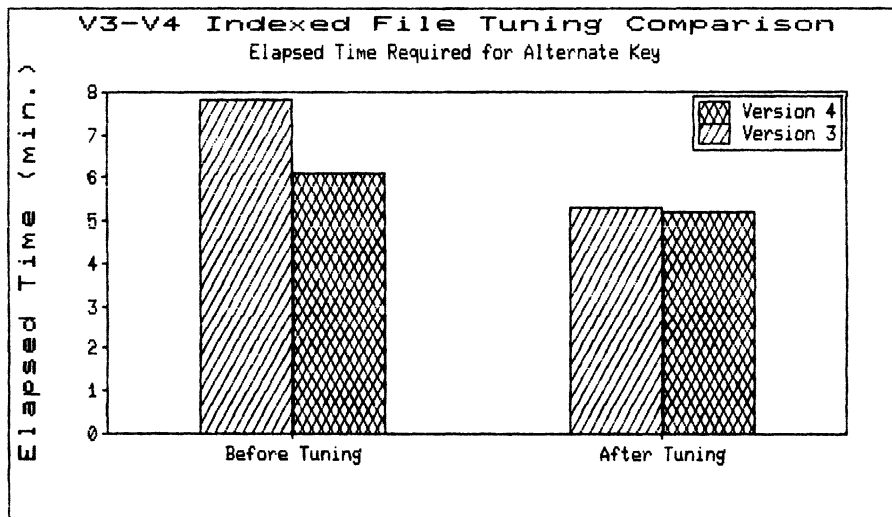


FIGURE 8

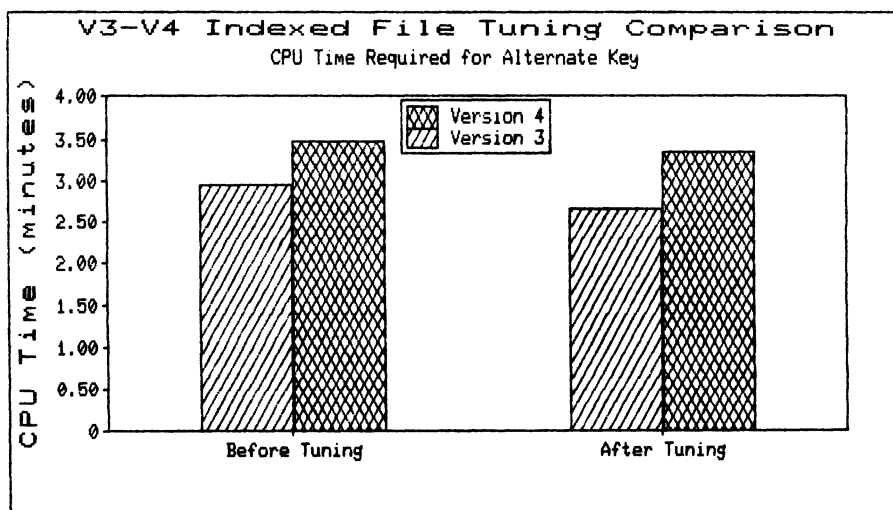


FIGURE 9

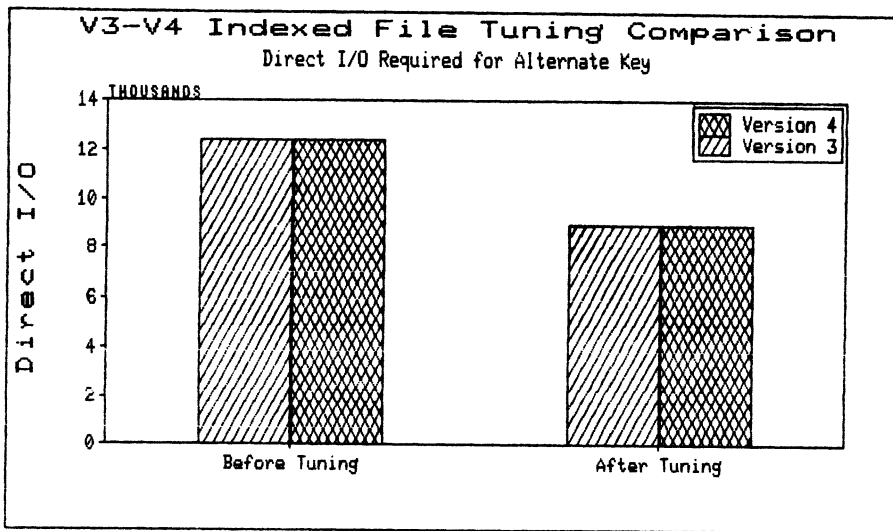


FIGURE 10

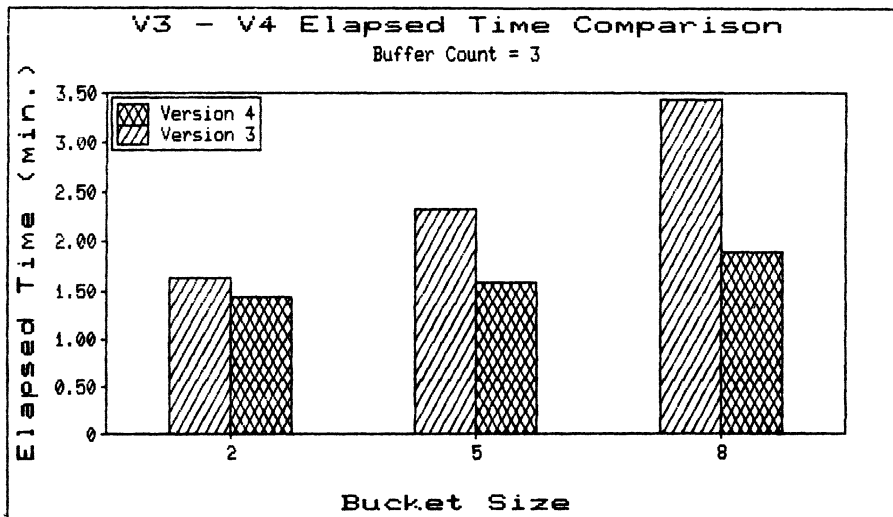


FIGURE 11

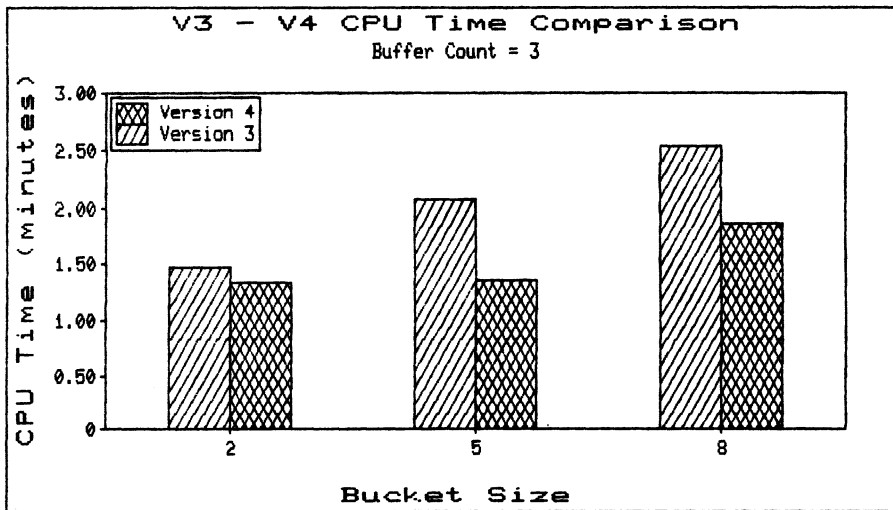


FIGURE 12

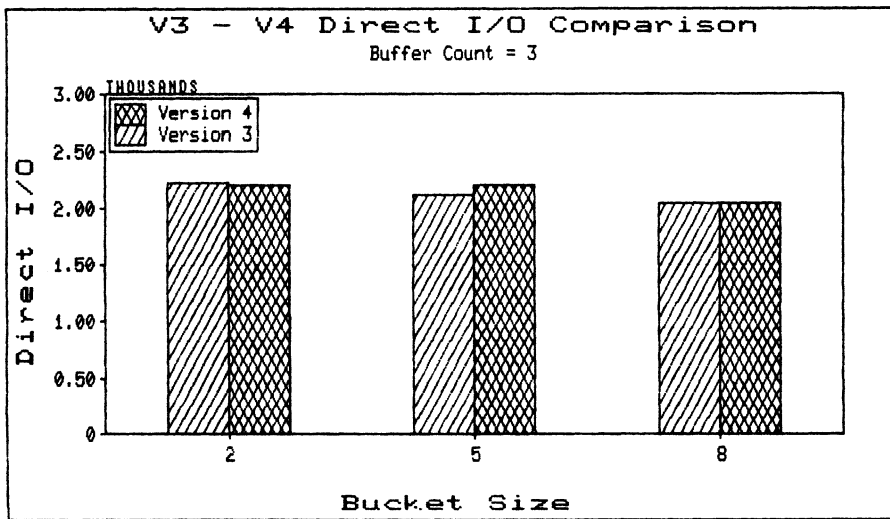


FIGURE 13

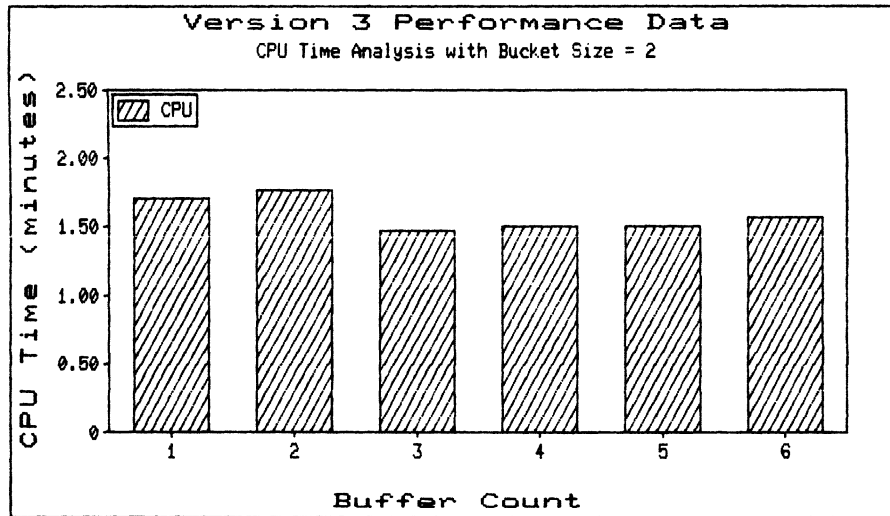


FIGURE 14

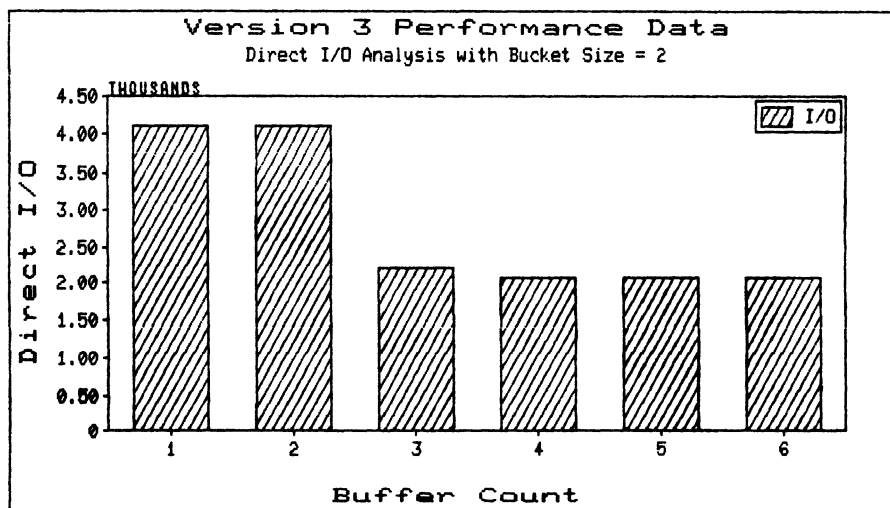


FIGURE 15

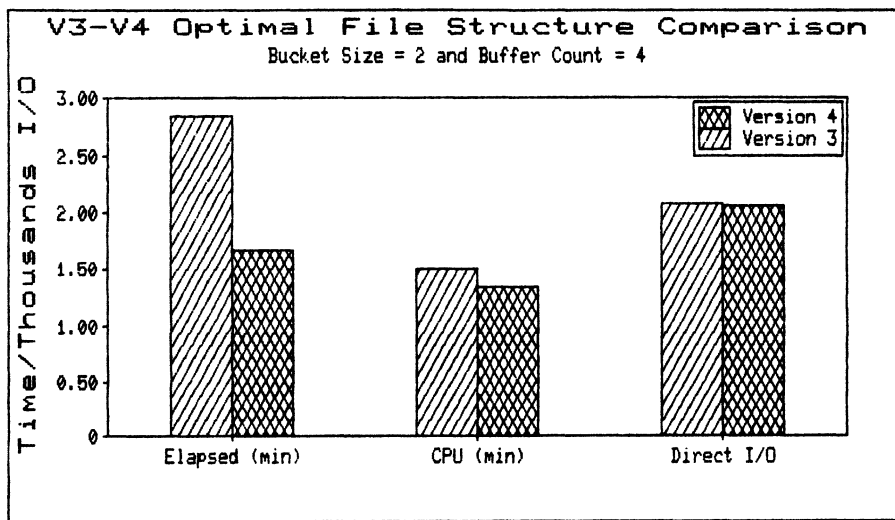


FIGURE 16

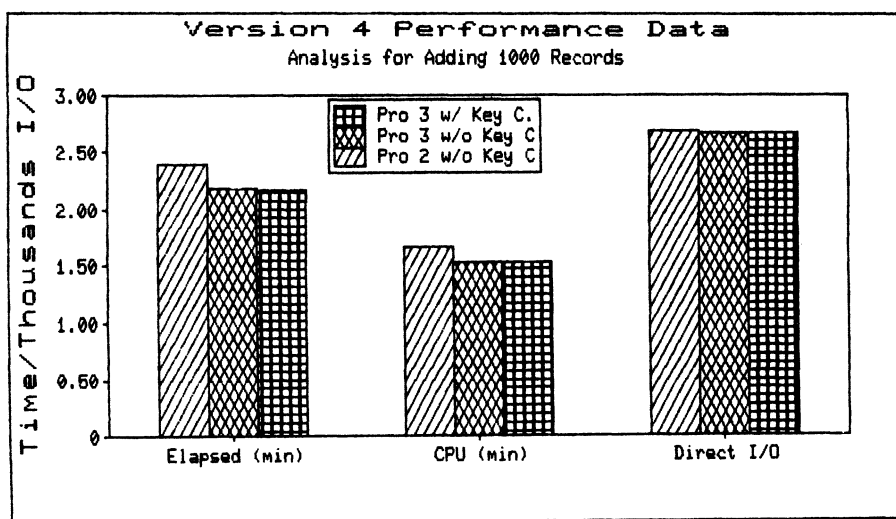


FIGURE 17

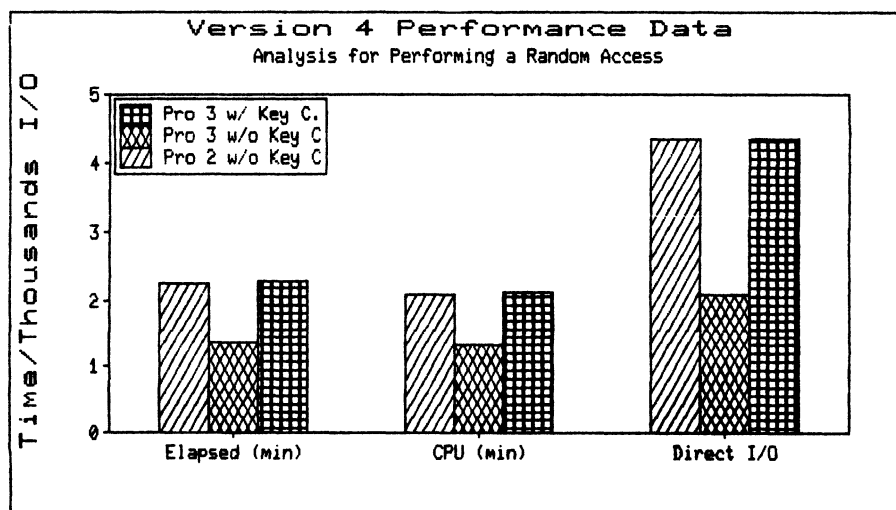


FIGURE 18



MACVAX CONNECTION

Bob Wilson  
General Electric Company  
Lanham, MD, 20706

ABSTRACT

Bringing the Macintosh to the VAX world makes the best of both better.

General Electric developed a VAX emulator for the Macintosh Imagewriter printer, MACVAX. This emulator allows the VAX to output Macintosh graphics on LXY-11s, REGIS, and VAXSTATION I devices. As a result, a VAXSTATION I based prototype was created in a fraction of the time that traditional methods would have taken.

Recently, we needed a menu driven operator interface for an automatic test equipment (ATE)

system. But a common problem with text menus has been understanding technical instructions (i.e., "jargon"). However, by using Macintosh generated artwork, the operator can see exactly what is needed to perform an unfamiliar task.

To see the contrast, read the instructions in Figure 1 while hiding the hardware sketch with your hand. Removing your hand reveals details that are difficult to describe in English.

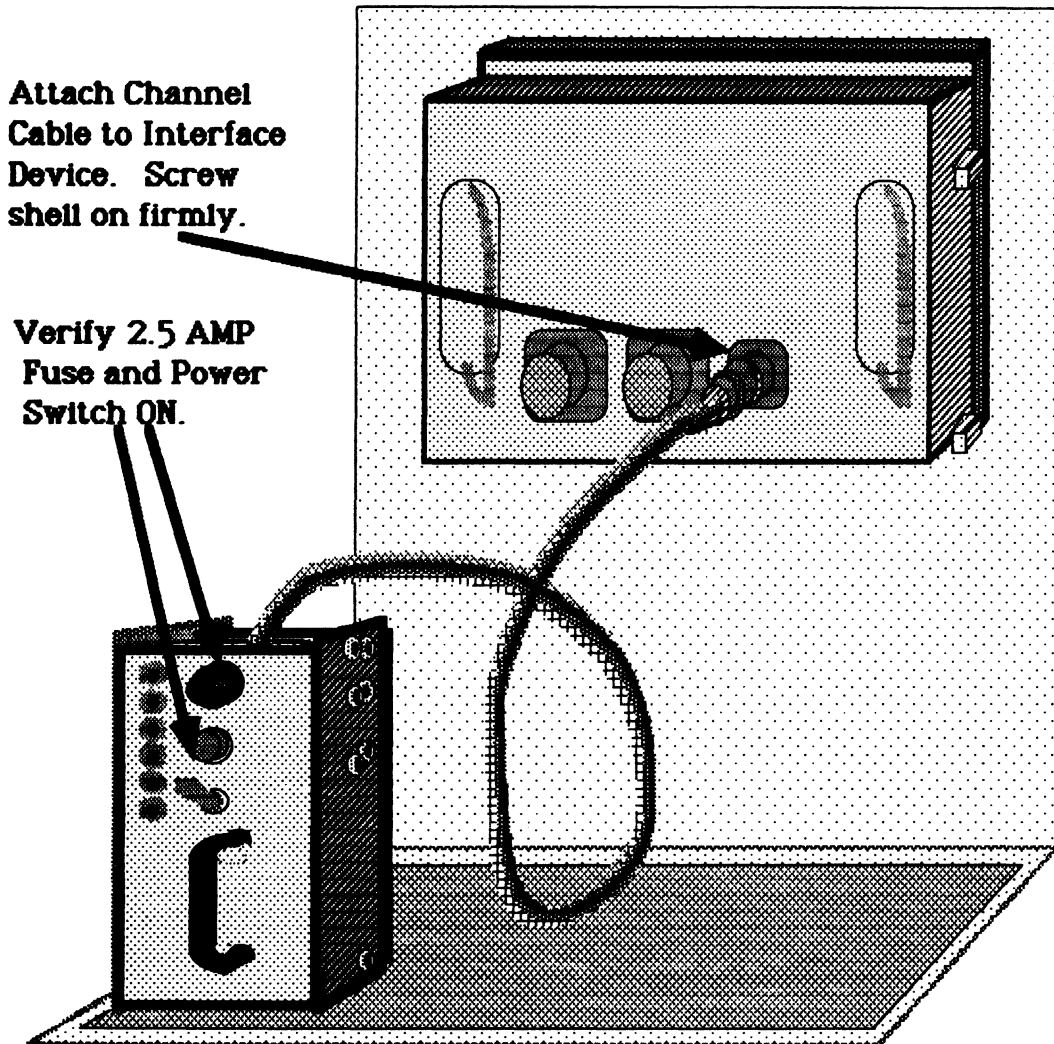


Figure 1

The ability to rapidly make new menu screens allowed us to concentrate on WHAT should be in the menu rather than HOW to make a screen. As a result, we could experiment with menu structure to improve operator performance.

The original menu, a classical hierarchical tree (see Figure 2), took only forty lines of DCL to implement. But it was too tedious because the operator had to pass through the branch nodes (blank boxes) to go between information screens (filled boxes). Adding a horizontal short-cut made it faster but still required visiting branch nodes. The last version resembles a procedural language and consists of just the information nodes.

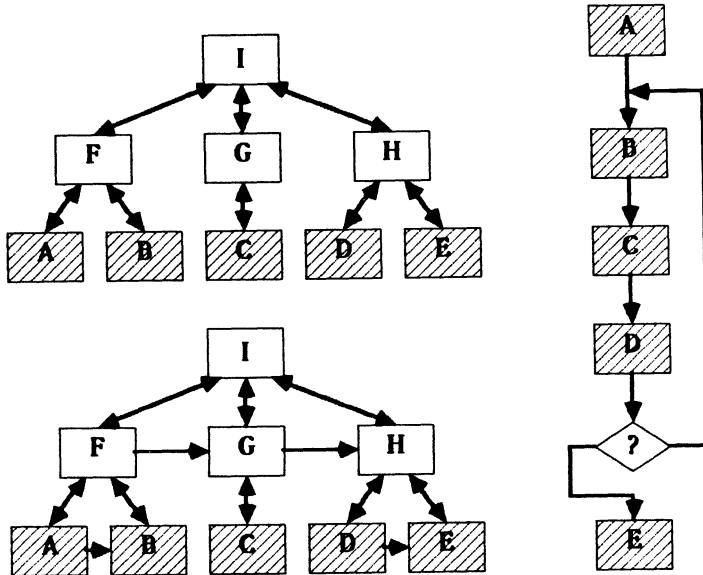


Figure 2

Making an emulator requires solving problems in physical, electrical and logical connections. Unlike VAX RS-232 terminal ports, the Macintosh uses a 9-pin, D connector feeding a sub-set of RS-422 (see Figure 3). In spite of TTL voltage levels and differential signals, my fifty foot cable has had no problems with either DZ-11 or Emulex terminal ports at 9600 baud.

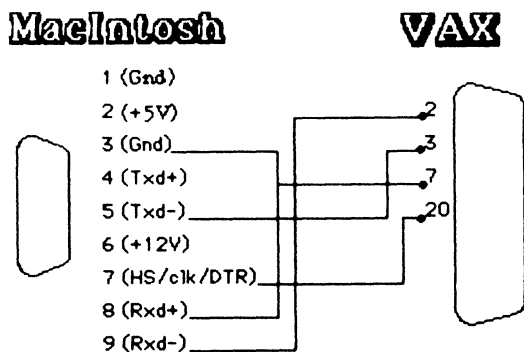


Figure 3

To read the printer data, the emulator sets Data Terminal Ready (DTR) which the Macintosh uses as printer ready. The Macintosh recognizes both DTR and XON/XOFF as flow control to prevent over-running the printer. Fortunately, VMS also uses XON/XOFF to avoid type-ahead buffer overflow.

The Macintosh sends ASCII data, control characters and escape sequences to the Imagewriter printer in either draft, standard, or high resolution mode. In draft mode, the Imagewriter ROMs translate ASCII into dot matrix characters like most dot matrix printers. Standard mode is 72-bits per inch, one pass, bit-mapped. High resolution mode sends bit-mapped, interleaved passes to "diddle" the bits and reduce the jagged edges of sloping lines. To simplify the software, the emulator handles only standard mode.

The Imagewriter graphics code sends 8-bit bytes to fire the print head pins (see Figure 4). Since any ASCII value can appear, you should not route the printer data through terminal networks that respond to special ASCII control characters.

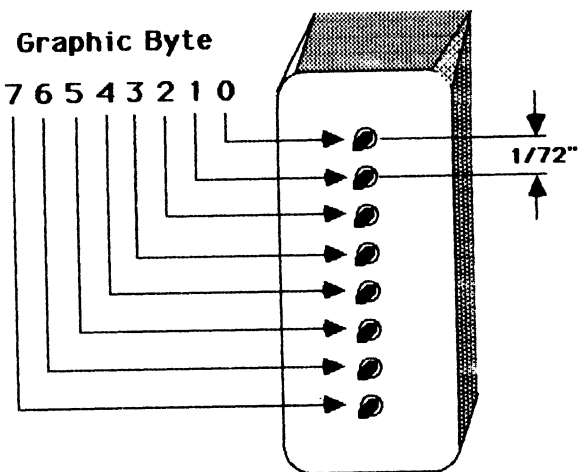


Figure 4

We implemented the emulator as a VAX foreign command with the following syntax:

```
$ MACPRINT input output /qualifiers
```

The "input" is either a terminal or the name of a file typed "\*.MCI". The "output" is either a VAXSTATION I device or the name of a file whose type is determined by one of four qualifiers:

```
/IMAGEWRITER - Record aligned escape sequences (*.MCI)
```

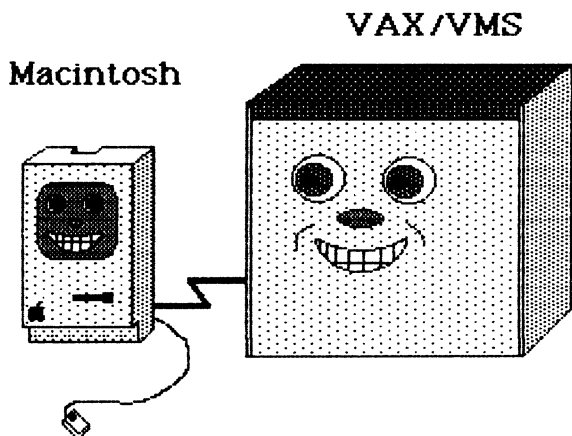
```
/LXY11 - Printronix raster lines (*.LXY)
```

/VT125 - Raster lines translated to REGIS (\*.REG)

/VS - VAXSTATION I bit-map (VCA0:)

/IMAGewriter output generates a file type "\*.MCI" which you can print on an Imagewriter. Thus one spooled Imagewriter could service multiple Macintoshes. Also, you can use the "\*.MCI" file to generate the other output formats. Furthermore, the file aligns the escape sequences on record boundaries to simplify writing translation programs for other graphics devices (i.e., Versatec, LA-50).

/LXY11 uses Printronix 6-bit, horizontal bit-map format. Compared to the Imagewriter, printing is almost three times faster and of better quality (see Figure 5). In the benchmark, the transfer time is the period the Macintosh is tied up sending the data to the printer.



**Benchmark Time**

|          | Imagewriter | VAX     |
|----------|-------------|---------|
| Transfer | 80 sec.     | 35 sec. |
| Printing | 95 sec.     | 15 sec. |

Figure 5

/VT125 converts the raster lines into an "\*.REG" file consisting of REGIS line and dot commands. The escape sequences to set the device into graphics mode are not part of the file since different REGIS devices may need different startups. To display the output on a VT-125, I enter, "\$ TYPE START.REG,data.REG". Take care since this translation often generates a file ten times larger than the original.

/VS must be used with "VCA0:" to go directly to the VAXSTATION display memory. Because the VAXSTATION screen has more pixels than a standard output page, /NORESET, /XADD and /YADD qualifiers allow building a complex screen from multiple Macintosh images. Finally, a single

MacDraw file of two legal size pages can completely cover the screen by using /DUAL.

Standard Macintosh pixels are square at 72 bits per inch. Many of the DEC devices have horizontal, rectangular pixels closer to 60 bits per inch causing the images to appear fatter (i.e., circles are squished). To compensate, just draw the images a little skinnier.

The emulator does not support direct text transfers from the Macintosh to the VAX. To up-load text, I normally use MacTerminal to insert text into a VAX file. Other alternatives include KERMIT and XMODEM protocol systems which can be found on user group disks and the DECUS tapes. In particular, MACX found in PCS-51 of CompuServe works well with MacTerminal's XMODEM.

The Macintosh does a fine job of emulating a VT-100 (see Figure 6) with MacTerminal. The smaller foot-print saves desk space and I like the Selectric style keyboard over the new "ergomatic" style. Sad to say but at 9600 baud, the internal buffering leads to a ten to twelve line delay before XOFF halts the screen. Furthermore, DELETE and the ARROW keys require two finger operation (i.e., CONTROL and a second key).

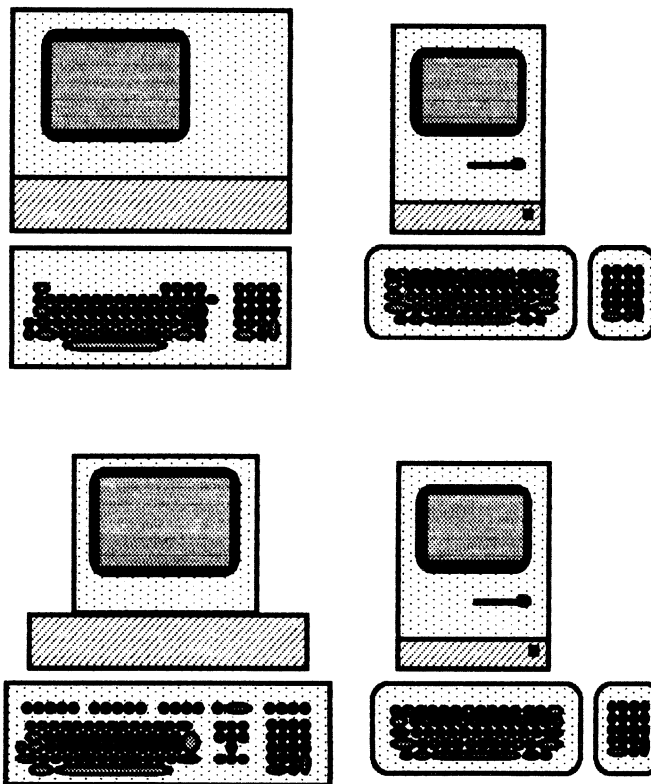


Figure 6

I have received two types of complaints about the Macintosh screen. The most common is the



small screen size (see Figure 7). I don't find this to be a problem since the active area is the same width as a line of text on standard typewriter paper with margins. The second problem which the VAXSTATION I shares, is the sensitivity some people have to the black on white display.

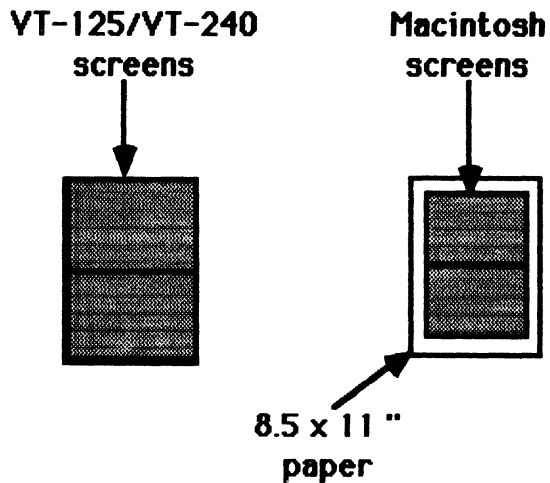


Figure 7

Traveling frequently, the Macintosh passes as carry on luggage in a large nylon bag. On all except early 727s and DC-9s, it fits in the overhead storage. Though weighing less than twenty-five pounds, the shoulder strap helps to carry the weight when hiking to a connecting flight.

The Macintosh is one of the best, a PLC (Proper Little Computer) with excellent graphics editors. Likewise, the VAX is a superb, multi-user, development system. Combining the two makes a symbiotic relationship where the VAX world gains easily generated artwork graphics and the Macintosh world gains high speed, high quality, output devices. Together, they accomplish more than the simple sum of their parts.

#### BIBLIOGRAPHY

1. Imagewriter User's Manual, 1983, Apple Computer, Inc.
2. "The Imagewriter and Beyond", Daniel Farber and Adrian Mello, MACWORLD, September/October 1984, pp 76-83, Volume 1, Number 4.
3. The Apple Macintosh Book, Cary Lu, 1984, Microsoft Press

## VAX/VMS Security Considerations

Robert R. Wells  
Technology Development Corporation  
621 Six Flags Drive  
Arlington, Texas 76011  
(817) 461-1242

### ABSTRACT

Some level of security is necessary on almost all computer systems: Whether it's just a microcomputer sitting around the office or the biggest mainframe deep in the bowels of those super secret research centers; security is important. This paper addresses some general security issues applicable to many brands of computers with some specific comments aimed at those sites using the DEC VAX.

### General Background

As demonstrated quite dramatically by several recently well publicized news stories, there are indeed individuals taking advantage of other people's computers. The debate rages on whether these individuals are looking for some sort of emotional challenge, something to vandalize, or simply desire some extra computer time. Regardless of the reason, if someone has access to some of your computer resources, that shouldn't, you could be asking for it. An important thing to be learned is knowing your defenses. In the following tale of woe, recognize that the most effective defense mechanism is rendered useless, essentially breached by the person most responsible for security, the owner of the account.

You're logged-in to the system manager's account, it's 3pm, you're thirsty: A soda would sure taste great. You get up, leaving your office in search of the closest vending machine, hoping that it won't rob you of 50 cents this time. It's only a 5 minute walk and you don't think twice about leaving your terminal logged on for such a short while. In the meantime, somebody walks into your office, notices the vacant terminal and decides to "explore." The visitor intends to give the command:

```
DIR *.*;*
```

Oops, instead the fingers somehow remember another neural pattern and type:

```
DEL *.*;*
```

It doesn't matter if he stays around to tell you about the accident or attempts to cover things up and run (although the latter may avoid hazardous blows to facial areas), the damage has been done. Time to find the backup tapes. (a small note, be sure employees are reminded to log-out during company fire-drills)

Security begins with educating your user community; many don't realize the potential damage that might be caused if someone acquires access to the

corporate computers. Most companies have sensitive data on their systems that if divulged beyond their premises could have some serious consequences upon their ability to do business. Make your users stop and think about what kind of data lives in their--no, the company's--files. Point out some specifics; for instance: the accounting department's pay rates and salaries, the legal department's text of upcoming contract proposals, the management staff's company portfolio, the personnel department's employee records, the engineering department's classified material, etc.

Managers need to be kept aware of all this too. Get their support.

If possible, develop a formal company-wide policy concerning who has access to the computers, what level of access, consequences for being negligent, etc. BUT! Be sure that everyone knows that such a document exists. (When you develop one of these, or if you already have one, the author would greatly appreciate seeing it.)

We've mentioned a need for security, but what are we really trying to be secure from? DEC identifies two categories of illicit computer activity:

- o Probing
- o Penetrations

Probing occurs when you, as a system manager, haven't made use of the available security mechanisms and a user actively exploits parts of the operating system that aren't adequately protected. Users that engage in probing may or may not be knowledgeable about the computer system, they may simply be seeking a "challenge" or "just casually browsing" as a guilty user once said. This kind of activity, though less serious than a penetration may grow into a bigger problem. Don't downplay its significance. Probing generally starts with a user trying to see whose directories he can see, what mail he can read, or possibly what accounting data he can access. Systems on which considerable probing exists indicate a lack of security commitment or training on the part of the users or a lack of enforcement from the system manager.

Penetration is a successful circumvention of all existing security controls and usually indicates a more sophisticated user. The potential for damage is extremely high if the person is malicious. In this situation, the blame falls more to the software and the security mechanisms than to any system manager failure.

Who will probe, penetrate?

- o Software analysts
- o Programmers
- o Managers
- o Engineers
- o Data entry personnel

And don't forget.

- o Former users and employees
- o Competition
- o Hackers

### Gaining Access

There are 2 broad categories of users on a VAX, each has differing methods of access:

1. Local
2. Remote

Under the category of "local" we'll include those users gaining access via terminals that are directly wired to the VAX and those dialing up through a modem. For those sites that must deal with port-selectors, terminal servers, and whatever other complicating devices, I'll cover that subject as somewhat of a combination of direct wire and modem. "Remote" users are those that are making use of DEC's networking software, DECnet. Remote users and networks will be covered in a later section.

Whether one simply walks up to a terminal, or dials-in via a modem, the first and most obvious aspect of VMS security is the username and password prompts we see at login time. Not so apparent is the fact that this is the strongest and most critical deterrent to unauthorized computer usage.

At most sites, users wanting access to a VAX, hit return on a terminal and are greeted with something similar to:

```
Froboz Magic Robot Company
VAX/VMS Version V3.7
```

Username:

This kind of welcome has some good points and some bad, but mostly bad. If it's only 2 lines long, how did we already get off to a bad start? Glad you asked. Depending upon what level of security your site is aiming for, just this simple message may be giving away too much information.

For your authorized users, the announcement message is not so important or enlightening, they most likely already know the company name, and if they

wanted could easily find out what operating system is being used and its version. Your users should be bound by some kind of moral or ethical conduct as employees or clients; they should not want to create problems for their company and thereby to themselves. If this is not the case, you do have a strong computer usage policy in place to discourage indiscriminate computer conduct or "exploring", don't you?

More to the problem is what extra information you're providing to those coming in through your modems who might or might not be employees or authorized users. In this age of cheaper modems and wider communications, such a message probably tells more about your system than you think and to more people than you really want to know. (Have you seen the latest modem sales statistics?) Telephone lines provide a convenience for employees or clients to dial-in to your computer system; unfortunately, in addition to being a convenience, these same telephone lines provide ample anonymity for those spreading mischief.

Back to the 2 line announcement, here are some things a potential outsider (dialing-in by modem) might surmise from the message and be thinking while developing a plan of "attack" on your machine.

- o I've reached a working computer
- o It's a VAX (oh boy!)
- o It's running VMS (Good, I know some VMS tricks).
- o It's at VMS V3.7 (fewer security mechanisms than V4.x)
- o Froboz is just down the street, I'll go rummage through their trash looking for computer printouts and passwords.

Probably this is more than you want to give out to just anyone. If you want to welcome your users to the machine, save it until after they've proved they're worthy by letting them supply a valid username and password.

### Usernames

On to the next line of the display, the "Username:" prompt. Much like a password, a username is required as part of the login sequence; the difference being a username is less sensitive, but still company-private. An obvious point but worth repeating, the username works in conjunction with the password. Without the answer to that first question that VMS asks in the login sequence, no password will do any good.

Unlike passwords, it's generally not practical (or desirable) to keep usernames a secret from those that you work with; they have a genuine need to know, after all that's what makes MAIL and PHONE so nice. You might, however, consider not using employee last names as the username: After all, is there a company in the world that doesn't have a SMITH, JONES, or WILSON? As a suggestion, some sites concatenate the last name with the first 2 or 3 characters of the first name, so that employee

"LARRY JONES" has the user name JONESLA. This has the added advantage of resolving the standard problem of multiple users named JONES. Unless you already have a better directory naming scheme, use the concatenated name for the directory as well.

### Passwords

It would be easy to write an entire paper just on the proper selection and management of password. Here are some guidelines we try to follow:

- o Passwords MUST be a minimum of 6 and preferably 8 characters in length.

Although VMS provides A-Z, 0-9, \_ and \$ as candidate password characters, I suggest restricting passwords to only the "alpha" (A-Z) symbols, because for most typists, the finger reach to the number and special symbol keys is quite distinct, awkward, and slow. Unless they're fast, you can easily watch the hands of most people typing numbers: If their password is only 8 characters long and you see them type the first 4 as numbers you might conceivably "guess" what the remaining symbols are regardless of composition. VMS V4.x allows you to enforce a minimum password length on a user by user basis, set it! In VMS 3.x the same can be accomplished with various patches available from various sources (the usual caveat about messing with DEC's code though). If your users squawk, ask them for a reasonable explanation why they should be allowed to jeopardize the system!

- o Don't select passwords associated with the user.

This means no husband's or wife's name, no children's or pet's names, no hobbies (like STAMPS, COINS), no phone numbers, no department names (like ENGINEERING), etc. Of course, no initials.

- o Don't allow any word that might be found in a dictionary (the Webster's variety) as a password.

The dictionaries that come with most spell checking software can be readily adapted to throwing words at a password prompt until a valid one is found. The advent of these programs on microcomputers heightens the importance of this guideline.

- o Do use nonsense phrases for passwords.

We often suggest such things as FORCEBEWITHTYOU, TENFOUR, BORN2BEFREE, ITSMILLERTIME, well you get the point. Impress upon your users that this is their chance to be as profound (or profane) as they wish.

- o Don't use the same password on multi-vendor hardware.

Just because VMS has an adequate password hashing algorithm, don't assume that the vendor of XYZ computer does too. That manufacturer may simply store the password as plain-text in a file. Use different passwords on different machines.

- o Do change passwords regularly.

How often should passwords be changed? You might just as well ask "How long is a piece of string?" The nature of the data in a particular account usually governs the frequency of change. The more confidential the data, the more often the password should be changed. The same applies to those accounts with more than the standard privileges. For privileged accounts, consider changing the password every month; otherwise, for non-privileged accounts every 3 or 4 months. This too can be enforced under VMS V4.x.

- o Don't use company wide default passwords for new accounts.

When you create new accounts for users, you presume they're going to use them. Not always so! Sometimes these new users may not log-in for months, leaving the account with nothing more than a public password. Along the same lines, don't make the new password a predictable format like JONES123, etc. Pick something random.

- o Don't include passwords in text or command files.

I can almost guarantee that at some point, somehow, that file will get printed and exposed to everyone. (Murphy's law supports me on this.) An old hacker's trick is to scan all command files on a disk for the occurrence of the symbols "::" indicating an explicit DECNET access control string that follows an account password.

- o Do change the passwords to the standard accounts: FIELD, SYSTEM, SYSTEST.

New VMS systems have preset passwords for these accounts. Since these accounts are present on almost every VMS system in the world they are likely candidates for people trying to break into your system. If you have a VMS source license, change the password to that standard account as well.

Don't let your field service engineers set their own password to the FIELD account. They're only human (believe it or not) and if given the chance they would probably select the same password for every machine at every site in their service area.

- o Do explain to users that VMSCAI, EDTCAI, etc DO NOT require their VMS log-in password.

New users are easily intimidated. If the software asks for a secret word or password, you guessed it, 9 out of 10 times they'll enter their VMS log-in password.

Just in case your users have lulled you into believing they know how to select a password and don't need your recommendations or guidelines, I present to you:

Over the last couple years, I've "analyzed" the authorization files at several VAX sites and

determined that, on the average, 40% of all user passwords could be found in a large spelling dictionary or by searching through all possible 3 character strings (the success range was 27% to 74%). To emphasize this point even further, remember it only takes one password on one machine for someone to "break-in." You should recognize this as the weakest link theory. Incidentally, I've never failed to "crack" at least one privileged account.

#### Dial-ins

Dial-up lines are one of the soft-spots in the VAX login process. Not only is there the problem of keeping unwanted outsiders from logging-in, but what about making sure your authorized users have logged-out. Consider what might happen in the following scenario:

It's 2 am, Sandy the 3rd-shift computer operator is having problems with the nightly backup and would like you to dial-in to give a hand (operators always call at 2 am). You grumble, get out of bed and dial-in to your machine. You solve the problem and just before you log-out, you get a phone glitch. Your modem throws garbage on the screen and hangs up. Fine, you were done anyway and go back to sleep. Across town, a hacker has a microcomputer with an auto-dial modem sequentially going through all the phone numbers in your exchange, (555-1234, 555-1235, etc). The modem gets to 555-2341 and viola it displays the message "CONNECT, FOUND ONE" on the terminal. Now when the return key is hit there's no Froboz announcement message, but instead:

\$

Wow! The persistent hacker says. No username to guess and no password to break. (again) OH BOY!

The story could take several turns here, that's not important. Realize that the person now has access to your privileged account. Why? Because the modem on the VAX side did not hang up when your modem did.

Modems should always be configured (on the computer end) to hangup when they detect a loss of carrier. On a VAX the minimal configuration of a modem port should include:

```
SET TERM Txxx /MODEM/HANGUP/PERM
```

You may of course need to add other site-dependant qualifiers (/AUTOBAUD, /DISCONNECT, etc).

In addition to telling VMS about the modem port configuration, don't forget or omit reviewing the modem hardware configuration. Read the documentation from the modem manufacturer, the modem itself may require some DIP switch or jumper changes. Pay particular attention how the modem controls the RS232 signal "data carrier detect."

Depending upon what kind of terminal I/O card the modem is connected to on the VAX (DZ, DMF, DMZ, etc), there may be other problems too; for example,

DMF ports 2-7 (zero based) do not support modem "hangup" control signals, DON'T CONNECT MODEMS TO THESE PORTS (get the message?).

Those sites with port-selectors, etc, this discussion of modems is very important to you because you tend to have all your terminals routed through these gizmos. These devices, almost universally, appear to the VAX as fast modems, meaning they require the same kind of RS232 control signals. As a bonus, or finally to resolve some long standing confusion, VMS V4.x allows port-selectors, terminal servers, etc to be differentiated from modems with a "/DIALUP" qualifier applied to the latter on a "SET TERMINAL" command. Alternately, prior to V4.x, the work-around was to check the terminal speed, 1200 baud and below indicated a real modem and not a port-selector.

Just like passwords, the telephone numbers to your dial-in lines should be protected, handed out only on a request and need-to-know basis. One particular site, through some mis-communication or accident, had their dial-in telephone numbers published in the local phone directory. (For some strange reason log-fail numbers over those ports shot up quickly.) This isn't advised. If practical, consider changing the telephone number to all of your dial-in lines at least once a year, remember the bill the phone company will send you for this will be small compared to the possible consequences.

Keep a close eye on your modems, especially the newer models with n number storage and auto-redial of last number. They often, for the simple command of simply hitting return, will show you the last number dialed (DEC's DF112 for example). If you've just used VAXNET, VMODEM, KERMIT or some other terminal emulator to dial-out of your VAX into another computer, many modems will retain that number for whomever comes along later. Worse yet are those modems that will for another simple command will list 15 or so telephone numbers stored in an internal table. Before you purchase modems, check around, there are some that provide the same function but hide the phone list requiring the user to enter, instead, a number from 1-15 corresponding to a telephone number in the list. (The users never see the list, they're just told that AERO-VAX is number 3 or DELTA-PDP is number 11.) These modems also don't display the last number dialed. If you already have a big investment into a certain brand of modem, check with the manufacture, often there is a simple and low cost fix (new PROMS for instance).

One of the big security features not all sites take advantage of are the access hours/days. AUTHORIZE will allow the setting of primary and secondary hours of the day and days of the week during which a user may log-in. If certain users don't need access after 5 pm or before 8 am or they need the machine only during the week, then use this feature. All the times and days can be set on a user by user basis and can be overridden with the "SET DAY" command at any time.

Along the same idea, if another group of accounts never log-in over the dial-in lines, then note that too with AUTHORIZE. By reducing the number of accounts that can dial-in, you've limited the number of security soft-spots.

#### While logged on

UIC based protection has been around for a long time. Now, in addition, VMS V4.x has many, many new enhancements in this area (ACLs won't be covered here). I won't go into details except to point out that:

- o System directories do not require world read access.

Protect the SYSEXE, SYSLIB, SYSMAINT, SYSMGR, SYSTEST, SYSHLP, SYSUPD so that they only have world execute access (W:E). This prevents users from using wildcard file specifications (for instance they won't be allowed to issue a "DIR SYS\$SYSTEM" command).

- o User's don't need to be able to run system manager utilities (AUTHORIZE, SYSGEN). Remove world execute privilege from these images.
- o The 000000 directory should be protected the same as system directories or better yet, remove world access altogether. This helps eliminate some of the probing.
- o ACLs can be placed on certain items, like files, and will describe all sorts of attempted operations on those files by users.

This is a really good way to monitor probing. If your system directories are protected adequately and you setup or install ACLs properly, any action on files within those directories can be logged. For instance, if a user repeatedly tries to DUMP or COPY the SYSUAF.DAT file, you should suspect that something is wrong (plus you have the proof right in the operator's log).

- o Devices can and should have stringent protections too.

In your system startup command file, protect all of your terminals with a command like "SET PROTECTION=(W) Txxx:/DEVICE." (Modem ports and other certain public devices may need to have a more relaxed protection such as W:RWPL). Terminal protections should reduce the likelihood of something called a grabber program. A grabber program is a small user-written routine that allocates a terminal and simulates the login sequence. It waits for some unsuspecting user to hit return and types the message specified in SYS\$ANNOUNCE then the phrase "Username:". Guess what. The user enters their username; the program prompts for "Password:"; the user enters their password; finally the program writes both the username and password to a "recording" file for later reference. Now, because the grabber program can't really perform a log-in sequence it simply displays the message (which we've ALL seen ourselves) "user authorization

failure" and exits. No big deal, the user simply tries again thinking he fumbled the password.

Device protections alone can't prevent a grabber program, they can only help curb them. Under VMS V4.x, if you've had any log-fail attempts since you were last logged on, you'll get a message similar to:

```
Welcome to VAX/VMS version V4.x
Last login on Tuesday, 2-JUL-1985 13:25
Last batch on Tuesday, 2-JUL-1985 09:49
5 failures since last successful login
```

You should instruct your users to pay attention to this display and report any excessive log-fails that they can't account for. Just as important, if they do get a log-fail message they should verify that it is reported as above (if it's not, chances are there's a grabber program running around).

While on the subject of log-fails, check-out the new sysgen parameters LGI\_RETRY\_LIM, LGI\_RETRY\_TMO. They specify maximum numbers of log-fails within a certain time frame before an account will be rendered temporarily disabled.

Although VMS allows it, don't disable this feature.

#### Privileges

If ever there was a poorly chosen word for anything, "privileges" has to rank at the top. Users somehow feel that it's share and share alike: "If the system manager has SETPRV privilege, so should I". Some users are a little more crafty (usually the system programmers), they'll come with their manager and exclaim they just can't finish their current project without just one little privilege... Which one? No, not SETPRV (that's to blunt) just CMEXEC. (CMEXEC allows one to get into kernel mode unchecked and mess with the system data structures. With CMEXEC they can do anything on the system they please including giving themselves any other privilege they desire.) You must know what privileges allow what; otherwise, you're at the mercy of the first knowledgeable programmer that comes along. Read the descriptions of the privileges closely (refer to the security manual) when assigning anything more than just the standard TMPMBX or NETMBX to an account. If a request arrives for anything else, make them justify it in gory detail.

Even when users have justifiable needs for heavy-duty privileges, monitor their actions. The CMKRNL privilege, for example, allows the running of "ANALYZE/SYSTEM." With this utility it's a simple matter to look at terminal typeahead buffers and observe other user's passwords as they log-in. Including yours.

One way of avoiding issuing privileges is to create a captive account. Someone logs-in to a particular account and is restricted, by means of a CAREFULLY written command procedure, to only a few commands. An example might include the need to run the INSTALL utility to install a shared global section.

(INSTALL requires CMKRNL privilege.) Rather than give someone CMKRNL, create an account, suitably privileged, that has as its only function, the execution of a command procedure that performs the required installation. WARNING! Read the DEC security manual regarding captive accounts. Here are some things to watch out for concerning captive accounts:

- o Limit mail usage. Someone might try to invoke mail with the "/EDIT" qualifier and include some sensitive files from other directories. With mail they might also create some unwanted files (like a new LOGIN.COM maybe).
- o Don't allow the use of the TECO editor. There is a command within TECO that will permit exiting any command procedure.
- o Be sure to create the account as CAPTIVE and disable control-Ys. DEC also recommends setting DISWELCOME and LOCKPWD.

I'll illustrate this point with another tale of woe close to me. A particular site (remaining nameless) had created a particularly well endowed captive account (having CMKRNL privilege), called INSTALLGLBS, but had failed to add the CAPTIVE flag. Imagine our (oops, someone's) surprise when the user logged in as:

Username: INSTALLGLBS/DISK=CSA9:

Well, very few systems have 9 CS: devices. Sure, VMS complained about not finding CSA9 but it still allowed the log-in. The biggest reason for CAPTIVE is to ensure that the command file specified in the AUTHORIZE field LGICMD is execute. Our (oops, someone's) special, CAREFULLY written command file was ignored and egg was all over my (oops, someone's) face.

The more you read about privileges the more it seems like they all have a certain air of doom about them. Well it's true, they do. Here are some I would be extremely suspicious about:

- o SETPRV
- o CMEXEC or CMKRNL

Wary about:

- o LOG\_IO
- o PHYS\_IO

Those in the above list have "sneaky" or additional side effects that might not be readily apparent from reading their description. I'm sure the others might too, I just haven't worked out any really neat tricks with them yet. Among the others to look out for, include: BYPASS, READALL, SYSPRV, OPER, PFNMAP, SECURITY, SYSNAM, and WORLD. The descriptions of these are pretty good, you'll know what you're getting yourself into.

Another way of not granting privileges to someone is through the use of installed images. Images can be installed with privilege just like accounts.

So, if you want to write a "WHO\_IS\_ON\_THE\_SYSTEM" (Aren't V4 filenames neat?) program for everyone's use, but don't want to grant WORLD privilege to everyone (good for you) then create the program and use the INSTALL utility to grant just the image, the WORLD privilege. The command looks something like:

```
INSTALL> SYS$PUBLIC:WHO.EXE/PRIV=WORLD
```

If you're not running VMS V4.x yet, then there's another thing you've got to (MUST) look out for. You MUST link privileged images with a command like:

```
$ LINK/NOTRACEBACK WHO
```

The "/NOTRACEBACK" qualifier prevents someone from running the image in debug mode with a "RUN/DEBUG SYS\$PUBLIC:WHO" command. Running a privileged image with the debugger is a pretty easy way to inherit the privileges of the installed image. This is fixed in VMS V4.x, you can't install images with privileges if they were not linked properly. (Don't worry, if you don't have object modules, DEC has available a command procedure that will properly patch the image to prevent the debugger from running.)

#### Daily Operations

How many sites out there receive a new piece of software and without thinking, type RUN or @ to see what it does? Don't! At the very least give the thing a quick look to see "how" it does "what."

We recently purchased a word processing package that initializes itself at each use with a command file. We looked at the command file and it included the command:

```
$ SET PROT=(S:RWE,O:RWED,G:RWE,W:RWE)/DEFAULT
```

I find it somewhat unsettling to discover that these software packages go about changing such things as my default file protection, especially without me knowing it. Get the picture?

Be careful of software you don't control. Don't run programs out of directories you don't control. Here's yet another story (it's called a Trojan Horse):

Good ol' Joe, he's always writing this really neat code. He's just finished an amazing "SET DEFAULT" utility; you simply type SD and it reads your thoughts about what you would like your default directory to be. Good ol' Joe tells everyone it's finished and that it lives in the [GOODOLJOE] directory (his directory, of course). Everyone begins to use it, you included.

```
SD and it puts you at SYSEXE, good
SD and it puts you at SYSUPD, good
```

Then,

```
SD ... it pauses ... it puts you at SYSMGR, good
```

What you didn't realize was that SD detected it was running with privileges (your account) and the pause was really SD busy copying the company's financial data to one of Good ol' Joe's subdirectories. SD finished it's original task of changing your default and exited. Nobody the wiser.

Enough said, except to say be careful of updates and new releases. What was harmless today, may not be harmless tomorrow.

On the topic of software, examine incoming and outgoing tapes, disks, etc. Don't let somebody get you into trouble by bringing in or carrying out purloined software. If you do a show process and see somebody running Ada, and you know you don't have an Ada license, illegal software is rummaging about your system and it probably came in by magtape. Tapes that you loan or give to users for their use, degauss them first. Initializing a tape with the VMS "INIT" command is not sufficient (INIT only writes on the very first part of a tape).

Limit access to the operator's console. Beyond just having information printed on it that users don't really need to see, it's quite a powerful little device. Ready for another story?

Yet another famous site... Programmers were allowed into a certain computer room at any time, that's where the magtape drives were located that they needed. The only terminal in the area was, by design, the operator's console. One of programmers logged on, fumbled a few control characters and found the ">>>" prompt (LSI monitor prompt). Curious about what utility prompted with ">>>" he resorted to what he always did, typing "H" for HELP. Well, no doubt you've guessed what happened (for you non-VAX folks "H" is also short for Halt and the VAX complied).

The brave user given access to the console might even attempt to re-boot the machine. There are all kinds of potential security problems when someone else is able to re-boot, don't allow this. At re-boot time the VAX is at its very most naked, vulnerable state. It comes to no surprise to you, I'm sure, that all older VAX cabinet keys are standard and identical. Everyone hoped that with the new FCC cabinets we'd all get new and different locks. Well what do you know, we did! Now the bad news, check your keys. We have a particular brand of magtape cabinet that's pretty popular, it too has a key lock. In fact it has the same key as the master cabinet key for the VAX!

In no particular category, here are some other operational tips:

- o Re-define the logical SYS\$SYLOGIN to a file with some other name than SYLOGIN.COM. Users don't really need to see what's in that file and if you rename it you may slow or deter some probing.

- o Re-define the logical SYSUAF for the same reason.
- o When users leave a company or lose access to their account, don't remove it from the authorization file. Better to just inactivate the account with the "/DISUSER" qualifier. That way if they secretly come back and try to gain access to the old accounts, you'll be able to determine that with the accounting utility. On the other hand, if you remove the account, log-fail attempts show up as simply <login>. If you leave the account you'll get the name.

Backups are critical for numerous reasons, be aware of their security though. Remember that backups have a complete copy (or should) of all your disk files, programs, data, on and on. Unlike disk packs/drives, magtapes can easily be fitted into a briefcase; don't let people walk out of your site with last week's complete image save. Protect them. In addition to the physical security of the media, there is a need for a logical volume protection. When you perform your backups, use the "/PROTECT" qualifier to prevent others from mounting the tape and gaining access to otherwise protected files. NOTE! NOTE! NOTE! Somewhere around the upgrade from VMS V3.4 to V3.5 BACKUP's default for the "/REWIND" qualifier changed from "/REWIND" to "/NOREWIND", not all sites caught the implication of this. BACKUP only records the volume protection when it's allowed to initialize the tape. The only time BACKUP initializes a tape is when "/REWIND" is included. Get it? If you just use "/PROTECT" without "/REWIND", BACKUP ignores the protection qualifier. A good number of sites have command files to perform all of their backups. If the command files aren't updated to include the change, all those tapes are left with whatever the previous volume protection is.

#### Networks

I'm only going to pass along a few points concerning DECNET.

- o All DECNET nodes not under your control should be considered hostile. That is, the information passing between nodes should be considered public domain.
- o Ethernet networks are especially susceptible to eavesdropping by other nodes.
- o Protect all files [SYSEXE]\*NET\*.DAT to disallow world access.
- o Use proxy log-ins wherever possible to eliminate the need for explicit access control strings.
- o The need for privileged DECNET default accounts has just about been eliminated.



## Conclusion

In conclusion, I'd like to emphasize that there is a tremendous amount of information to be found in DEC's new security manual, be sure to read at least chapters 1-8 and the appendices A-C.

Don't wait until it's too late to find out that you have security problems, by then you may just be the next sensational wizkid-cracks-computer news event we hear about.

## Acknowledgements

This paper has drawn upon the expertise and opinion of several people, among them in no particular order:

Eric Zipp (TDC, Arlington, Texas),  
James Fischer (xxxxxxxxxxxxxxxx),  
Doug Brown (Sandia National Labs, Albuquerque,  
New Mexico),  
Stephen Thor (NYU, NY, NY),

Gentlemen, my thanks. I would especially like to thank Mark Pilant (DEC, Nashua, New Hampshire) for his review of a draft of this paper and for his helpful suggestions.

I also extend my appreciation to DEC and its writers for their manual Guide to VAX/VMS System Security.

As always, although I've taken great care to avoid any mis-information, I take sole responsibility for any errors appearing in this text.

THE INSTRUCTION UNIT OF THE VAX 8600  
A PIPELINE IMPLEMENTATION OF THE VAX ARCHITECTURE

Fernando C. Colon Osorio, Steve Ching, Mario Troiani  
John Bloem, and Nii Quaynor

High Performance Systems and Clusters  
Digital Equipment Corporation  
Marlboro, Massachusetts 01752

ABSTRACT

The instruction and operand fetch unit (IBOX) of a High Performance Implementation of the VAX architecture, the VAX 8600, is described in this paper. The VAX 8600 delivers a performance speed-up over previous implementations by the use of HIGH SPEED ECL technology, and an internal organization that consists of a four stage pipeline. In this four stage pipeline, up to four simultaneous instructions can be in several stages of execution at any time. This parallelism contributes to an overall performance improvement of more than four times over the VAX 11/780. Furthermore, under favorable conditions, the IBOX can deliver to the Instruction Execution Unit (EBOX) one instruction every 80 nsecs, which means this high performance implementation is capable of executing instructions at a peak rate of 12.5 mips. In this paper, special attention is given to the internal organization of the VAX 8600 IBOX as it differs from previous VAX implementations.

Keywords: VAX instruction set, pipelined architecture, global control, local control, register log, scoreboard, stall conditions, data dependency, control dependency, resource dependency, elasticity, rigidity

1 INTRODUCTION

THE VAX 8600 Computer System is the first pipelined implementation of the VAX architecture [1]. Like its non pipelined predecessors, the VAX 8600 implements the full VAX instruction set and runs under the VMS (or UNIX/ULTRIX) operating system. In addition, the primary goal for the VAX 8600 was to provide higher performance and reliability than its predecessor, the VAX 11/780. In this context, the performance speed up factor needs to be clearly defined if we are to avoid the confusion that usually arises when discussing performance. First, let us define a given program's speedup factor as the time it takes to execute on the VAX 11/780 divided by the time to execute on the VAX 8600. The VAX 8600 "ideal" or "true" measure of performance improvement is then the average over all programs of such speedup factor. Since the universe of all programs is too large, one selects a proper subset of favorite benchmarks for the comparison. This subset of benchmarks

can be labelled as the constant unit of work (CUW), and its selection is often the reason for conflicting reports in the literature. The execution time of this constant unit of work in our model, is the product of three quantities: the number of instructions, the average number of cycles per instruction, and the cycle time of the machine under evaluation.

The performance aim of the VAX 8600 was to reduce the average number of cycles per instruction from 10 in the VAX 11/780 to 6, and also to reduce the cycle time of the machine from 200 nsecs in the case of the VAX 11/780 to 80 nsecs. In order to achieve the goal of reducing the cycle time of the machine, custom ECL gate arrays and standard 10K ECL logic was utilized throughout the design. This technology provided the  $2\frac{1}{2}$  times performance improvement that was required. The rest of the performance gain was contributed by achieving the goal of reducing the average number of cycles per instruction through the use of a four stage pipeline. This four stage pipeline

is capable of overlapping the fetching of instruction stream data, with the decode of instructions, the prefetch of operands from memory, and the execution of instructions, see Figure 3c. In the VAX 11/780, on the other hand, the operand address calculation, operand fetch, and operand write stages are all merged into the execution stage (EBOX), see Figure 3b. In the VAX 8600 up to four simultaneous instructions can be in several stages of execution at any time.

The remainder of this paper is organized as follows. In section 2, a limited description of the VAX instruction set is presented. Section 3, provides an overall description of the VAX 8600 internal organization with special emphasis on the major busses that support the flow of instructions and data between stages. Section 4 introduces an abstract model of pipelines, and describes the VAX 8600 in terms of the major components of the model. Section 5 describes in detail the internal organization of the instruction unit (IBOX) and its associated control structure. Finally, in section 6 our conclusions are presented.

## 2 VAX INSTRUCTION SET

The VAX Architecture [1] has a rather rich and powerful instruction set. Each instruction, in general, consists of one byte of opcode, optionally followed by one to six operand specifiers. These specifiers represent the accessing scheme for an operand, or the displacement in a branch instruction, or the target address in a call type of instruction. The data type and usage of each specifier is derived from the opcode. There are also two bytes long opcodes for multi-precision floating point operations, instruction set extension, and user defined operations. The instruction set is standardized so that each VAX implementation is able to execute the same software image as well as the same operating system environment. This compatibility is the basic goal for all VAX implementations, including the VAX 8600.

## 3 VAX 8600 STRUCTURE

Functionally, the CPU (see Fig. 1) consists of four separate microcoded units for memory and I/O (MBOX), instruction fetches and preparations (IBOX), execution (EBOX), as well as a co-processor for high speed floating point execution (FBOX). A description of these subsystems and the interconnecting busses follows.

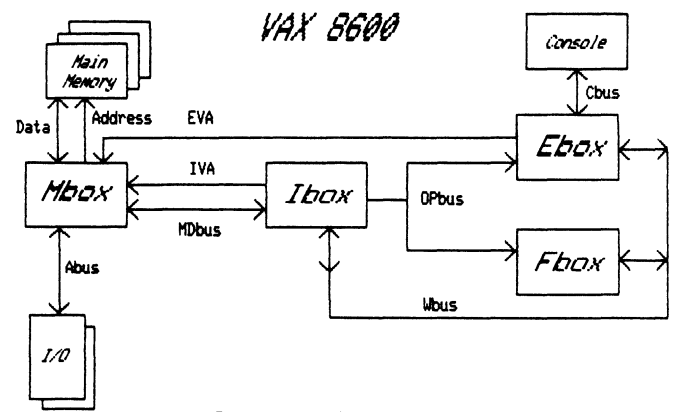


Fig. 1. VAX 8600 CPU Organization

### 3.1 System Busses

There are a number of internal busses that are key to the organization of the VAX 8600. These include:

**IVA** - the IBOX Virtual Address bus, which carries virtual addresses from the IBOX to the MBOX during instruction fetch, operand fetch and IBOX Write operations.

**MDBUS** - the Memory Data bus, a data bus for both reads and writes to MBOX memory.

**OPBUS** - the Operand bus, which carries operands from the IBOX to the execution units.

**WBUS** - the Write bus, which contains execution unit results to memory via the IBOX or to General Purpose Registers (GPR's).

**EVA** - the EBOX Virtual Address, which contains virtual addresses from the EBOX to the MBOX during EBOX operand references and certain memory management routines.

**ABUS** - the I/O bus, which interfaces the CPU to the outside world.

### 3.2 MBOX - The Heart Of System Communication

The primary purpose of the MBOX is to tie together the main memory, the physical cache, the cpu ports and the I/O subsystem. In this capacity it is the communication center at the system level.

The MBOX contains a cache for instructions and data, a virtual address Translation Buffer (TB), and the physical memory. These resources are accessed by an I/O port and three other fixed priority cpu ports, as shown in Fig. 2. The MBOX being a system communication center must contend with several concurrent activities requiring communication services. To cope with these numerous state requirements the

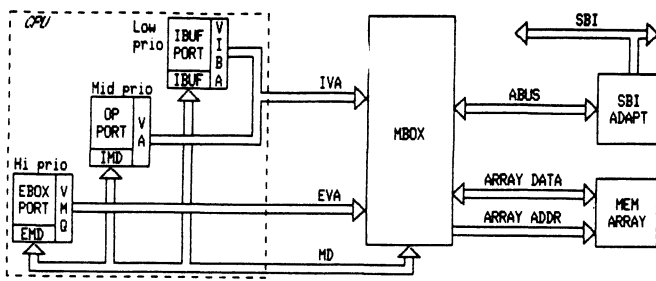


Fig. 2. Port Organization

MBOX is heavily microcoded and also calls upon EBOX microcode on occasions to assist with some memory management functions. The MBOX has the capability of queuing a number of memory requests from both the instruction fetch and execution units. Both the IBOX and EBOX can request MBOX service through their own memory ports and busses (IVA and EVA). A common MDBUS provides interface for both read and write data. Input/Output data are handled via adapters that interface to the ABUS. A more detailed description of the MBOX can be found in [2].

### 3.3 IBOX - The Heart Of The Pipeline

The primary purpose of the IBOX is to continuously feed microcode dispatch addresses and operands to the execution units (EBOX and FBOX), so that the latter may execute the VAX instruction set. In order to do so, the IBOX must prefetch the instruction stream from the MBOX and then interpret it: parse the specifiers, fetch the operands and build the dispatch address for the EBOX. Three pipeline stages, including a microcoded operand address calculation engine, are used to implement this functionality at high throughput. This results in extensive control logic needed to synchronize the flow of data and control through the pipeline. Furthermore, the IBOX contains the logic to maintain the many program counters representing the different instructions executing concurrently in the pipeline.

The virtual ownership of the pipeline, including the critical EBOX dispatch interface, the almost exclusive MBOX interface, and the maintenance of the PC's make the IBOX the heart of the pipeline. It is thus the target of much of the complexity of the VAX 8600.

### 3.4 EBOX And FBOX - The Heart Of The VAX Architecture

In general, EBOX and FBOX consume the dispatch address and operands setup by the

IBOX and perform only the operations as specified in a macro instruction. In this mode, they are isolated from memory access and freed from specifier evaluation and operand fetching. They can thus be optimized for high speed execution. The EBOX also performs the secondary function of the management of the boundary conditions for both hardware (machine checks such as single and double bit memory errors and parity errors) and of the VAX architecture (interrupts and exceptions). In particular, memory management is almost totally performed here: TB misses, page faults and access violations, page crossings and unaligned EBOX memory references are detected by the MBOX but are all serviced by the EBOX. In this respect, the execution units are the heart of the VAX architecture.

## 4 THE VAX 8600 PIPELINE

Pipelined computers are not new: from the early days of the IBM Stretch [3] and the IBM 360/91 [4] to the scalar units of the CDC [5] and CRAY [6] machines, pipelining has been a proven if expensive method for performance enhancement.

In most Von Neumann processors the instruction fetch and decode functions are performed sequentially in the only "stage", the execution unit, which is also the entire cpu. A typical example is the PDP11, where the concurrency is microprogrammed (see Fig. 3a).

Most existing VAX implementations have added the stage for instruction prefetch, thus reducing the instruction fetch latency, the primal example being the VAX 11/780 (see Fig. 3b).

The VAX 8600 is the first implementation of the VAX Architecture that separates instruction preparations, e.g. effective address calculation and operand fetches, from instruction execution itself (see Fig. 3c).

The significance of the VAX 8600 design lies in the successful resolution of the implementation difficulties which stem from the combined complexities of the VAX Architecture and the pipeline approach: the more complex an architecture, i.e. the more the control and data dependencies, the more difficult it is to pipeline it.

While the basis and fundamentals for such designs can be found in [7,8], and a more recent pipeline model is discussed in [9], we present here a simplified model for the purpose and scope of this paper. Then we show the VAX 8600 pipeline using such model.



Fig. 3a. PDP11 Instruction Execution

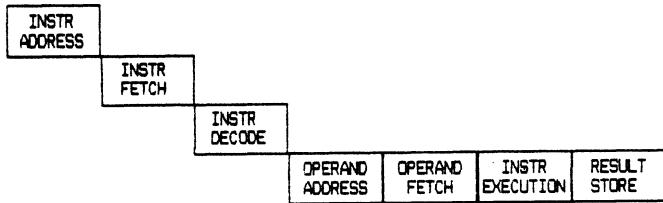


Fig. 3b. The VAX 11/780 Instruction Pipeline

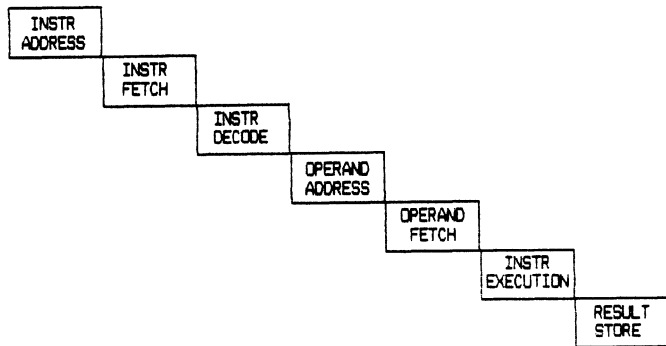


Fig. 3c. The VAX 8600 Instruction Pipeline

#### 4.1 A Pipeline Model

Let's define a pipeline stage, see Fig. 4a, as an entity with four fundamental attributes:

STAGE := (FUNC, HW, PREC, PMC)

where:

FUNC := the "function" of the stage, usually an INPUT BUFFER, an OUTPUT BUFFER and a MAPPING between the two.

For example the function of the Operand Access Unit stage (OAU) is to compute an operand effective address and then to fetch it from the MBOX and to load it into the output buffer (IMD - the IBOX Memory Data register).

HW := the "hardware residency" of the stage. For example the OAU stage resides in the IBOX hardware.

PREC := the "precedence" of the stage in the sequence of stages. This precedence is fixed and means that the Instruction Decode stage, for example, is a successor of the PREFETCHER stage. Note that the precedence relation is a logical concept and not a physical one. That is, although the memory write function of the EBOX stage is the last stage of the pipe, it shares resources with the Operand Access Unit stage.

PMC := the number of "physical machine cycles" needed to execute a stage's "logical cycle". Conceptually then, an item gets processed and goes through the stage in a single logical cycle. The reason for this distinction between logical and physical cycles will become apparent with the examples below:

Example 1: consider the OAU stage processing of simple specifier, such as register mode. In this case, one logical cycle equals two physical cycles.

Example 2: consider again the OAU stage processing of a complex specifier, such as longword displacement deferred indexed, @LD(Rn)[Rx], with a cache miss in the indirect reference. In such a case one logical cycle will equal N physical cycles, where N is directly dependent on the state of various system resources.

A stage may consist of one or more "units", each consuming a minimum of one physical cycle. A "pipeline" is a sequence of stages connected by "transport" mechanisms, which move an item from the output buffers of a stage to the next one. Except for the first and last stage, such a structure can be partitioned into a "current" stage, all its "precedent" stages, and all its "subsequent" stages. One can also define the "predecessor" stage as the immediately precedent one, and the "successor" likewise.

What has been described so far can be considered the "data path" of a pipeline.

#### 4.2 Control Of The Ideal Pipeline

While the data path of a pipeline model just discussed is a simpler concept, its control is more difficult. In the ideal case shown in Fig. 4a, the synchronization

is relatively simple and is based on "local control".

LOCAL CONTROL := the "stop and go" of the pipeline is controlled by flags which are transported together with the items. These are the "valid flags" of the input and output buffers. The following two basic operations can give them the values of either "empty" or "full":

LOAD - at the completion of a logical cycle, a stage writes an item into its output buffer and sets the buffer's valid flag to "full".

DRAIN - at the beginning of a logical cycle, a stage reads an item from its input buffer and sets the buffer's valid flag to "empty".

Depending on the operation and on certain values of these flags, one of these conditions can occur:

INPUT STALL := the input buffers valid flags are "empty" and the stage wants to drain them. Then the stage must not load the output buffers.

OUTPUT STALL := the output buffers valid flags are "full" and the stage wants to load them. The stage must then stall to avoid data overrun.

Even in the case of an ideal pipeline, an important performance issue is that of "elasticity" of the pipe. That is, the ability of the pipe to deliver results at full bandwidth in spite of its "irregularity". Irregularity usually refers to the fact that different stages in the pipe have different PMC's and hence the time to process an item in each stage, its logical cycle duration, is variable. "Rigidity", the reciprocal of elasticity, measures the dependence of a stage on the stalled state of another stage. In other words, the rigidity is related to the speed with which the stall flags ripple through the stages, in either direction. Rigidity is counterproductive in that it stifles concurrency. For that reason, extra buffering is sometimes used: it allows a stage to execute even if some output buffers are already full, thus relieving output stalls. This also means that the successor's input buffers will be able to be "preloaded", thus relieving input stalls as well.

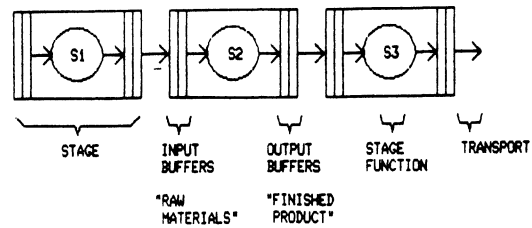


Fig. 4a. An Ideal Pipeline Model

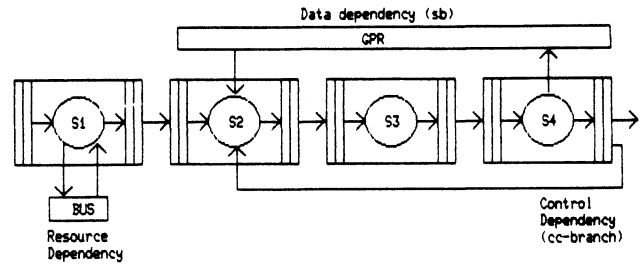


Fig. 4b. Pipeline Dependencies

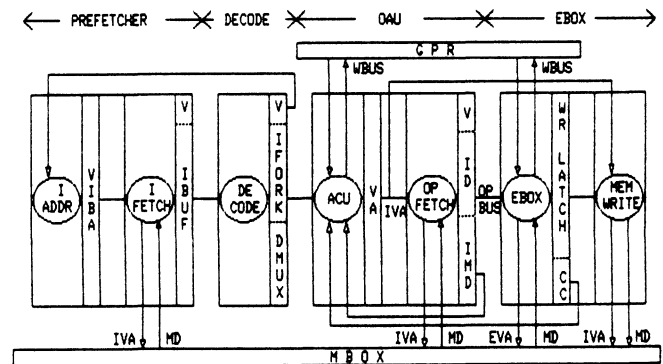


Fig. 4c. Simplified VAX 8600 Pipeline Model

However, simple minded FIFO extra buffering may introduce the negative effect of increasing the pipeline "latency", i.e. the number of physical cycles needed by an item to travel through the entire pipeline. This effect can be minimized by the use of "bypass" circuitry, as described in [9], at the cost of a very significant amount of control complexity. To minimize such complexity, one can reduce the number of buffers to just one. In this case, the stage buffer functions both as the output buffer of a stage and as the input buffer of its successor stage. The VAX 8600 design is very close to such model, see Fig. 4c.

### 4.3 Dependencies

While any pipeline model has embedded the "trivial" dependency of a stage on the

predecessor stage, a more realistic pipeline model (see Fig. 4b) must include nontrivial data and control dependencies. These two factors make the implementation of the pipeline more difficult. Such dependencies can be grouped into two dimensions. These are: type (data/control) and direction (forward/backward).

**DATA:** dependency of a stage on data values produced by other than the predecessor stage.  
Example: the OAU stage must wait until the EBOX has updated a GPR before it can use it in the address calculation, see Fig. 4c.

**CONTROL:** dependency of a stage on control produced by other than the predecessor stage.  
Example: the OAU stage, which also processes branches, must wait until the EBOX has generated the condition codes for the instruction preceding the branch before it can resolve it, see Fig. 4c.

Each of the above dependencies can operate in either direction:

**BACKWARD:** a piece of a data or control item affects a precedent stage.  
Example: either example above.

**FORWARD:** a piece of a data or control item affects a subsequent stage.  
Example: the IBOX Write address dependency, which is described later in Section 4.4.

Separate from the above, there are "resource" dependencies, when a stage needs to use a resource shared among many stages. The memory in the MBOX, for example, is a resource shared by three of the VAX 8600 stages. All of these dependencies sometimes allow a more efficient control of the pipeline via "global control".

**GLOBAL CONTROL** := the "stop and go" of certain stages is controlled by key flags which are broadcast to them by another stage. Note that this mechanism operates in conjunction with the local control.

In the next section the concepts just introduced will be used to represent the VAX 8600 in terms of a simple pipeline abstract model.

#### 4.4 A Simplified VAX 8600 Model

A simplified model of the data path portion of VAX 8600 pipeline is shown in Fig. 4c. In this model the FBOX is not shown, as its locus of control is very similar to that of the EBOX. This four stage design has two critical resource dependencies: the MBOX, which is used by the the Instruction Prefetch stage (PREFETCHER) and sometimes by the Operand Access Unit (OAU) and EBOX stages; and the GPR's, which are used normally by the OAU and EBOX stages.

Before discussing the simplified model, let's follow an instruction as it goes through the pipe.

At the beginning of instruction processing, assume that all the IBOX buffers are invalid. In this case the EBOX dispatches the instruction prefetcher at the new instruction stream address. The PREFETCHER stage starts prefetching and loading instructions into the Instruction Buffer (IBUFFER). This is actually a simplification; the detailed mechanism is described in Section 5.1. The Instruction Decode stage (DECODE) drains the IBUFFER, and from the opcode generates a microcode dispatch address (not shown) for the EBOX. The Operand Address Calculation unit (ACU) in the OAU stage parses the operand specifiers and computes their effective address, in the process reading and possibly modifying the GPR's (e.g. autoincrement mode, (Rn)+). The Operand Fetch unit (OPFETCH) fetches these operands at that effective address and passes them to the EBOX. The EBOX then executes the instruction it was dispatched to; in doing so it drains the operands and writes the result into the GPR's, in case of register destination. In case of memory destination (and only in that case), the Memory Write unit (MEM WRITE) is used: it takes the result data from the EBOX and writes to memory via the Operand Port (see Fig. 2) at the address forwarded by the ACU unit. Such mechanism is called "IBOX Write".

Let's now look at each stage of the pipe of Fig. 4c more in detail.

The Instruction Prefetch stage (PREFETCHER) is composed of the Instruction Address Calculation unit (IADDR) and the Instruction Fetch unit (IFETCH). The IADDR unit computes the next value of the Virtual Instruction Buffer Address register (VIBA) and issues an IBUFFER Request. The IFETCH unit fetches a longword from the VIBA register and loads it into the IBUFFER. This stage resides in both the IBOX and the MBOX. Its logical cycle lasts two physical cycles in the case of a cache hit, or N physical cycles

otherwise, where N depends on the memory access delay.

The Instruction Decode stage (DECODE) is composed of only one unit and always executes in one physical cycle. It decodes opcodes and specifiers from the IBUFFER and loads control data into the IFORK buffer and instruction stream data into the DMUX buffer. The DECODE stage resides entirely in the IBOX.

The Operand Access Unit stage (OAU) is composed of the ACU unit and the OPFETCH unit. The ACU unit computes an operand effective address, loads it into the Virtual Address register (VA), and issues an Operand Request. The OPFETCH unit fetches the operand from the MBOX and loads it into the the IBOX Memory Data register (IMD). The OAU stage also forwards VA to the MEM WRITE unit. Note that this stage can contain two instructions at any given time. The OAU stage resides in both the IBOX and the MBOX. Its logical cycles lasts a minimum of two physical cycles.

The EBOX stage is composed of the EBOX unit and the MEM WRITE unit. The EBOX unit executes instructions and stores results either into the GPR's or into the Write Latch for memory writes. In this case it initiates an IBOX Write command. The MEM WRITE unit actually performs the write operation at the VA address forwarded by the OAU stage. The EBOX stage resides in the EBOX, FBOX, MBOX, and partially in the IBOX for memory writes. Its logical cycle lasts a minimum of one physical cycle for non IBOX Write instructions. It lasts at least three for IBOX Write instructions.

In the simplified model each stage has only one output buffer, which functions also as the input buffer of the successor stage. Thus a drain operation is implemented as an interstage drain signal. Note that in this case the elasticity of the pipe is reduced to a minimum. In the worst case, if the pipe is full and the last stage stalls, then all the stages in the pipe will stall. Also, since a stall condition must be detected before loading the output buffers, in certain cases the stall conditions may become more forgiving:

INPUT STALL := the input buffers valid flags are "empty" and the stage wants to drain them, AND the predecessor stage "doesn't intend" to load them.

OUTPUT STALL := the output buffers valid flags are "full" and the stage wants to load them, AND the successor stage "doesn't intend" to drain them.

In such a model there are some interesting examples of control dependencies:

- a) the PREFETCHER stage issues an IBUFFER request to the MBOX on the basis of the number of valid bytes in the IBUFFER, on the validity of the DECODE stage's output buffers, and on the "intention" of the OAU stage to drain them.
- b) the OAU stage must wait for the EBOX to complete the instruction preceding a branch in order to resolve it and start prefetching at the target address.
- c) in the case of an IBOX Write, the OAU stage forwards to the MEM WRITE unit the destination address (hardware-wise, it stays in the VA register). MEM WRITE then waits for the EBOX unit to provide the data. The reverse can occur: the EBOX may have to wait for disposing of the data when the ACU unit takes a long time to compute the effective address.

## 5 IBOX

The three pipe stages residing in the IBOX are physically composed of:

- a) An instruction prefetch stage (PREFETCHER in Fig. 4c), which prefetches the instruction stream for the IBUFFER. It is also used to fetch string data in string instructions.
- b) Decoding logic which produces dispatch addresses based on opcode and its specifiers for the operand address calculation unit micromachine and the EBOX. This is the DECODE stage as defined in the pipeline model.
- c) A micromachine, called the ACU micromachine, which implements the functionality of the OAU stage and part of the MEM WRITE unit. This includes operand address calculation, operand fetches, and result writes.

Notice that the part of the MEM WRITE unit resides in the IBOX. It maintains the memory write address for result operands and shares responsibility with the EBOX unit to perform the actual result write.

In addition, the IBOX maintains:

- a) a number of program counters for tracking different instructions being



executed at different stages in the pipe,

b) a local copy of the GPR's for operand effective address calculations and operand sourcing,

c) a register scoreboard for resolving register access conflicts,

d) a Register Log (RLOG) for register state restoration during exceptions and interrupts,

e) a branch decision mechanism, and

f) a number of control mechanisms to synchronize the pipeline.

The importance of the VAX 8600 IBOX lies in the many functions it has to perform, and the extensive controls required to correctly synchronize all four stages of the pipe.

Figure 5 depicts the datapath of the IBOX. The following sections describe the function of many features in the IBOX.

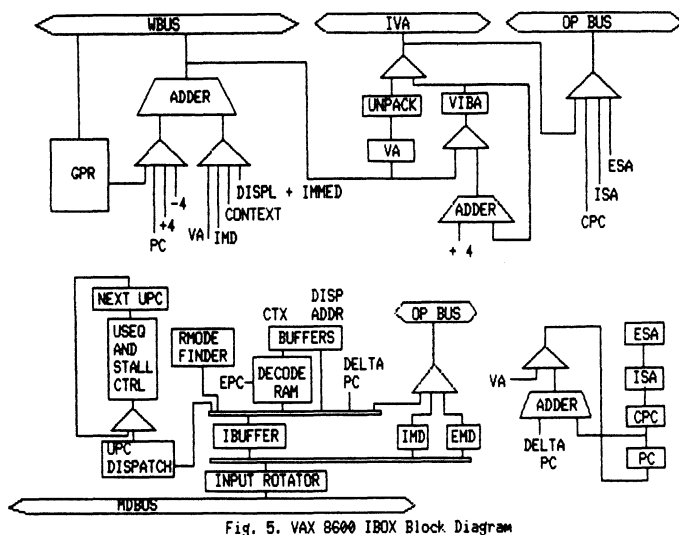


Fig. 5. VAX 8600 IBOX Block Diagram

### 5.1 Instruction Prefetch

The prefetcher has an eight bytes wide IBUFFER and an associated addressing and control logic. It attempts to initiate a prefetch whenever an empty byte is detected inside the IBUFFER. The Virtual Instruction Buffer Address (VIBA), register contains the next address in the instruction stream to be fetched from. Prefetch request addresses share the IVA bus with the ACU unit (see Figure 2). Since operand fetch is a result of executing an already decoded instruction it has a higher priority in using the IVA bus. Prefetches, on the other hand, can be postponed and thus have lower priority.

The memory subsystem queue can accept a second prefetch even if a previous prefetch is still in progress. This mechanism results in better utilization of the available memory bandwidth. Data received through the MDBUS is loaded into the appropriate location inside the IBUFFER. The VIBA is updated to form the next address whenever a prefetch request is accepted by the MBOX.

During a cold start, after an exception, or for certain branches, such as the CASE type of instruction, the prefetch sequence must start fresh from a new instruction address. In this case, the EBOX places the new address on the WBUS and dispatches the ACU micromachine to an IBOX startup sequence. Instead of loading the address to VIBA and starts prefetching, the ACU micromachine initiates two consecutive requests before handing off prefetching to the prefetch unit.

For some instructions requiring stream data or a stream of operands to be read consecutively from memory in its execution, the IBUFFER becomes a high speed data buffer supplying operands to the EBOX through the OPBUS.

### 5.2 Instruction Decode

Instruction decoding in the VAX 8600 is similar to that in the VAX 11/780, in the sense that the operand specifiers are decoded sequentially one at a time. When the IBUFFER contains prefetched instructions, byte zero contains the opcode of the current instruction, and byte one the first byte of the specifier currently being decoded. An instruction is decoded by looking up from a decoding RAM (DRAM), which is organized as an array of 512 sections, each of which has eight entries. Each entry is addressed by its section and entry index. The opcode byte plus the fact of whether the instruction has an extended opcode will address the section. The Execution Point Counter (EPC), which essentially is a pointer indicating the position of the specifier in the instruction will select the particular entry. The output of the DRAM consists of information specifying the data context (byte, word, long, etc), data type (address, integer and different floating point formats), and accessing mode (such as read, write or modify) for each specifier. It also provides the dispatch address (EFORK) to the EBOX.

After each specifier decode, the IBUFFER shifts out the consumed specifier and shifts into the decoding position the next specifier. The DECODE stage also increments the EPC so that the new decode points to the next DRAM entry. The output of the DRAM plus data extracted from the specifier, such as GPR information and literal value, is buffered for the OAU

stage.

During decoding, using the specifier byte, a dispatch generation mechanism creates a dispatch address(IFORK) for the ACU micromachine. This process will continue until the last specifier of the instruction is decoded and consumed. A bit in the DRAM output will indicate such occurrence. When this happens, the IBUFFER shifts out byte zero and the last specifier, thus allowing a new instruction to be shifted in.

### 5.3 The ACU Micromachine

With reference to the simplified pipeline model(Figure 4c), the ACU, OPFETCH, and MEM WRITE units are described here together. In this way, their functionality and synchronization mechanisms can be appreciated better. The IFORK saved in the DECODE stage provides the entry to the proper microsequence routine in the ACU micromachine. Using the buffered DRAM and specifier data, the ACU micromachine performs the necessary computations to calculate the effective virtual address and initiate operand reads from memory or from the GPR's if necessary. A copy of the GPR's is maintained in the IBOX so that register access can be done locally i.e. faster. This allows also register accesses (reads) by the IBOX, EBOX and FBOX simultaneously.

For an operand which comes from a register source, data read from GPR file, after passing through the ACU adder, will be loaded to the instruction data(ID) register. Immediate data, which comes from a buffer in the DECODE stage, takes a similar route through the unpack logic to the same ID register. The operand data is then ready for the EBOX through the OPBUS. The unpack logic is used to convert fixed point short literals to a floating point format.

For an operand fetch from memory, the ACU micromachine loads the operand effective virtual address from the adder into the VA register and issues an operand fetch request through the IVA bus. The IMD register holds any operand data returned from MBOX before forwarding it to the EBOX through the OPBUS. If the addressing mode is indirect(e.g. autoincrement deferred) then the returned data in IMD is the final virtual address of the operand. The ACU micromachine loads IVA with IMD data and issues another operand fetch request. The EBOX Memory Data register(EMD) serves a similar function, but holds memory data returned as a result of EBOX requests. Placing the EMD physically in IBOX eliminates the need for the EBOX to interface with the MDBUS directly.

The ACU micro sequences for many simple and frequently used specifiers take one

cycle, so that one specifier can potentially be processed in each cycle. Some examples of such specifiers are: the register mode, Rn; the register deferred, (Rn); byte, word, and long displacement modes, B^D(Rn), W^D(Rn), and L^D(Rn). The successful processing of an operand specifier in the OAU stage also loads the earlier buffered EFORK into a register accessible by the EBOX.

The completion of an operand fetch may take many cycles if the source is in memory. Here, the execution unit may have already started executing the EFORK microsequence, and attempt to read and use the source operand which is not yet available. To resolve this the OAU stage provides additional operand data valid flags.

The ACU micromachine also issues the actual operand write request for most instructions. In this case, the micromachine saves the calculated destination address and waits until operand results are ready from the EBOX. When the results are ready, the EBOX writes them, via the WBUS, into a register internal to the IBOX, the WR LATCH. It also releases the ACU micromachine to issue the appropriate operand memory write request.

### 5.4 Multiple Program Counters

The VAX 8600 CPU maintains a number of program counters for each of the instructions under execution in the pipe. This is necessary so that instruction restart after exception fixup is possible. The program counters are:

- a) Program Counter(PC) which follows the opcode, operand specifier, and immediate data or addresses as they are decoded.
- b) Current Program Counter (CPC) which points to the instruction to be executed next in the OAU stage. Normally, this is the instruction currently being decoded.
- c) IBOX Starting Address (ISA) which points to the instruction being executed in the OAU stage.
- d) EBOX Starting Address (ESA) which points to the current instruction being executed in EBOX and FBOX.

The prefetcher maintains its own instruction stream address pointer, the VIBA register, for requests to fill the IBUFFER.

The updating of the CPC, ISA, and ESA happens when an instruction enters the DECODE, OAU, and EBOX stages respectively. In general, CPC will be loaded with the address of the beginning of the

instruction to be decoded. ISA will be loaded with CPC when the OAU has started processing with that instruction. Similarly, ESA will be loaded with ISA when the EBOX begins to execute that same macro instruction.

### 5.5 Instruction Backup And Unwinding

In the VAX architecture, an exception may occur during the execution of an instruction. An example of an exception will be a page fault on a memory read. For most instructions, the VAX architecture requires that the program state be restored to that prior to the execution of the instruction, so that after a fix up sequence, the same instruction can be restarted. For some types of instructions, such as the string instructions, total program state restoration is impossible. The constraint here is that those instructions must be able to continue after exception processing.

In the VAX 8600, the program state that is required to be restored are those GPR's which have been modified during address calculation, and the instruction starting address. Those addressing modes, such as the autoincrement and autodecrement, will modify GPR's and such information is kept in the RLOG. During instruction unwinding(also called instruction backup), the ACU micromachine will restore from the RLOG those affected GPR's. Since a number of instructions can be residing in different stages of the pipe simultaneously, the RLOG has enough entries allowing register restoring for multiple instructions. The PC for the instruction in question will also be restored from either CPC, ISA, or ESA depending on the state of the pipe stages. This mechanism is also used to handle interrupts.

### 5.6 Branch Instruction Processing

The IBOX also calculates the branch target addresses, and performs branch decision for most branches. This includes those conditional(e.g. BEQL, BNEQ), unconditional(e.g. BRB) branches, as well as the computed branches(e.g. ACBL, SOBGTR). Such decisions are made by looking at the appropriate bits in the condition code result from an execution prior to the branch. The branch prediction scheme used here is biased towards branch taken. Figure 6a and b shows an example of the microinstruction sequence for a branch instruction.

During a conditional branch, the ACU micromachine holds the branch target address in the VA register, and will

attempt to initiate an instruction fetch from that address prior to when a decision can be made. A condition code synchronization signal (CCSYNC) from the EBOX signifies that the condition code will be ready in the next physical cycle. In cycle 3, when CCSYNC is received, the ACU micromachine issues the first request of the branch target instruction stream. In the next cycle, when it receives the condition codes, it uses them to decide on whether the branch is to be taken. Because of signal delay, the decision will not be known early enough to inhibit the instruction fetch issued in cycle 3, in the case the branch is not to be taken. In that case, correction must be performed in cycle 4.

A branch taken decision(Figure 6a) means that the instruction prefetch request is correct, and additional requests can be issued. The IBOX then flushes the PREFETCHER and DECODE stages, which still hold the old instruction data, and allows the new instruction stream to be loaded and decoded.

A branch not taken decision(Figure 6b), on the other hand, causes an abort of the prefetch request initiated earlier in cycle 3 from the target address, therefore allowing the prefetcher and decoder to resume the processing of the current instruction stream. There is no penalty for branch not taken here if the current instruction stream is already in the IBUFFER, and the cost of starting a new instruction stream is also kept at a minimum. This scheme gives a simple but yet effective mechanism to handle branches.

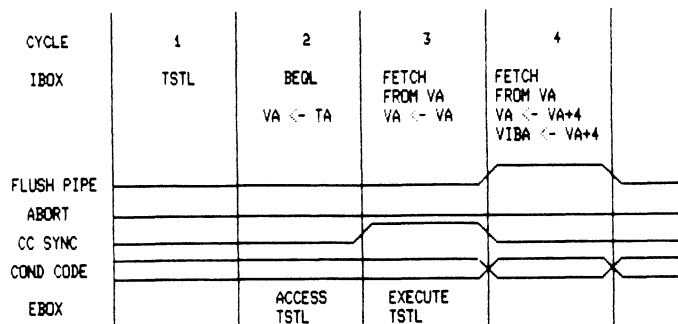


Figure 6a. Branch Instruction Taken Sequence

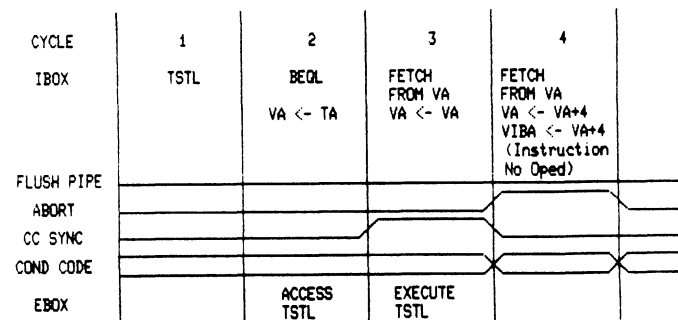


Figure 6b. Branch Not Taken Sequence

The EBOX is responsible to handle the remaining types of branches and other instructions which can alter instruction flow. This includes: CASE instructions, subroutine calls, and returns. The mechanism used is the same as that described for cold start in Section 5.1.

### 5.7 Data Dependency Resolution

The use of pipelining in the VAX 8600 IBOX requires additional mechanisms to resolve data dependency among instructions. Data dependency can happen in many situations, two key examples are:

a) Register conflicts when a source operand uses a register which is also the destination register of the previous instruction. For example, in

```

MOV L R0,R1
MOV L (R1),R2

```

Sourcing of R1 by the ACU unit in the second instruction must be inhibited until the first instruction is completed in the EBOX.

b) Memory conflicts if out of order memory access is allowed. For example, in

```

MOV L R0,(R1)
MOV L (R2),R3

```

If R1 equals R2, then operand read for the second instruction must be postponed until the write in the first instruction is issued. This also mandates additional collision detection logic exists.

The VAX 8600 IBOX uses a register scoreboard and a single operand port to resolve both problems. The scoreboard provides a simple reservation table mechanism to accomplish this resolution. The ACU unit will enter the GPR number to the scoreboard for every register destination specifier it processes. For every subsequent ACU sourcing from a GPR, the scoreboard is looked up to detect any conflict. If such a conflict exists then the sourcing operation is temporarily inhibited via a scoreboard stall. A write to the GPR by the EBOX will remove it from the scoreboard thus allowing the previously stalled sourcing operation to resume. In the VAX 8600, the scoreboard can be looked upon as a two entry associative memory structure.

Figure 7 shows an example of the functions of the scoreboard for the instruction sequence described in a).

Cycle 1: The ACU unit is processing the MOV L R0,R1 instruction. The scoreboard at this time is assumed to

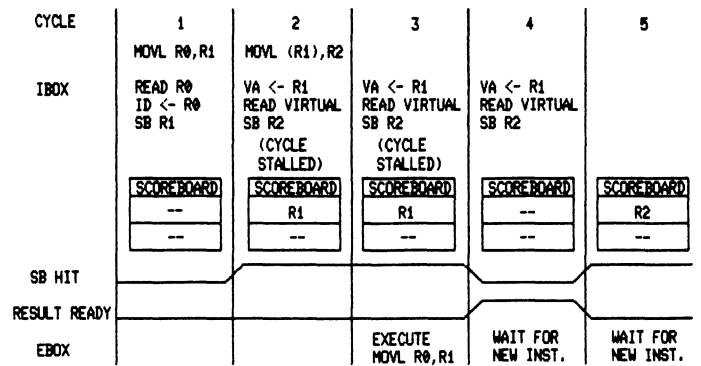


Figure 7. SCOREBOARD EXAMPLE

be empty. The ACU unit reads R0 and loads ID, the cycle is completed without problem.

Cycle 2: The scoreboard is loaded with R1. Since cycle 2 requires using R1 as address source, the IBOX control discovers that there is a scoreboard hit on R1, and the cycle can not be completed. In this case the ACU micromachine will attempt to execute the same microinstruction again next cycle.

Cycle 3: EBOX can execute the first MOV L instruction and the result is not available until the beginning of cycle 4. Just the same as in cycle 2, the ACU micromachine still stalls in cycle 3.

Cycle 4: The ACU unit can continue and finish the second MOV L instruction.

Cycle 5: The scoreboard is now loaded with R2. Similar to those earlier stalled cycles, the ACU micromachine will not be able to complete if the next instruction uses R2 in operand evaluation. In that case, the ACU micromachine stalls until write to R2 is completed.

Memory conflicts will not happen in the VAX 8600 because the ACU micromachine controls both operand read and write for most instructions via the operand port. The micromachine is sequenced in such a way that out of order memory access from IBOX is impossible.

Certain instructions whose operand address may not be known at the time of decoding (e.g. bit field instructions) will be handled by the EBOX. Operand fetch is done directly by the EBOX via the EBOX port (see Figure 2). In those instructions, the IBOX suspends itself after the completion of the address calculation of all specifiers. An IBOX suspension will prohibit any new operand fetch requests from the operand port. This prevents the potential memory conflict from occurring when IBOX attempts

to read operands for the next instruction, while the current operand result has yet to be written by the EBOX.

## 5.8 Instruction Optimizations

The IBOX generates a number of optimizations to give better performance to the CPU. For instructions using GPR as result destination, the DECODE stage will consume also the GPR specifier during the decoding of the specifier immediately before, and present only a single dispatch address to the execution unit. In addition to the source operand, the IBOX also supplies to the EBOX the destination GPR address, which the EBOX will use to access its local GPR file. This optimization essentially cuts away one dispatch to the EBOX.

Another form of optimization eliminates scoreboard stalls when the source operand is in the same GPR to be updated in the future by the previous instruction. In this case, the ACU unit will ignore the scoreboard stalling situation, and present a modified dispatch address to the EBOX signaling to this fact. The EBOX will access the correct updated GPR value from its own local copy subsequently.

## 5.9 Pipeline Stage Synchronization

As described earlier in the section on the VAX 8600 pipeline, interstage communication in the VAX 8600 is done through a number of drain signals, as well as a few global flags. Here each stage of the pipe set the valid flags of the output buffer to "full" when data is ready. The drain signal returns the fact that the buffer is going to be consumed by the successor stage. This will make the valid flag "empty". The global flags are generally broadcast to most other stages. This interlock mechanism provides the basis for the synchronization among pipe stages.

Since each stage of the pipe may take a varying number of physical cycles to complete, there are, at times, empty or full conditions in a pipe stage. An empty condition occurs in a pipe stage when it wants to drain its input buffer but it is empty. This condition will cause an input stall or idling. A full condition occurs in a pipe stage when it wants to load its output buffer but it is full. This condition will cause an output stall. Other reasons, such as resource contention, will also cause the idling and stalling condition.

Each stage uses a different scheme to handle such conditions. In both the PREFETCHER and the DECODE stages, internal

flags are maintained to indicate empty or full conditions. The PREFETCHER keeps track of the number of valid bytes in the IBUFFER and initiates a new prefetch if necessary. Data removed from the IBUFFER by the DECODE stage will decrease the number of valid bytes, whereas new prefetched data will increase the number. When the IBUFFER is full the PREFETCHER will have an output stall, i.e. no new prefetch requests will be issued. The DECODE stage loads the output buffer valid flags after each decode. It will assume an output stall if the buffer is not drained by the ACU unit. The ACU unit, in turn, can drain such buffer during its execution and clears the valid flags, thereby allowing decoding to be resumed.

The ACU micromachine contains the more complicated stalling and idling mechanism. This is where most resource contention, dependency conflicts as well as full and empty condition can occur.

There are essentially three types of stalling and idling in the ACU micromachine

1. Resource contention and busy, and dependency conflict stalls - Resource contention includes simultaneous update of GPR by the instruction and execution unit, and use of certain bus by two resources at the same time, etc. This is best exemplified by the register dependency conflict detection in the scoreboard. Another form of this kind of stalls can be resulted from memory requests not being accepted due to memory busy. A full condition which prevents any further progress of execution is also another example. In general, for this type of stalls, the micromachine will suspend the execution of the current instruction, and resume when such stall condition is removed.

2. Idling and No ops - Empty conditions happen in the ACU unit, for example, when the instruction decoder cannot provide a dispatch address due to not enough valid bytes in the IBUFFER. Another no op condition is microtraps due to unaligned data references, and flushing of the pipe. In both cases, the micromachine will execute the instruction, but none of the pertinent machine state will be modified. In the next cycle the micromachine will normally execute a fresh new instruction generated through traps or the availability of the next dispatch address.

3. Special stalls - In certain cases where the purpose of the execution is only to supply dispatches to the EBOX, the micromachine will stall to prevent modification of most of the state. A few states such as EFORK loading is still allowed. This kind of stall occurs most often for single byte instructions without any specifier. Here, a superfluous dispatch address to the ACU is generated, and should not be executed to modify any

state unintentionally. But, the EBOX dispatch must be loaded and the appropriate program counter updated.

## 6 AN EXAMPLE

In order to get a more global view of the whole process of executing a piece of code on the VAX 8600 pipeline, an example is given in this section in Fig. 8. The program segment, shown in the box in Fig. 8, employs two key mechanisms of the design: a branch and an IBOX Write. The primary purpose of this example is to show the following aspects:

a) the flow of many instructions through the pipe, including their use of the stages, units and resources.

b) the state of the pipe at any given physical cycle, snapshot-like, in order to appreciate the interaction among the various instructions active in the pipe.

According to the mechanism described in Section 5.6 and in Fig. 6a. At the beginning of cycle 8 the CMPL instruction in the EBOX sends CCSYNC to the ACU unit, which in turn issues an IBUFFER request, "ibf" in Fig. 8, from the branch target address, "TA" in Fig. 8. This request will result in the IFETCH unit fetching the INCL instruction in cycle 9. Also in cycle 9, the condition codes, "cc" in Fig. 8, computed by the CMPL instruction arrive at the ACU unit, where they determine that the branch is to be taken. The ACU then issues a "flush" command to the PREFETCHER and DECODE stages to make room for the new instruction stream. Notice that instruction execution will resume in the EBOX four physical cycles after the branch: this is a relatively small penalty for a branch, given that the pipeline latency is normally six physical cycles.

The INCL instruction which was prefetched by the branch mechanism arrives in cycle 11 in the ACU unit, where the operand effective address is loaded in the VA register. In the same cycle a memory read request is issued and the operand address is kept in VA until the EBOX is ready to do the write. The operand is fetched in cycle 12 and passed to the EBOX in cycle 13. Then the EBOX performs the increment function, sends the result to the MEM WRITE unit into the WR LATCH and issues an IBOX Write command ("ibwrite") to the ACU micromachine. This in turn issues the memory write request to the MBOX via the Operand Port, see Fig. 2. The EBOX waits two extra cycles after having issued the IBOX Write in order to handle potential memory problems, such as a page fault, before the ESA register is overwritten by retiring the instruction. Execution of the following instruction stream resumes normally in cycle 16.

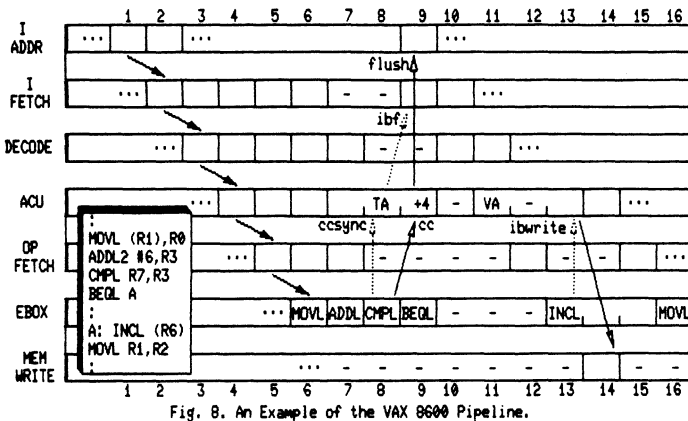


Figure 8 shows how simple instructions, such as the first three in the example, flow through the pipe in a straightforward way, using only one physical cycle per unit. All pipe units are then kept busy constantly, thus achieving the VAX 8600 peak throughput of 12.5 mips, which corresponds to the pipe executing one macroinstruction per physical cycle. Notice that in this case results are written to the GPR's, so that the MEM WRITE unit is not utilized. Also, simple memory reads do not stall the pipe, but are performed in only one cycle in the OPFETCH unit. Moreover, the ACU unit immediately starts processing the next specifier after having issued a memory read request: related memory problems, if any exist, will be handled by the EBOX.

The branch instruction which follows in the example is one in which the branch is taken. It is therefore processed

## 7 CONCLUSIONS

In this paper, the instruction and operand fetch unit (IBOX) of the VAX 8600 implementation of the VAX architecture has been described. In addition, a simplified model of pipeline implementations was introduced. In this model, a "pipeline" is described as a sequence of stages connected by a transport mechanism, which moves an item from the output buffers of a stage to its successor (i.e., a partial ordering). In connection with this model, the crucial issues in designing a pipeline were discussed in reference to a specific implementation, that of the VAX 8600 and its IBOX. The most important of such issues are:

1. the hand-off of items from one stage to the next, that is, the issue of local vs. global control,

2. buffering, which relates to the number of items within a stage,
3. contention for resources, and the associated stall conditions, and
4. dependency of one stage on the activity of another stage (e.g. forward and backwards dependencies)

The significance of this implementation of the VAX architecture, and of the design presented here, lies in the successful resolution of the complex design problems, which occur in the pipeline implementation of modern architectures, such as the VAX-11. Specifically, the use of register scoreboard to prevent the use of stale register data, a facility to recover in the presence of exceptions, and synchronization mechanisms to deal with VAX-11 specifics, such as unaligned references, can be considered a major accomplishment. The capabilities of this design, i.e. a four times speed improvement over the VAX 11/780, and under favorable conditions, the ability of the IBOX to deliver to the EBOX one instruction every 80 nsecs, which means a peak execution rate of 12.5 mips, certainly make the VAX 8600 a major engineering achievement.

#### Acknowledgment

A project of this magnitude requires diligent efforts from a large number of people, from the architects and designers, to technicians. From different support personnel in CAD to manufacturing. To name all the people actively involved in this project would need many pages. However, the authors are particularly grateful to the following people: Bob Glorioso, Jud Leonard, Al Helenius, Clem Liu, John Derosa, Tom Knight, Pete Rado, Rich Glackemeyer, Albert Yu, Elaine Hanson, Frank McKeen, Tryggve Fossum, Bill Bruckert, and Jim Lacy.

#### References:

- [1] Digital Equipment Corp. 1981. "VAX Architecture Handbook," Digital Equipment Corporation, Maynard, Massachusetts.
- [2] Bruckert, W., Quaynor, N., Colon Osorio, F.C., 1985 "The Virtual Memory And Cache Unit Of The VAX 8600," Internal Technical Report, Digital Equipment Corporation, Marlboro, Massachusetts.
- [3] Block, E. 1959. "The Engineering Design of the STRETCH Computer", Proc. EJCC, pp.48-59.
- [4] Anderson, D.W., Sparacio, F.J., and Tomasulo, R.M. 1967. "The IBM System/360 Model 91: Machine Philosophy and Instruction Handling", IBM J. Res. Dev., January, pp. 8-24.
- [5] Hintz, R.G., and Tate, D.P. 1972. "Control Data STAR-100 Processor Design", Proc. COMPCON, IEEE No. 72CH0659-3C, pp.1-4.
- [6] Cray Research Inc. 1976. "CRAY-1 Computer System, No. 2240004, Cray Research Inc., Bloomington, Minn.
- [7] Kogge, Peter M. 1981. "The Architecture of Pipelined Computers," McGraw-Hill, New York.
- [8] Siewiorek, D., Bell, C.G., Newell, A. 1982. "Computer Structures: Principles and Examples," McGraw-Hill, New York.
- [9] Lampson, B.W., McDaniel, G.A., Ornstein, S.M. 1981. "An Instruction Fetch Unit For A High-Performance Personal Computer," XEROX Palo Alto Research Center, Palo Alto, California.

IN SEARCH OF THE VAXINTOSH  
CUSTOMIZING VMS V4.0 FOR DCL WINDOWS

James G. Downward  
KMS Fusion, Inc.  
Ann Arbor, Michigan 48106-1567

ABSTRACT

Inevitably, while in the editor, testing a program, or performing some task requiring the deepest concentration, an interruption occurs forcing one to terminate the current activity and start up another. This causes significant loss of time and efficiency. To ameliorate this situation, a simple method of adding windows to VMS is presented which allows single keystroke entry to and exit from DCL windows. These windows may be accessed from DCL, command procedures, MAIL, and three DEC video editors.

INTRODUCTION

Over the years, a hitherto unmentioned law governing the art of programming has become increasingly obvious to me. Namely,

During normal working hours,  
interruptions are the rule,  
not the exception.

Inevitably, while in the editor, or testing a program, or performing some task requiring the deepest concentration, an interruption occurs. The interruption may take the form of a phone call, a direct confrontation with an irate user, new VAX mail, a brilliant idea which just must be tried out, or perhaps yet another fire to be put out. Prior to VMS V4.0, there was little one could do to ameliorate this problem. VMS V4.0, however, has a number of features which may be used to help minimize the disruptive effects of these interruptions. Specifically, DCL "windows" can be implemented which can be used to manage each interruption without losing the context of the previous activity.

This paper will describe methods of implementing such windows for

1. DCL AND MAIL
2. Command Procedures
3. The VMS V4.0 editors EDT, LSE and TPU.

While VMS and a number of utilities have the SPAWN command available to the user, the window interface to be described has been found in practice to be simpler to use because,

1. Windows are consistently invoked with a single key command.
2. Confusion is minimized because one always knows if one is in a window.

THE DCL WINDOW INTERFACE

Ideally, each user would have a "VAXintosh"-like terminal. By pressing a single key, work in progress would be interrupted, the application context and screen saved, and a fresh, full function window similar to what one might see on a

Macintosh or VAXStation would appear. The user could switch back and forth between windows without losing context, move them around and resize them to meet the particular application needs, and finally close a window returning the the initial work in-progress when the interruption occurred.

Unfortunately, standard DEC VT100/VT200 terminals cannot support such bit-mapped graphic windows and until either VMS has real virtual terminals or all VMS utilities and RMS use the SMG terminal I/O interface, process-wide, overlapping, cellular-text windows are also hard to implement.

Consequently, the various "window" implementations to be discussed here are designed to meet a more limited set of goals, namely:

1. Ease of implementation.
2. Single keystroke Window activation from DCL or application prompt.
3. Normal user/DCL interaction.
4. Consistent user interface for windows activated from DCL, command procedures, editors, MAIL, etc.
5. Window display identifies current window in use.
6. Simple return to the parent process by typing Control-Z.
7. Simple access to sequentially nested windows.

To meet these goals, DCL "windows" are implemented as subprocesses running a command dispatcher procedure which sets up the terminal, prompts the user for input, issues the command to DCL and then loops back to prompt for more input. Several types of window procedures are possible depending on what is considered to be an acceptable response time.

The Basic Window Procedure

The window procedure shown in Listing 1 is fast, simple to implement, and can be adapted to work with hardcopy terminals. It reminds the users



that a window is active via the prompt string. When invoked it first executes the SETUP commands which are located at the end of the command procedure to optimize response time. During the SETUP section, the VT100 screen is erased and the process name is obtained to include as part of the prompt string. After the initialization is completed, the user is prompted for input and that input is sent directly to DCL. If the user types Control-Z on input, the command procedure exits back to the parent process. Note that throughout the command procedure, extreme care is taken to insure that SYS\$INPUT is always correctly pointing at the user's terminal. A second item to note are the lines starting with "!\*\*\*". These lines should be uncommented if the window procedure is to be used with EDT or any application image which can keep an active subprocess available for possible re-use until the image exits.

#### The Window With Banner Procedure

The window procedure shown in Listing 2 works with VT100 compatible terminals. At its core, this procedure is quite similar to the first procedure. However, it uses numerous VT100 escape sequences to display and maintain the window banner at the top of the terminal screen. Specifically, it

1. Establishes a scroll region below the window banner so that the banner will not be overwritten with as each new command is entered.
2. Uses the ASK utility program (read with timeout) to request cursor location from the VTxx terminal.
3. Restores the cursor to the correct line and refreshes the window banner after each command is processed.
4. Saves and restores screen attributes between commands.
5. Establishes an internal typeahead buffer so that user input can be separated from cursor location requests.

This procedure uses two non-standard foreign commands, PAGE and ASK. The ASK command invokes the ASK image and behaves similarly to INQUIRE except that it can perform a read with timeout and can read escape sequences from a terminal. The PAGE command is a command procedure which erases screens of VT1xx and VT2xx terminals. Besides erasing ANSI terminals, it knows how to erase the graphic overlays of ReGIS graphic terminals such as the VT125. Both ASK and PAGE are on recent VAX SIG Symposia tapes.

By extending the second procedure, it is possible to implement even more elaborate window procedures. For example, for VT240 terminals, a task to save the screen contents and cursor location to a SIXEL file prior to exiting a window could be added. On return to the window, the screen could be restored to its previous state and the cursor repositioned to its previous position. However, the save and restore screen operations significantly slow down window creation and deletion times.

#### Limitations of the Window Procedures

While using a procedure to emulate a DCL window provides the user with a nearly normal terminal environment, user commands with embedded single quotes will not work correctly. In practice, this limitation has not been found to be a serious problem.

A second limitation is that it is assumed that each window is entered and deleted on a "last created, first deleted" basis and that as each window is deleted, one returns to the "parent" of that window. However, using the ATTACH command it is possible for the experienced user to "jump" between multiple open windows. If this is done, however, it is possible for things to get very confused, and exiting a low level window back to its parent can either delete an active subprocess of the window or worse, leave the subprocess dangling unaccessed in the system.

#### ADDING WINDOWS TO DCL AND MAIL

Once one has developed a window procedure, providing single keystroke access to DCL is very straightforward by using the DEFINE KEY command.

```
$ DEFINE/KEY/NOLOG/NOECHO/TERMINATE F20
"SPAWN/NOLOG @DCLWINDOW"
```

Unfortunately, one must either be at the DCL prompt or interrupt the currently executing image in order to start up the new window because the DEFINE KEY facility does not allow multiple commands to be embedded in the key definition. However, for VT2xx terminals one can define a UDK (User defined key) to contain both a Control-Y and the above SPAWN command. The UDK is then activated by pressing SHIFT-F20. The drawbacks of this method are that creating a UDK load module is somewhat tedious (it must be written in HEX) and that one must remember to issue the CONTINUE command to resume the interrupted image immediately after exiting the window.

In a similar fashion, this facility can be added to MAIL by creating a file SYS\$SHARE:MAIL\$KEYDEF.INI with the command

```
DEFINE/KEY F20/TERMINATE/NOECHO "SPAWN @DCLWINDOW"
```

and defining

```
$ DEFINE MAIL$INIT SYS$SHARE:MAIL$KEYDEF.INI
```

in the system wide LOGIN.COM file.

#### ADDING WINDOWS TO COMMAND PROCEDURES

It is simple to create command procedures which allow an exit to a new window at any prompt by using the ASK command (mentioned above). In the following command procedure code fragment, the ASK command is used to prompt the user for a choice. If the user presses the F20 key, or any key generating an escape sequence, the escape sequence is checked to see if it is a command to create a new window.

```

$Start:
... ..
$ ASK/UPPER Choice "Choice: " ! Get all of input line
$ Option[0,1]='Choice' ! Isolate 1st character
$ IF Option .EQS. "<ESC>" THEN - ! If any escape sequence present
 GOTO Check_Escape_Seq ! go find which one
... ..
$Check_Escape_Seq:
$ IF Choice .EQS. "<ESC>[34 " THEN - ! If F20 key pressed
 GOTO New_Window
... ..

$New_Window:
$ SPAWN/NOLOG @DCLWINDOW" ! A fresh window, please
$ PAGE ! Erase the screen (if needed)
$ GOTO Start ! Go back to where we started

```

#### ADDING WINDOWS TO EDT, LSE, AND TPU

Using the basic procedures discussed above, it is also possible to add windows to DEC's video editors, EDT, LSE, and TPU. Since adding windows to EDT is the more difficult than adding them to LSE/TPU, the required EDT changes will be discussed first.

#### Modifying EDT V3.0 for DCL Windows

Adding windows to EDT requires that a new version of EDT be built. This involves creating a new EDT mainline to parse the DCL command line to set filenames and flags, establish a user XLATE subroutine and exit handler, and invoke the callable EDT subroutines in SYS\$SHARE:EDTSHR.

The version of EDT to be discussed here, VPWEDIT, behaves identically with EDT V3.0 except that it provides access to DCL windows. Two new EDT features allowed VPWEDIT to be developed. First, EDT is provided as a sharable image which may be called as a subroutine from within a user program. This functionality is described in Appendix D of the RSX EDT V3.0 reference manual or in the VAX/VMS Utility Routines Reference Manual. Using the callable EDT interface allows one to create a program which exactly mimics functionality of the EDT editor. Second, EDT provides the NOKEYPAD XLATE command which allows the user to pass command strings to a user-specified subroutine.

Creating the VPWEDIT is straightforward. A mainline program extracts the switches and files specified on the DCL command line and calls EDT specifying an action subroutine, WINDOW, to call if the XLATE command is used. The WINDOW action routine establishes an exit handler to keep track of the current subprocess in use so that it can be deleted when the editor exits. The WINDOW action routine creates or attaches subprocesses as needed to provide the DCL window. In the following sections, these program modules will be discussed in greater detail.

#### The Editor Mainline

The VPWEDIT editor is designed to be an exact replacement for EDT. This replacement is

transparently accomplished by inserting a command of the form

```
$ DEFINE EDT SYS$SYSTEM:VPWEDIT
```

in the system-wide login command procedure. Once this is done, DCL will invoke the VPWEDIT image rather than the EDT image in response to the EDIT command. The only function of the editor mainline is to find what switches and files are on the DCL EDT command line. The code for VPWEDIT is shown in Listing 3. The example here was written in BASIC but it could just as easily have been implemented in any other supported VAX language. BASIC was chosen because it handles variable length strings automatically. For efficiency, VPWEDIT should be installed as /OPEN/SHARE.

#### The WINDOW Subroutine

When the NOKEYPAD XLATE command is processed, the subroutine specified in the EDT\$EDIT call for handling the XLATE command is called and any text associated with the XLATE command is passed to the function subroutine. Since the XLATE command is not directly accessible when using the keypad editor, the window is invoked by control key functions which should be defined in the system wide EDT initialization file, SYS\$SHARE:EDTSYS.EDT.

```
DEFINE KEY FUNCTION 34 AS "XLATEWINDOW Z."
DEFINE KEY GOLD V AS "XLATEWINDOW Z."
```

The key definitions in this example will cause the WINDOW subroutine to be invoked if either the F20 key on a VT2xx keyboard or GOLD V on a VT100 keyboard is pressed.

Since creating a subprocess is time consuming, the WINDOW function subroutine (Listing 4) keeps track of whether or not one is available for use. If no subprocess exists, it spawns a subprocess and invokes WINDOW.COM. WINDOW.COM is identical to either of the two previously discussed versions of DCLWINDOW.COM except that the lines with "!" have been uncommented.

The function of WINDOW.COM is to prompt the user for input, pass that input on to DCL, and to attach back to the main process if a Control-Z is entered. When DCLWINDOW.COM is first invoked, it acquires the PID of the parent process so that it

can be attached back to. If, however, the subprocess already exists, the WINDOW function subroutine simply attaches back to the subprocess running DCLWINDOW.COM, which again accepts the user's input and passes it to DCL. In the event that attaching to the subprocess fails because the subprocess was deleted, the WINDOW function subroutine tries to create a new subprocess.

The WINDOW function calls the subroutine, EXIT SPAWN. The EXIT SPAWN routine (Listing 5) is used to insure that any subprocesses created are deleted when the image exits.

#### MODIFYING LSE/TPU TO SUPPORT DCL WINDOWS

Modifying LSE/TPU to support windows is very simple since LSE/TPU have a SPAWN command Built-In. For example, to implement DCL windows for LSE it is only necessary to load a TPU procedure, WINDOW.TPU. By redefining the LSE command,

```
$ LSE*DIT:=LSE/COMMAND=WINDOW.TPU
```

every time LSE is invoked, pressing the F20 key on a VT2xx terminal will create a DCL window. The TPU

procedure file required to do this would be

```
PROCEDURE NEW WINDOW
 SPAWN ("@DCLWINDOW");
ENDPROCEDURE
DEFINE_KEY ('NEW_WINDOW',F20);
```

#### CONCLUSIONS

DCL windows accessed from command files and VPWEDIT have been in use at KMS Fusion for nearly two years. Experience has shown that their use has significantly aided productivity. In time it is hoped that the VMS Screen Management Package will allow for more sophisticated displays of multiple, overlapped windows.

The code for VPWEDIT is part of the KMSKIT submission to the Spring 1984 VAX Sig Decus Symposium tape.

#### ACKNOWLEDGMENTS

This work was performed under DOE Contract DE-AC08-DP-40152.

```
$ Vfy:='F$VERIFY(0)'
$! DCLWINDOW.COM
$THE_START: ! Once only code at end
$ GOTO Setup ! Return to Start
$START: !
$ READ/PROMPT="'Prc'> "/END=DONE SYS$COMMAND Cmdline
$ 'Cmdline'
$Start_1:
$ IF Tmp .NES. "'F$LOGICAL("SYS$INPUT")'" THEN - ! Restore SYS$INPUT if
 ASSIGN/USER/NOLOG SYS$COMMAND SYS$INPUT ! needed
$ GOTO START ! Get next command
$!
$Done: ! Got Z, LOGOUT, etc.
$ WRITE SYS$OUTPUT "<ESC>[m<ESC>[2J" ! Erase VT100 screen
$! Uncomment the next lines for use with a Parent process which knows the
$! WINDOW is open and can ATTACH to its child.
$!*** SET MESSAGE/NOTEXT/NOFACILITY/NOIDENT/NOSEVERITY! Disable attach message

$!*** ATTACH/IDE='Parent' ! Back to the main proc.

$!*** SET MESSAGE/TEXT/FACILITY/IDENT/SEVERITY ! Error messages again
$!*** GOTO Setup ! Awake again, do setup
$ EXIT ! Done with window
$Setup:
$ Parent=F$GETJPI("", "OWNER") ! Establish our parent
$ ON CONTROL Y THEN GOTO Start_1 ! Initialize first time
$ ON ERROR THEN GOTO Start_1 ! only
$ WRITE SYS$OUTPUT "<ESC>[m<ESC>[H<ESC>[2J" ! Erase VT1xx screen
$ IF .NOT. F$GETDVI("SYS$INPUT", "TRM") THEN - ! Insure we point at
 ASSIGN/USER/NOLOG SYS$COMMAND SYS$INPUT ! physical terminal
$ TMP:='F$LOGICAL("SYS$INPUT")' ! Double check in case
$ IF Tmp .NES. "'F$LOGICAL("SYS$INPUT")'" THEN - ! SYS$INPUT is LOGICAL
 ASSIGN/USER/NOLOG SYS$COMMAND SYS$INPUT ! not real terminal
$ Prc="'F$PROCESS()'" ! Use with input prompt
$ ON ERROR THEN GOTO Start ! Start over on error
$ GOTO Start
```

Listing 1. Basic command dispatcher

```

$ Vfy:='F$VERIFY(0)'
$ GOTO Setup ! Initialize
$START:
$ ON Control_Y THEN GOTO Start_0 ! Reinit on Y
$ ASK Cmdline "<ESC>[10D<ESC>[2K$ 'Ovflow'" ! Get new command
$ IF Cmdline .EQS. " Z" THEN GOTO Done ! If Z, just exit
$ IF Cmdline .EQS. "<ESC>[3 " THEN Ovflow="" ! If catch cursor posit
$ IF Cmdline .EQS. "<ESC>[3 " THEN Cmdline="" ! update, Null command
$ Cmdline=Ovflow+Cmdline ! Tack on any leftover
$ L=F$LOCATE("<ESC>",Cmdline) ! Any escape seq?
$ IF L .LT. F$LENGTH(Cmdline) THEN Cmdline=F$EXTRACT(0,L,Cmdline)
$ IF Tmp .NES. "'F$LOGICAL("SYS$INPUT")'" THEN - ! Be sure to point at
$ ASSIGN/NOLOG/USER SYS$COMMAND SYS$INPUT ! real terminal
$ 'Cmdline'
$ Ovflow="" ! Null out any overflow
$Start_0:
$ ON Control_Y THEN GOTO Done ! If Y while here, exit
$
$ WRITE SYS$OUTPUT "'Header'" ! Refresh banner
$ IF .NOT. F$GETDVI("SYS$INPUT","TRM") THEN - ! Make sure is terminal
$ ASSIGN/NOLOG/USER SYS$COMMAND SYS$INPUT !
$ ASK tmp "<ESC>[6n" ! get screen position
$ l=f$length(tmp) ! may have typeahead
$ Istrt=F$locate("<ESC>[",tmp) ! trapped with position
$
$ Ibeg=Istrt+2 ! info, so strip out
$ IF Istrt .EQ. L THEN Istrt=F$LOCATE("[",Tmp) ! the escape sequence
$ IF Istrt .EQ. L THEN Ibeg=Istrt+1 ! (find start and end)
$ Iend =F$locate("R",F$EXTRACT(Istrt,l-Istrt,Tmp))! and place rest of
$ Tmp2="'F$EXTRACT(Ibeg,Iend,Tmp)'" ! command in overflow
$ ovflow=Tmp-<ESC>-"["'Tmp2'" ! buffer
$ IF Tmp2 .EQS. "2;1R" THEN WRITE SYS$OUTPUT "" ! If at line 2, <cr><lf>
$
$Start_1:
$ IF Tmp .NES. "'F$LOGICAL("SYS$INPUT")'" THEN - ! Point SYS$INPUT at
$ ASSIGN/NOLOG/USER SYS$COMMAND SYS$INPUT ! terminal
$ X:='F$VERIFY(0)' ! Turn off logging
$ GOTO START ! get more info
$Done:
$ WRITE SYS$OUTPUT "<ESC>7<ESC>[1;24r<ESC>8<ESC>[1A"
$ PAGE !
$ Vfy:='F$VERIFY(Vfy)' !
$! Uncomment the next lines for use with a Parent process which knows the
$! WINDOW is open and can ATTACH to its child.
$!*** SET MESSAGE/NOTEXT/NOFACILITY/NOIDENT/NOSEVERITY! Disable attach message
$!*** ATTACH/IDE='Parent' ! Back to the main proc.
$!*** SET MESSAGE/TEXT/FACILITY/IDENT/SEVERITY ! Error messages again
$!*** GOTO Setup ! Awake again, re-setup
$ EXIT
$Setup:
$ Parent=F$GETJPI("", "OWNER") ! Establish our parent
$ ON CONTROL_Y THEN GOTO Done ! xit if Y here
$ HELP:=HELP/NOPAGE ! Redefine symbols
$ STO*P:=LOGOUT ! as needed by screen
$ ON ERROR THEN GOTO Start_1 !
$ WRITE SYS$OUTPUT -
$
$ "<ESC>[1;24r<ESC>[?8h<ESC>[0m<ESC>>" !
$ IF .NOT. F$GETDVI("SYS$INPUT","TRM") THEN - ! Be sure pointing at
$
$ ASSIGN/NOLOG/USER SYS$COMMAND SYS$INPUT ! terminal
$ TMP:='F$LOGICAL("SYS$INPUT")' !
$ Prc="'F$PROCESS()'" ! Get process name
$ Len=F$LENGTH(Prc) ! and stick in middle
$ N=(80-Len)/2 ! of the banner
$ Ovflow="" !
$ PAGE ! erase screen
$ Header1="<ESC>[?6]<ESC>[;'N'H<ESC>[1m'F$PROCESS()'<ESC>[m"+
$ "<ESC>[;64H(CTRL/Z -> Exit)"
$ Header2="<ESC>[A<ESC>7<ESC>[?61<ESC>[H<ESC>[K<ESC>[;'N'H"+
$ "<ESC>[1m'F$PROCESS()'<ESC>[m<ESC>[;64H(CTRL/Z -> Exit)"

```

```

$ header3="<ESC>[2;H<ESC>)0 Nssssssssssssssssssssssssssssssssssss"+
"ss 0<ESC>[3;24r<ESC>[3;H"
$ Header4="<ESC>[2;H<ESC>)0 Nssssssssssssssssssssssssssssssssssss"+
"ss 0<ESC>[3;24r<ESC>8"
$ Header="''Header2''Header4'' ! Display header
$ WRITE SYS$OUTPUT ""Header1''Header3'' ! (both parts)
$ IF Tmp .NES. ""'F$LOGICAL("SYS$INPUT")"" THEN - ! be sure pointing at
$ ASSIGN/NOLOG/USER SYS$COMMAND SYS$INPUT ! real terminal
$ IF P1 .EQS. "" THEN GOTO START ! if input line not
$ 'P1' 'P2' 'P3' 'P4' 'P5' 'P7' 'P8' ! NULL, repeat
$ IF Tmp .NES. ""'F$LOGICAL("SYS$INPUT")"" THEN - !
$ ASSIGN/NOLOG/USER SYS$COMMAND SYS$INPUT !
$ X:='F$VERIFY(0)' !
$ GOTO Start ! back for more

```

Listing 2. Command dispatcher with window banner

```

1 REM
! VPWEDIT.BAS
!
! Mainline for the EDT editor with DCL windows. The editor can
! be built by:
! $ BAS VPWEDIT
! $ BAS WINDOW
! $ FOR EXITSPWN
! $ LINK/NOTRACE VPWEDIT,WINDOW,EXITSPWN,SYS$SHARE:EDTSHR/SHARE
!
2 EXTERNAL INTEGER FUNCTION CLI$PRESENT, CLI$GET VALUE
EXTERNAL INTEGER CONSTANT CLI$_PRESENT, CLI$_DEFAULTED
EXTERNAL INTEGER CONSTANT CLI$_NEGATED, CLI$_ABSENT
EXTERNAL INTEGER CONSTANT EDT$_RECOVER, EDT$_NOCOMMAND
EXTERNAL INTEGER CONSTANT EDT$_NOJOURNAL, EDT$_NOOUTPUT
EXTERNAL INTEGER CONSTANT EDT$_NOCREATE

EXTERNAL INTEGER EDT$_FILEIO ! Callable EDT subroutines
EXTERNAL INTEGER EDT$_WORKIO !
EXTERNAL INTEGER WINDOW ! XLATE subroutine to invoke
EXTERNAL INTEGER FUNCTION EDT$_EDIT ! Callable EDT
DECLARE INTEGER RESULT
DIM INTEGER PASSFILE(1%)
DIM INTEGER PASSWORK(1%)
DIM INTEGER PASSXLATE(1%)

PASSFILE(0%)=LOC(EDT$_FILEIO) ! Pass file names to subroutine
PASSWORK(0%)=LOC(EDT$_WORKIO)
PASSXLATE(0%)=LOC(WINDOW) ! Pass name of subroutine
DECLARE LONG Ret Status
DECLARE LONG Options ! Sum of all DCL options

Input_File$="" ! Initialize file names
Output_File$="" !
Journal_File$="" !
Command_File$="" !
ON ERROR GOTO 1000 ! Exit if disaster strikes

10 IF CLI$PRESENT('INPUT') AND 1% THEN ! Find what DCL asked for
CALL CLI$GET_VALUE('INPUT',Input_File$)
END IF

20 IF CLI$PRESENT('RECOVER') AND 1% THEN ! If recover switch seen
OPTIONS=OPTIONS+EDT$_RECOVER ! set in option word
END IF

30 IF CLI$PRESENT('READ_ONLY') AND 1% THEN ! If READ_ONLY switch seen
OPTIONS=OPTIONS+EDT$_NOOUTPUT ! set in option word
END IF

```

```

50 Ret Status=CLI$PRESENT('COMMAND') ! If COMMAND switch seen
 IF (Ret Status = CLI$ Present) THEN ! get the command file
 CALL CLI$GET_VALUE('COMMAND',Command file$)
 ELSE IF (Ret Status = CLI$ DEFAULTED) THEN
 REM Use default command file !
 ELSE IF (Ret Status = CLI$ _NEGATED) THEN! If NOCOMMAND seen
 Command File$="" !
 OPTIONS=OPTIONS+EDT$M_NOCOMMAND ! set in option word
 END IF

60 IF CLI$PRESENT('OUTPUT') AND 1% THEN ! If output specified
 CALL CLI$GET_VALUE('OUTPUT',Output File$)
 ELSE
 REM Default to output=input !
 END IF

70 Ret Status=CLI$PRESENT('JOURNAL') ! If JOURNAL Seen
 IF Ret Status = CLI$ Present THEN
 CALL CLI$GET_VALUE('JOURNAL',Journal File$)
 ELSE IF Ret Status = CLI$ DEFAULTED THEN
 REM Use default journal file !
 ELSE
 OPTIONS=OPTIONS+EDT$M_NOJOURNAL ! Else show NOJOURNAL
 END IF

80 Ret Status=CLI$PRESENT('CREATE') ! If NOCREATE switch seen
 IF Ret Status = CLI$ Negated THEN
 OPTIONS=OPTIONS+EDT$M_NOCREATE ! set in options word
 END IF

90 Result=EDT$EDIT(Input File$,Output File$, & ! Call EDT
 Command File$,Journal File$,Options, & ! passing file names
 PASSFILE(0%)BY REF,PASSWORK(0%)BY REF, & ! and options
 PASSXLATE(0%)BY REF) ! and xlate subroutine

100 IF (RESULT AND 1%) =0% THEN
 PRINT "VPWEDIT -- Call to EDT$EDIT failed, GET HELP"
 CALL LIB$STOP(RESULT BY VALUE)
END IF
GOTO 32000
1000 RESUME 32000
32000 END

```

Listing 3. VPWEDIT Editor Mainline

```

1 REM
! WINDOW.BAS
! This subroutine is invoked via the XLATE command and the user command
! string, CMD$, is passed to it. The command string is established with
! the EDTSYS.EDT initialization file.
!
2 FUNCTION INTEGER WINDOW(CMD$)
 DECLARE INTEGER SUBPROC
 DECLARE LONG Istatus
 EXTERNAL INTEGER FUNCTION LIB$ATTACH
 COMMON (PIDVAL) LONG Sub_Pid
 IF CMD$="" THEN ! If null, return
 WINDOW=1 ! but show success
 GOTO 100 !
 END IF
20 IF CMD$="WINDOW" THEN ! If "window" wanted
 IF Sub_PID = 0 THEN ! If no window yet
 Command$=' $ @WINDOW' ! spawn a window
 CALL LIB$SPAWN(Command$,,'TT:',,,Sub_pid,SUBPROC)!with return addr
 CALL EXIT_Spawn(Sub_Pid) ! Establish exit handler
 ELSE ! Else if window exist
s Istatus = LIB$ATTACH(Sub_PID BY REF) ! just attach to it
 IF Istatus = 2280 THEN ! but if it went
 Sub_PID=0 ! away, show it
 GOTO 20 ! and make a new one

```

```

 END IF ! END IF
 END IF ! END IF
 CMD$='REF ' ! Repaint screen on
 END IF ! return to EDT
 SUBPROC=1 ! Always show success
90 WINDOW=SUBPROC
100 FUNCTIONEND

```

Listing 4. The WINDOW Function Subroutine

```

C+
C Here we establish an exit handler to insure that any subprocess is deleted
C on exit.
C CALL Exit_Spawn(Sub_PID)
C Where
C Sub_PID I*4 PID for created subprocess
C-
SUBROUTINE EXIT_SPAWN(Sub_PID)
IMPLICIT INTEGER*4 (A - Z) !
LOGICAL Is_Set ! Routine not yet called

INTEGER*4 EXIT_STATUS, EXIT_BLOCK(5) !
EXTERNAL EXIT_SPAWN2 ! Call on exit
COMMON /PID/ PID,Is_Set !
DATA Is_Set /.FALSE./

C Declare exit handler
EXIT_BLOCK(2) = %LOC(EXIT_Spawn2) !
EXIT_BLOCK(3) = 1 ! Transmit one argument
EXIT_BLOCK(4) = %LOC(EXIT_STATUS) !
PID=Sub_PID !
IF (Is_Set) RETURN ! Only declare once
STATUS = SYS$DCLEXH (EXIT_BLOCK) ! Requires Implicit
Is_Set=.TRUE. ! Light a candle
IF (.NOT. STATUS) CALL LIB$STOP (%VAL(STATUS)) ! INTEGER*4 (A-Z)
RETURN ! Back to mainline
END

SUBROUTINE EXIT_Spawn2(EXIT_STATUS)

C .. This routine is called when the program tries to exit.
C .. Its function is to delete the current active subprocess (if any)
C
IMPLICIT INTEGER*4 (A - Z) !
LOGICAL Is_Set
COMMON /PID/ PID,Is_Set

IF (PID.NE.0) Istatus=SYS$DELPRC(PID,)
RETURN
END

```

Listing 5. The VPWEDIT Exit Handler

# DESIGNING RELIABILITY INTO THE VAX 8600 SYSTEM

BY

William Bruckert and Ron Josephson

Digital Equipment Corporation  
Marlboro, Mass.

## [ABSTRACT]

The failure rate of a system is directly related to the number of components used in its design. Therefore, the designers of a large CPU must put emphasis on fault avoidance, fault tolerance, and fault minimization to ensure that the overall system failure rate is acceptable. The VAX 8600 system contains many features to assure its reliability. Conventional approaches, such as parity checking, and non-conventional ones, such as array address checking through ECC codes, were used to overcome the higher failure rate generated by having more components. This paper will cover the most important steps that were taken to provide that reliability.

The cost of a failure is proportional to the size of a system, since more compute power is lost and more people are idled as size increases. Since the failure rate is directly related to the number of components in the system, a much greater emphasis must be placed on fault tolerant designs in larger systems in order to keep the costs of failures at an acceptable level [1]. The VAX 8600 system is the largest, most powerful computer produced by Digital Equipment Corporation. We made customer satisfaction the most important engineering goal, thereby placing a high priority on the machine's reliability.

Reliability can be subdivided into four areas: fault avoidance, fault tolerance, fault minimization and improved mean time to repair (MTTR). Fault avoidance is realized by reducing the system failure rate through improved quality of the components, interconnects, design and manufacturing. Fault tolerance is the negation of the effects of faults through correction codes, redundant hardware, reconfiguration, and retry [2]. Fault minimization is the reduction of the effects of a fault by tagging corrupted data that has damaged the machine state or other data. Fault minimization is also achieved by having the hardware give accurate and detailed fault information. The MTTR is improved through remote diagnosing, the reduction of the time to diagnose a fault, and the increase of diagnostic accuracy. The application of each of these four areas to the VAX 8600

design will be discussed in detail in the following paragraphs.

Before these details are presented, however, a short explanation of the major parts of the 8600 architecture is warranted. The components in the VAX 8600 CPU are contained in four "boxes" that control operations and perform various functions. The E Box executes and retires instructions. The I Box prefetches and decodes instructions and prefetches operands. The M Box performs page translation, cache functions, I/O transfers, and memory array access. And the F Box performs floating point operations.

## FAULT AVOIDANCE

Our first goal in designing a reliable system was to reduce the number of failures that occur in the machine. This involved getting components, interconnects, and power systems with the lowest failure rates. Reducing the failure rates also involved constantly monitoring the failures that were experienced and determining their causes.

A major influence on the IC reliability was exercised by specifying how the chips were to be stressed and tested. The dips and the macrocell arrays (MCAs) were required to be burned in before testing. Thereafter all chips were to be functionally tested. However, in debugging the early machines we discovered bad dips. We had expected to find only a



handful of bad chips since they were all burned in. To identify the cause of these failures, all defective chips were analyzed, and the problem was identified as static that was "zapping" our modules. Subsequently, the design was changed so that all machines come with static grounding straps.

We also examined the designs of previous CPUs to determine which problem areas were typical. The backplane is an example. Wire-wrapped backplanes are difficult to build and test. They have several failure modes--such as cold flow of the insulation, a nicked wire, and scraps of wire. They can also be damaged during servicing of the machine. All these problems often result in intermittent faults that slowly but surely become more solid. Improving the quality control on the wire-wrapping process to obtain the desired reliability was a very difficult task, since the process is comprised of a large number of repetitive but not identical operations. Moreover, a very small error rate still produces quite a large overall failure rate. Therefore, early in the project, we decided to replace the wire-wrapped backplane with a multi-layer printed circuit card, which has a much lower failure rate.

In the power subsystem, fault avoidance was pursued by improving the ac input-power tolerance, the design testing, the manufacturing processes, and the environmental monitoring. In particular, manufacturing was a key area in which the reliability of the power supplies was improved. A new power supply tester was developed to improve our testing capabilities. It contains logic that can fully test the characteristics of a power supply and store the test data. The data includes line and load regulation and noise measurements.

A modular power supply (MPS) was designed to run from a single clock so that all regulators would be in synchronization. This synchronization allowed us to predict and control the output noise of the switching regulators. A new high-current connector was also developed that allows the regulators to be pluggable.

The power subsystem also contains the environmental monitoring module (EMM). The EMM was designed to monitor the status of the power supply and the environment inside the system. The EMM can measure the voltage output of every regulator, the inlet and outlet air temperatures, the air-flow velocity, and the ground-wire current in the primary power cord. The system protects itself by having the EMM monitor these conditions, log any deviations, and shut down the system if adverse conditions warrant it.

According to E.J. McCluskey, "Improper design of the hardware or software can result in a system which does not function at all. Such mistakes are, of course, quickly discovered and corrected. Other, less obvious design defects usually remain in any system even after it has been in service for a long time." [3] The results of design problems are logic circuits that either fail prematurely or sense signals falsely. The number of these types of errors is indirectly a measure of the quality of the tools used in the system's design.

At the beginning of a design project, rules are established to make sure that the goals for signal integrity and component failure rates can be achieved. It is usually impossible to develop rules that are both easy to check and at the same time don't overly constrain the design engineer. Often this results in complex rules. If they are inadvertently broken, the usual outcome is a decrease in the machine's reliability. The broken rules result in components that operate with excessive temperatures or signals that do not have adequate noise margins. A chip that runs too hot will fail sooner than anticipated; a signal that doesn't have adequate noise margin will sometimes be sensed incorrectly. Worse still is the fact that the component is blamed rather than the true cause, a violated rule.

As an example consider the operating temperature of an IC. There is a tradeoff between the maximum and minimum operating temperatures and the amount of noise margin available. If the temperature of an IC exceeds its maximum specified temperature, the amount of noise normally present from known sources, such as adjacent-run crosstalk, may be sufficient to produce a false signal. Therefore, it is important that all ICs stay within their specified operating temperatures. To ensure that, we developed a tool for use on the 8600 to check for chips that were getting too hot. If a chip was detected as being too hot, its layout was modified to correct the problem without changing the total power of the module.

A new timing analysis tool was also developed for the project. This tool enabled the designers to do a much more thorough job of timing analysis on this machine than had been done on previous projects. Using it involved running many separate programs that built a timing model of the machine from the schematics and the layouts of the modules, backplane, and MCAs. The results of the model were then used by a program that performed timing analysis of the design based upon a set of interbox timing specifications.

After the layouts of the modules were completed, every single run was analyzed to ensure that signal integrity had been achieved. The program computed the amount of noise generated from adjacent runs,

reflections, and the like. Based on these results, we made a number of reroutings to increase the integrity of certain signals.

#### FAULT TOLERANCE

All the efforts discussed in the previous section improved the machine's reliability. However, the logic could still fail and therefore it was important to have mechanisms to recover from a logic fault whenever possible. Fault isolation and fault tolerance are highly correlated, not separate issues. Data integrity and retry operations depend on good fault detection. So does the ability to reconfigure the system when a fault occurs, a situation that requires accurate fault isolation as well [4]. It is important to know what type of fault was made and what processes may or may not have been affected by it. To accomplish fault isolation, we had to develop an effective fault detection and reporting scheme.

The design philosophy for the fault system had several major concepts. The first was that faults that occur synchronously with the program counter (PC) should be reported synchronously to it. Synchronous faults have a direct relationship to the current value of the program counter. For example, consider a write to an I/O register. Only one cycle is required for the M Box to accept all the information to perform the write operation. In the meantime, the E Box could continue processing instructions. The problem here is that if the I/O write has a fault, the current PC of the machine would have no fixed relationship to that fault, thus making recovery more difficult. To solve this problem, the microcode will stall the E Box on an I/O write until the confirmation of that write is received.

A similar problem exists with a translation buffer (TB) miss on a prefetch for the instruction buffer. If a branch is ahead of the TB miss in the instruction buffer and the branch is taken, the TB miss will not be a problem and should not be reported. In this case the design requires a delay in sending the TB miss signal to the E Box (which performs the memory management operations) until it attempts to execute the instruction whose prefetching caused the TB miss. In general, synchronous faults are reported via E Box microtraps.

Faults that are asynchronous to the program counter are reported asynchronously. Asynchronous faults are ones for which the value of the program counter has no definite relationship and which are usually reported through interrupts. Two examples of an asynchronous fault are a fault occurring on a disk write to memory and a parity error on a cache writeback operation.

At the time a fault is detected, it may not be known whether the fault should be reported synchronously or asynchronously. In that case, both fault-logging mechanisms are invoked: a microtrap for synchronous and an interrupt for asynchronous faults. Consider the case of a parity error on an instruction prefetch. If the E Box executes a branch prior to using the bad data, then the synchronization will never be reached and the fault will be logged through an interrupt. In this case the microtrap condition will be cleared by the execution of the branch. If, however, the E Box attempts to execute the prefetched instruction with the parity error, then an E Box microtrap will occur and the trap routine will clear the interrupt.

The second major concept used throughout the design was that hardware faults are considered to be process faults only if a process attempts to use or store corrupted data. For example, if corrupted data is detected during a writeback to memory from the cache, a fault will be logged. However, the process will not experience a fault until it attempts to either consume the corrupted data or store it on a disk. This logic imposes the requirement that corrupted data can be marked for later detection, which is done with ECC code in memory. This subject is discussed in the UNIQUE RELIABILITY FEATURES section.

#### FAULT MINIMIZATION

When recovery is not possible, the next best thing is to control the amount of damage done by a fault. This tactic requires fault information that is accurate, relevant, and sufficient. Whenever a fault occurs, an error stack frame will be constructed by the E Box and placed in memory. The stack frame format is the same for all errors. We made no judgement as to what would be useful in determining which information was relevant.

In the case of damaged data, fault reporting alone is not sufficient, since it is not possible to determine which process will access that data. Therefore, when data damage occurs, the logic marks it as "bad" and any future user of that data will be notified of that fact.

#### MEAN TIME TO REPAIR

There are two kinds of machine failures: those in which fault symptoms are solid, and those in which fault symptoms are intermittent. Of the two, solid faults are easier to diagnose. To isolate solid faults, the console can examine the state of the signals that go from one module to another. Diagnostics are run to find the first failed test, which is then run in a single-step manner looking for the first incorrect signal. With the exception of multiple-source

signals, the source of the first incorrect signal value is the failing module (since all of its inputs have been checked by this process). In this way faults can be isolated to the field replaceable unit.

Intermittent faults are much more difficult to diagnose and they comprise between 80% and 90% of the faults. Diagnostics rarely provoke intermittent faults. But when they do, the fault reporting can often be confusing. This confusion occurs because a logic fault will usually take place in a circuit after it has been tested and while another circuit is being tested [5]. The number of fault checkers in a machine affect its ability to know that a fault has occurred and to identify the failing unit. The probability of a fault occurring in the logic that any given checker has checked is not affected by whether the result is used or not. If an intermittent fault occurs on a path that isn't being used, then no real fault has occurred. Therefore, the machine's overall reliability is increased by ensuring that fault checking is performed only on networks that are actually being used.

IN the APPENDIX a detailed list of the checkers included in the VAX 8600 system is available.

If a failure occurs that requires immediate power shutdown, then remote diagnosing through the console cannot be used. This occurs when the regulators detect an overheating condition or the power for the EMM is out of tolerance. In these cases a magnetic indicator code that contains the failing regulator number will be displayed on the EMM module. This code enables a field service technician to know which regulator to replace.

#### UNIQUE RELIABILITY FEATURES IN THE VAX 8600 CPU

In addition to the reliability features already discussed, the VAX 8600 design includes some not previously found on other Digital machines. These features are discussed under the four major areas used in the first part of this paper.

#### Fault Avoidance

The F Box executes self-diagnostics when it is not performing floating point instructions. These tests use "live" operands to enhance the detection of data-dependent faults. Both the E Box and the F Box are connected to a common source of instructions and operands. When the F Box detects that it cannot perform an operation, it will execute a diagnostic self-test. Exactly which self-test is performed depends upon the instruction. The number of machine cycles in the diagnostic routine is chosen to be equal to or less than the number of machine cycles used by the E Box. This insures that the F Box will always be ready for

the next floating point operation that will be passed to it. If a fault is detected, the F Box will be turned off and the E Box will perform the instruction that would have been done by the F Box, only at a much slower speed.

#### Fault Tolerance

The 8600 supports instruction retry where possible. If a fault occurs that causes a microtrap during an instruction, a set of instruction retry flags will be passed along through the various fault recovery stages. The flags indicate whether or not the CPU has performed an operation that would make restarting the instruction impossible. An instruction retry would be inhibited if an I/O-read, a memory-write, a state-modified, or an E Box abort bit is "on." Otherwise, the instruction can be restarted.

The data cache can recover from single-bit errors. A cache data entry consists of 32 bits of data, 4 bits of byte parity, and 7 bits of ECC. The write of the check bits is pipelined and occurs in the cycle following the write of the data. The parity bits are used for fault detection and the ECC bits for error correction. The M Box always passes data to the E Box or I Box before any checking is done. If the data contains a parity error, then either the E Box or the I Box, as well as the M Box, will detect it. The M Box will then block the acceptance of any more requests and will execute a data correction sequence. The ECC code and the data are then sent to the array bus, and normal array-to-M Box data correction is applied. The "corrected word" is then written back into the cache. At some point the E Box will discover that it has been shipped bad data. The system will then retry the instruction if possible. The retry will be successful if the original fault was correctable.

An important goal of the power subsystem is to increase its tolerance of bad ac input power. The power input is a true 3-phase input with very low neutral current. In previous designs the power-storage capacitors had been attached to the regulator outputs. The detection of power failures was performed by monitoring the ac line. In contrast, the VAX 8600's power system first converts power to 300 Volts dc and then sends that power to regulators in order to produce the final output voltages. Power storage is done at the 300 Vdc level. This higher voltage allows more energy to be stored, since the storage is provided by capacitors. Power-failure detection is performed by monitoring the voltage level on the 300 Vdc power supply. When its voltage reaches the level at which there is just enough energy remaining to perform a power-fail sequence, then an ac power failure will be declared. This method

allows continued operation regardless of the ac input waveform, as long as the machine receives sufficient energy, a fact that is especially helpful during brownout conditions.

#### Fault Minimization

The 8600 makes good use of the unassigned ECC codes (a 7-bit ECC can correct up to 57 bits of data). They are used to detect array addressing problems and to flag any corrupted data. When a memory write occurs, the parity of the address and an indication of the quality of data are sent to the ECC generator. The quality of data is good if no faults were detected during its transmission to the M Box and bad if the machine suspects that a fault is present. The address parity and quality information are inserted into the ECC generator by means of bits 32 and 33 of the data. Neither of these bits is stored in the array. When the data is read back, the computed address parity is sent along with a good-data signal to the ECC generator. If the computed syndrome is zero, the transaction is considered to be good. If the ECC generator decodes a single-bit error pointing to the address bit, then an address parity error will be declared. When that occurs, it means that the word that was just received did not come from the address that it should have. Thus, the ECC generator can check the address lines from the M Box to the MOS array chips and detect the control faults that caused the M Box to access the wrong data word. If the chip thinks the quality bit needs correction, then the data word was faulty when it was received. The requester of this data will then be notified that the data is bad. If a normal single-bit error occurs on a data word that was stored with a code indicating bad quality, then the M Box will flag an ECC double-bit error.

Most of the internal busses in the VAX 8600 CPU as well as the shifter and the arithmetic logic units (ALU) are parity checked. The ALUs are checked by triplication and parity checking the results. The I Box, F Box, and E Box each contain a set of general purpose registers (GPRs). When writes to the GPRs occur, all GPRs are written to simultaneously, thus keeping them consistent. If a GPR parity error is detected in one box, a recovery will be initiated that copies correct data from the equivalent GPR in another box to the failed GPR. Thus the machine can recover from GPR parity errors.

#### Mean Time To Repair

The number of microsequencers in the VAX 8600 system also adds to its reliability. Ordinary combinatorial control logic is difficult to check

without duplication. Using a microsequencer is a method of building control logic that is easily checked. For example, all the microcontrol stores are parity checked. The M Box also checks the parity of the address, stack underflow and overflow, and stack address parity. Microparity errors are recoverable in the E Box, F Box, and I Box. These faults are not recoverable in the M Box since its state is modified in an unrecoverable manner before the parity computation is complete.

#### SUMMARY

The task of making large machines reliable requires a continuous effort in all phases of the project, from conceptual design to manufacturing. In the future, machines will continue to get larger. Unless some major technology breakthrough that significantly changes the reliability of components occurs--as did occur when transistors replaced tubes--the fault-handling capability designed into large systems must be improved. This improvement is needed to overcome the inherently higher failure rate that comes with having more components. Based on this conclusion, we created many design processes, manufacturing processes, and fault handling features that increased the reliability of the VAX 8600 system. Careful monitoring and simulation were required to insure that true gains in reliability were actually achieved.

#### REFERENCES

- [1] Daniel P. Siewiorek and Robert S. Swarz, *The Theory and Practice of Reliable System Design* (Bedford: Digital Press, 1982)
- [2] Lynne S. Rosenthal, "Planning and Implementing System Reliability," *IEEE Total Systems Reliability Symposium* (December 12-14, 1983): 112-118
- [3] Edward J. McCluskey, "Reliable Computing Systems", Technical Note No. 182, Center for Reliable Computing, Stanford University (October 1980)
- [4] Vincent A. Cordi, "4381's Error Detection Fault-Isolation Speeds Repairs," *Computer Systems Equipment Design* (November 1984): 23-29
- [5] George H. Maestri, "The Retryable Processor," *IEEE Fall Joint Computer Conference* (1972): 273-277

APPENDIX  
FAULT CHECKERS  
IN THE VAX 8600 SYSTEM

IN THE I BOX

IN THE E BOX

ALU OUTPUT PARITY CHECK  
SHIFTER PARITY CHECK  
MICROCODE PARITY CHECK PER BOARD  
OTHER RAM STORE CHECK WITH  
SEPARATE ERROR FLAGS  
AMUX PARITY CHECK  
BMUX PARITY CHECK  
GPR COPY WRITE RECOVERY  
INSTRUCTION RETRY  
DIAGNOSTIC FAULT INSERTION

MICROWORD PARITY CHECK  
IBUFFER PARITY CHECK  
DRAM PARITY CHECK  
GPR PARITY CHECK  
OP BUS PARITY CHECK  
W BUS PARITY CHECK  
IMD PARITY CHECK

IN THE M BOX

MEMORY ADDRESS PARITY CHECK  
ECC ON CACHE AND MOS MEMORY DATA  
WRITEBACK ON SBE  
MICROWORD PARITY CHECK  
MICROADDRESS PARITY CHECK  
MICROSTACK PARITY CHECK  
MICROSTACK UNDERFLOW/OVERFLOW DETECT  
A BUS PARITY CHECK  
ARRAY BUS PARITY CHECK  
CORRUPTED DATA TAG  
CPR PARITY CHECK

IN THE F BOX

FBM MICROWORD PARITY CHECK  
FBA MICROWORD PARITY CHECK  
FDRAM PARITY CHECK  
GPRs PARITY CHECK  
SELF-TEST (WHEN NOT EXECUTING  
INSTRUCTIONS)

**POSTER PAPER**



IMPLEMENTATION OF A LOCAL AREA NETWORK AT  
LOS ALAMOS MESON PHYSICS FACILITY (LAMPF)

Anthony M. Gonzales  
Los Alamos National Laboratory  
Los Alamos, New Mexico

ABSTRACT

This paper presents a summary of the implementation of a Local Area Network at the Los Alamos Meson Physics Facility (LAMPF). The network described is unique in that LAMPF is a large complex with a broad area dedicated to experimental stations. The paper describes some of the problems that were encountered and why the ethernet topology was finally decided upon.

INTRODUCTION

The Los Alamos Meson Physics Facility (LAMPF) is part of the Los Alamos National Laboratory, managed by the University of California, for the U.S. Department of Energy. LAMPF is one of the worlds largest and most powerful nuclear science research facilities. The facility is primarily a tool for atomic, nuclear and particle physics research. The installation consists of a half-mile long linear accelerator and several experimental areas served by simultaneous beams from the accelerator. The base facility was completed in 1972 at a cost of \$57 million dollars. A layout of the experimental areas as they exist today is shown in Figure 1. Since the

completion of the base facility, a Weapons Neutron Research Facility and a Proton Storage Ring have been added to the complex and a Neutrino Experimental Tunnel is now under construction.

The beams from the accelerator include a 1 mA proton beam to beam area A, 0.1 mA beam to the proton storage ring, and lower intensity beams to lines B and C. The accelerator operates in a pulsed mode with a duty factor of about 10% so that the peak currents are much higher than the averages mentioned above. The data acquisition system, Q, was designed specifically for the computers and the experiments that are peculiar to LAMPF. Data for each experiment are typically collected through CAMAC and recorded on magnetic tapes with PDP-11 computers.

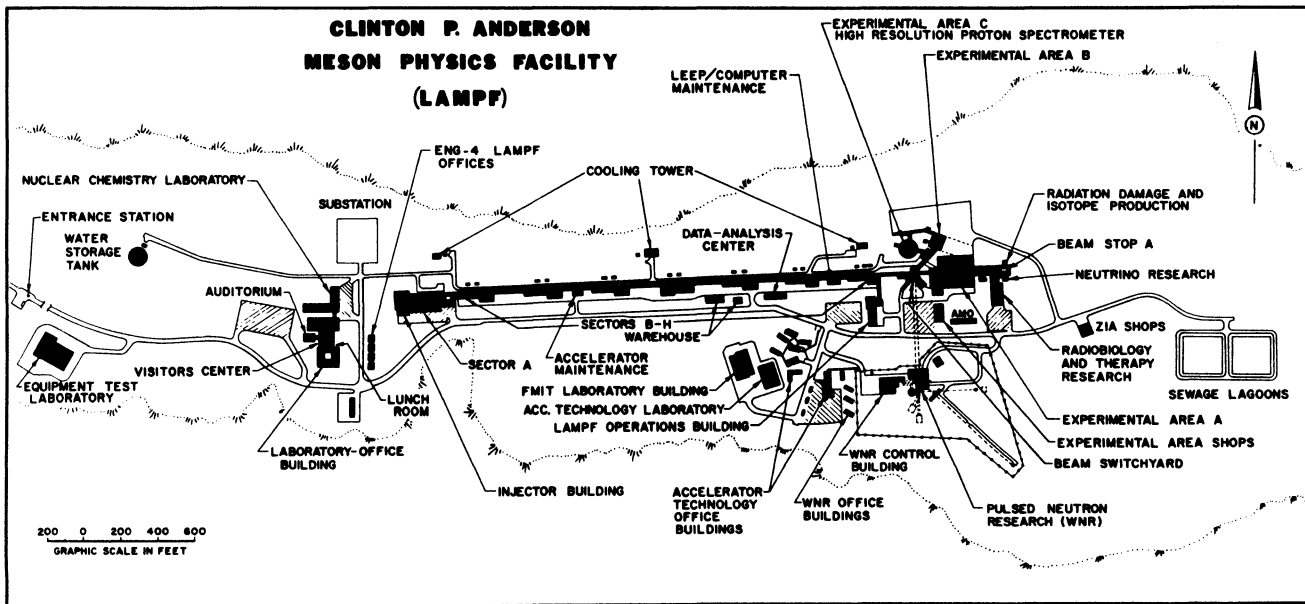


Fig. 1. LAMPF as it exists today.



The data acquisition computers are generally PDP-11/45, 11/34, 11/44 and 11/70's. As experimental needs increase, upgrades of PDP to VAX computers are planned and VAX-11/750's are now beginning to appear on site. Data rates for experiments generally do not permit analysis of all events as they are collected.

Analysis of LAMPF experiments is done for the most part at the Data Analysis Center (DAC). The analysis center currently houses 4 VAX-11/780's and 2 PDP-11/70's and in the near future will be adding the newly announced VAX 8600. The VAX computers are running VMS Version 4.0 in a cluster and therefore share common disks. Data tapes can be replayed on the VAXes and the Q programs have been recently modified to allow replay of data stored on disk. The data tapes can be spooled directly to disk and run on any of the computers linked into the cluster. DECnet is the protocol used for communication between the computers. The DAC also has a broadband connection to the Central Computing Facility, which at present is one of the largest computing centers in the world.

There are currently over 600 user accounts on the VAXes. The number of users on-site varies as different experimental programs receive beam time. Some users remain based at LAMPF year-round while others spend most of their time at home institutions. Because of the variable nature of the user community, constant updates and retraining are required as hardware and software change.

#### PROJECT INITIATION

In the fall of 1983 a committee was formed to assess the current status and to recommend directions for computing at LAMPF for the near future (Ref 1). This study included recommendations for both data acquisition and data analysis and emphasized methods to make more efficient use of resources in the next 5 years. One of the priority items to come from the Long Range Planning Committee was a recommendation for a Local Area Network to connect the experimental area computers to each other and to the DAC.

The benefits derived from such a network would include sharing of codes, which would lead to a greater amount of communication between experiments thus resulting in a high degree of information exchange. Also, shared resources would become available. For example, peripheral devices, such as line printers, could then be used from another experiment as need and usage dictated. The ability to share resources among the computers could also be used to off-load some CPU-bound jobs from DAC computers to experimental area computers during beam-off time (about six months per year). However, the network was not designed to be able to ship data from all locations to the DAC for analysis directly. Tapes will continue to be the primary method for data transfer to the DAC.

At LAMPF there are presently more than 70 computers. These range from micro PDP's to the VAX-11/780's. A majority of the small computers run RSX-11M systems. All of these computers are potential network nodes.

A list of requirements was drawn up to specify the capabilities required for an experimental area network. The requirements include high speed and a reliable technology that will be relatively easy to maintain. Because the network will include both PDP's and VAXes, a flexible system is needed that will provide both hardware and software compatibility for the different systems. Flexibility of adding or removing systems is needed since different experiments run simultaneously but not on the same schedule. It is also important that computers can be rebooted without affecting the network. A single networking system with a uniform interface is desirable because maintenance will be done by an on-site computer maintenance group. Hardware that can use a DECnet protocol is attractive because DECnet is used for communication between the VAXes in the DAC and the training required for users of the network can be minimized. A network that will not quickly become obsolete is also a requirement because the investment in both time and money will be significant. Finally, the network must be cost-effective because one cannot prioritize the experiments and decide who will be able to take advantage of the network.

#### DECIDING ON A NETWORK

Some technologies now offered on the market were explored. Point-to-point and ring networks were considered. Because of the layout of the experimental areas these networks were found to be rather inflexible and difficult to implement. Another kind of network that was studied was a broadband connection, but this option proved to be too expensive for our applications.

Ethernet is a simple and low-cost network. Ethernet protocol, with its high speed bandwidth of 10 megabit/sec, satisfies the requirements most comfortably. Some of the added benefits derived by the choice of ethernet are the flexibility of addressing nodes, the short delay time, and stability under all load conditions. The ability to add connections such, as personal computers or local area terminal Servers (LATS), or to remove connections without interruption to the overall network is a big plus.

#### DESIGN AND IMPLEMENTATION OF ETHERNET

Experiments are often mounted in trailers that are temporarily connected to the experimental buildings. The facility also supports "counting houses" equipped with data acquisition computers. Designing a single network to cover all the present experimental stations and anticipating new additions proved to be a project of considerable challenge.

The accelerator is run on radio-frequency (rf) power that can potentially create enough noise to degrade data transmission over ethernet. Phase 1 consisted of a test of data transmission while the RF power was on. A PDP-11/34 located down the beam line was connected to a PDP-11/70 located at the DAC with teflon-coated coax. After several days of testing, no distortion of data was detected. Teflon cable was chosen because it is well suited to handle harsh

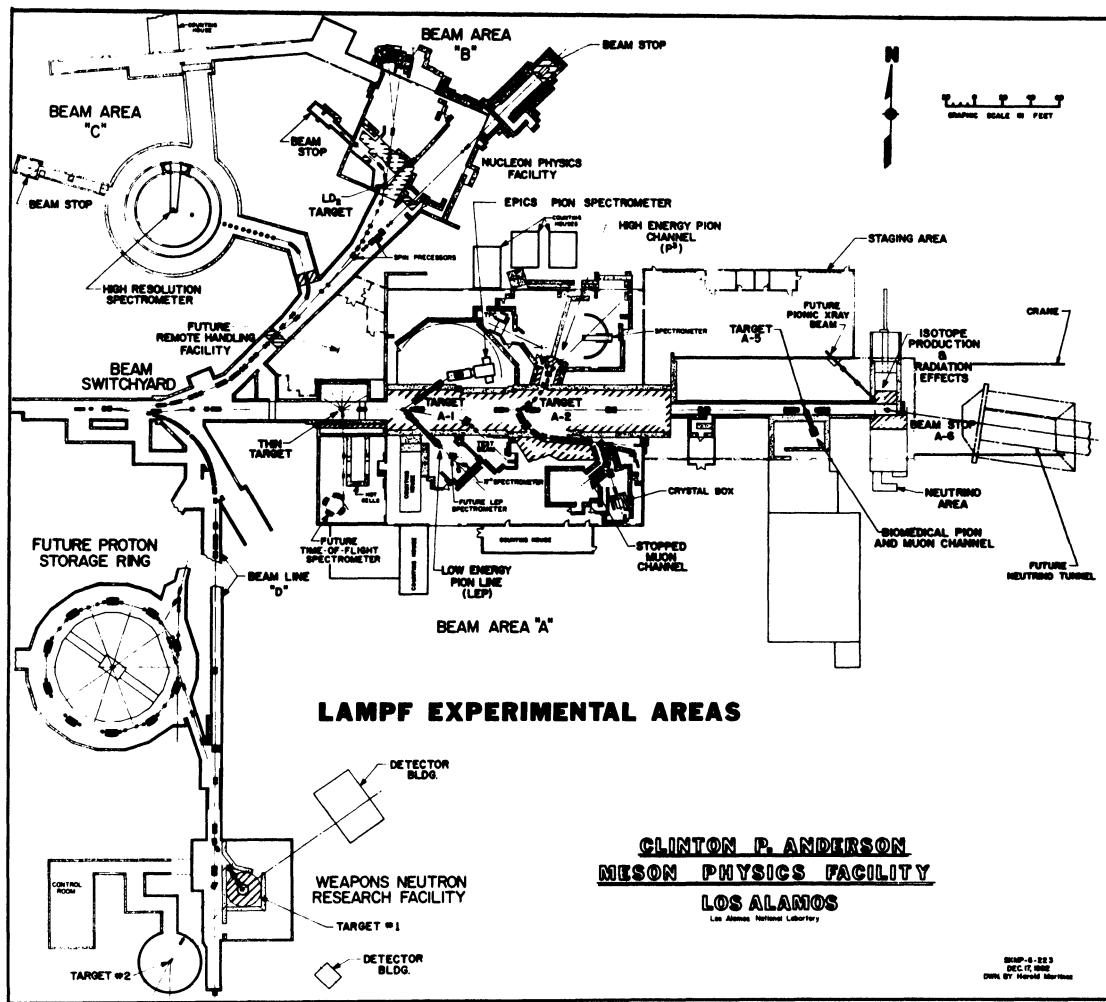


Fig. 2. Experimental areas showing beam lines.

environments such as are found in the experimental areas; for example, cables are subject to abuse as they are moved in cable trays.

Phase 2 consisted of laying out a plan for the network cable throughout the experimental areas. Figure 2 is a map detailing the experimental areas as they exist today. The map shows that the experimental areas are almost as large as they are spread out. Cable trays reach to all of the experimental areas but are in places difficult to access and run along walls near large sources of electro-magnetic interference.

Some special considerations were given to the cable layout. For example, active electronics cannot be placed in the cable trays. Ethernet repeaters are considered active because of the 110 volt circuits needed to power them. We purchased 40m transceiver cables so repeaters could be mounted along the walls of the beam lines. Another potential problem involved the cable paths to the neutrino area at the east end of Area A. Because the coax runs underground, teflon-coated ethernet cable was chosen. The conduit will be used for other transmission cables and there exists the possibility of leakage that could let moisture into the cables. Finally, because many computers are located in

trailers located outside the buildings, the cable paths chosen must provide convenient connections for those computers. The use of 40m transceiver cables provides just enough added distance to reach most of these areas and still remain within specifications.

The cable path chosen for the backbone of ethernet runs from the DAC along the accelerator to Area A, along the north wall in Area A, and into the Staging Area. A local repeater is required at that point to extend the network to the neutrino area. A second repeater will be connected in the switchyard and a section of ethernet cable will be run along the outside walls of Area B and around to Area C. Another repeater will be attached just inside Area A and will extend along the south side of the building. The ethernet cable layout holds the maximum distance between nodes to that required by the present specifications and requires no more than two repeaters between any two nodes.

After the completion of Phase I, the ethernet cable was run out to Area A. The transceiver that had been mounted on the cable for testing was removed and the tap in the cable was wrapped with electrical tape. A time-domain reflectometer (TDR) was used to test the cable for damage during the installation. Figure 3 shows the signal obtained from the TDR.

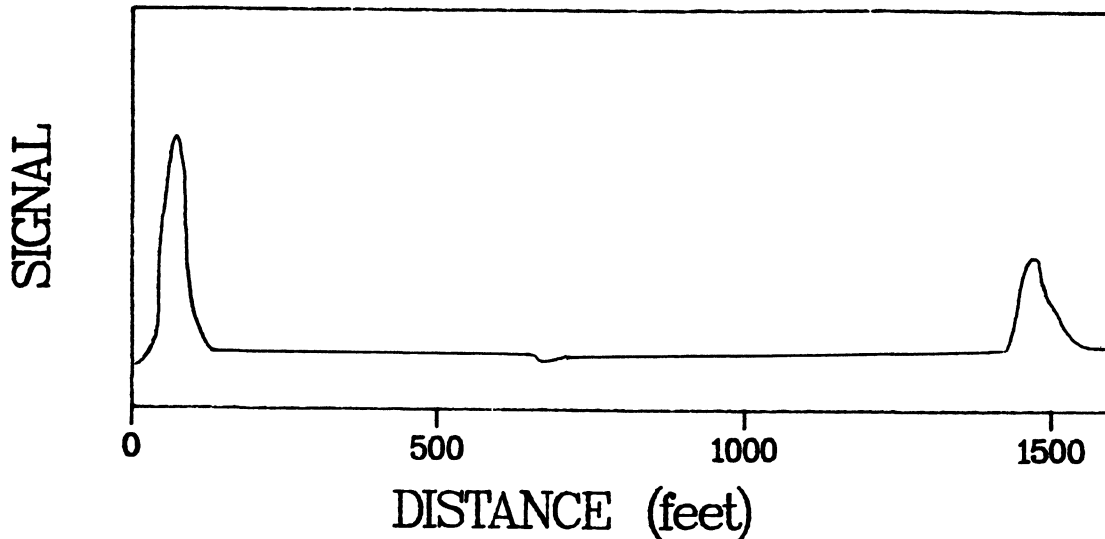


Fig. 3. Time Domain Reflectometer pattern.

The first peak is the signal sent down the cable and the final peak is the reflection from the unterminated end of the cable in Area A. The total length of the cable is 421m. The small glitch in the signal is from the hole left from the tap made for the Phase I test. Before many more experiments come on-line the effects of removal of taps will be investigated.

One final problem was related to the fact that most of the data acquisition computers are PDP-11/45's with an 18-bit bus structure and memory limited to 128K. The memory limitation makes it difficult to include both DECnet tasks and all tasks usually installed in the RSX-11M operating system. An attempt to install the Q data acquisition tasks brought the computers to a virtual halt for lack of pool space. The tentative plan is to connect only to 22-bit computers and to the VAXes.

#### FUTURE FOR LAMPF AND NETWORKING

By the time this paper is published most of the network connections mentioned will have been made. Future expansion will depend on the direction LAMPF takes and the ever-changing needs of the user community. Additional networks, perhaps to the Laboratory Office Building at the other end of the accelerator, may be installed in the future. Networks will be connected either through a routing computer or through some sort of network bridge. Fiber optics may be used to serve other sites. Certainly the network proposed here will be able to handle all of the existing experimental stations and will allow for efficient use of a multi-vendor environment.

#### ACKNOWLEDGEMENTS

I would like to thank Genaro Maestas and James Wilmarth for helping implement the network and especially thank Martha Hoehn for the time and effort she spent in helping me throughout the entire project.

#### REFERENCES

1. M.V. Hoehn et al., "Long-Range-Planning Committee for LAMPF Computing Needs Report," Los Alamos Scientific Laboratory report-10103-MS (JUNE 1984).

**Authors Index****Page****Page**

|                             |          |                                |              |
|-----------------------------|----------|--------------------------------|--------------|
| Abate, J.A. ....            | 33       | Lederman, Bart Z. ....         | 71, 127      |
| Anderson, John M. ....      | 145      | Leonard, Stevan ....           | 17           |
| Attaya, Steve ....          | 219      | Lockrey, Brian D. ....         | 111          |
|                             |          | Lund, T.S. ....                | 33           |
| Baren, Jill M. ....         | 83       | Mandley, Donald J. ....        | 385          |
| Barnard, Ralston ....       | 393      | Mansfield, Michael K. ....     | 169          |
| Beyer, John W. ....         | 471      | Mansfield, Patricia K. ....    | 169          |
| Birkelund, J.R. ....        | 33       | Mickelson, Carl T. ....        | 405, 411     |
| Blinn, Thomas P. ....       | 249      | Moriarty, Leonard J. ....      | 323          |
| Bloem, John ....            | 535      |                                |              |
| Brown, G. ....              | 49       | Naecker, Philip A. ....        | 509          |
| Bruckert, William ....      | 557      | Naegele, Mary Lou ....         | 27           |
|                             |          | Nagel, Bernard E. ....         | 39           |
| Ching, Steve ....           | 535      |                                |              |
| Creel, Larry R. ....        | 105      | Osorio, Fernando C. Colon .... | 535          |
| Curley, Robert F. ....      | 83       |                                |              |
|                             |          | Perkins, Dorothy C. ....       | 433          |
| Dayton, David ....          | 65       | Pflanz, Nancy R. ....          | 287          |
| DenTandt, Donald ....       | 211      | Podany, Mark ....              | 419          |
| Diba, Ali T. ....           | 497      | Porada, Susan ....             | 247          |
| Doubleday, Raymond J. ....  | 3        |                                |              |
| Downey, Arthur E. ....      | 349      | Quaynor, Nii ....              | 535          |
| Downward, James ....        | 549      |                                |              |
|                             |          | Ramsey, Besty ....             | 201, 217     |
| Ebinger, Larry W. ....      | 459      | Richardson, Robert C. ....     | 313          |
|                             |          | Riviere, Marisia ....          | 441          |
| Franklin, Sue Ellen ....    | 271      |                                |              |
| Friedman, Gary ....         | 509      | Schell, R. ....                | 329          |
| Friesen, R. ....            | 329      | Schornak, Clifford J. III .... | 185          |
| Fulton, Richard G. ....     | 175      | Shannon, Terry C. ....         | 55, 361, 367 |
|                             |          | Silverstein, Mark ....         | 13           |
| Galvin, Peter B. ....       | 223, 267 | Simmons, A. ....               | 329          |
| Glasser, Harold T. ....     | 509      | Simon, Denise ....             | 297          |
| Gonzales, Athony M. ....    | 565      | Smith, Theodore J. ....        | 83           |
| Gould, Herbert J. ....      | 27       | Stevens Jack ....              | 205, 213     |
|                             |          | Stewart, John N. ....          | 397          |
| Harenen, Harry ....         | 375, 379 | Szczur, Martha R. ....         | 433          |
| Hare, Keith W. ....         | 97       |                                |              |
| Hayashida, Myron K. ....    | 277      | Tenorio, Ramon ....            | 65           |
| Helton, J. ....             | 329      | Toriani, Mario ....            | 535          |
| Howell, David R. ....       | 433      |                                |              |
|                             |          | Valentine, Pamela A. ....      | 509          |
| Jalbert, Jeffery S. ....    | 97       | Vasconcelos, John J. ....      | 497          |
| Janik, Charles S. ....      | 155      |                                |              |
| Jaquith, Elliot F. jr, .... | 121      | Walraven, Robert ....          | 393          |
| Johnson, B. ....            | 49       | Wang, Ching Po ....            | 39           |
| Johnson, Sharon Linnea .... | 427      | Warner, Richard H. ....        | 445          |
| Josephson, Ronald ....      | 557      | Wells, Robert ....             | 527          |
| Joy, Michael D. ....        | 225      | Werner, Nancy E. ....          | 487          |
|                             |          | Wilson, Bob ....               | 523          |
| Kassebaum, Donald A. ....   | 221      | Wims, Anderw M. ....           | 39           |
| Kopec, Richard L. ....      | 149      |                                |              |
| Kramer, William T. ....     | 331      | Yochmowitz, M. ....            | 49           |
|                             |          |                                |              |
| Lamaestra, Susan M. ....    | 215      |                                |              |
| Lareau, Jean M. ....        | 45       |                                |              |





Special  
Fourth-Class Rate  
U.S. Postage  
**PAID**  
Permit No. 18  
Leominster, MA  
01453

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY  
249 NORTHBORO ROAD, BP02  
MARLBORO, MASSACHUSETTS 01752

