USA
1978
FALL

PROCEEDINGS OF THE DIGITAL EQUIPMENT COMPUTER USERS SOCIETY Vol. 5, No. 2

PROCEEDINGS OF THE DIGITAL EQUIPMENT COMPUTER USERS SOCIETY Vol. 5, No. 2 U.S.A. FALL 1978

PROCEEDINGS OF THE DIGITAL EQUIPMENT COMPUTER USERS SOCIETY

Vol. 5, No. 2

PAPERS AND PRESENTATIONS
USA FALL 1978

1978 FALL DECUS U.S.
SAN FRANCISCO HILTON HOTEL
SAN FRANCISCO, CALIFORNIA
NOVEMBER 27-30, 1978

Maynard, Massachusetts
ISSN 0095-2095

4/79-14

FOREWORD

This journal is a publication of the Digital Equipment Computer Users Society, a world-wide society of users of computers manufactured by Digital Equipment Corporation. The Society has at present more than 33,000 members, and has offices in Geneva, Switzerland; Kanata, Ontario, Canada; and Crows Nest, Australia as well as headquarter offices in Marlboro, Massachusetts, U.S.A.

The Society maintains a library of programs for interchange between members, and organizes meetings on local, national, and international scales to fulfill its primary functions of advancing the art of computation and of providing means of interchange of information and ideas between members. Five major technical Symposia are at present held annually, one each in Canada, Australia, and Europe, and two in the United States.

This journal consists of an annual volume in five parts, each the Proceedings of one of the annual DECUS Symposia. The issues are numbered according to the sequence in which the Symposia take place.

No. 1 — DECUS Europe Symposium
No. 2 — DECUS U.S. Fall Symposium
No. 3 — DECUS Canada Symposium
No. 4 — DECUS U.S. Spring Symposium
No. 5 — DECUS Australia Symposium

A cumulative index for the volume will be published with Number 5.

Proceedings for Symposia held prior to September 1974 are not assimilated into the series. In addition, all past Proceedings are available on microfilm from:

Europe and Eastern Hemisphere:	All Others:
University Microfilms International 18 Bedford Row London WC1R-4EJ, England	University Microfilms International 300 North Zeeb Road Ann Arbor, Michigan 48106

For details of subscription rates for the Proceedings, of back-issues availability, and of forthcoming DECUS Symposia, contact one of the following:

CHAPTER OFFICES

Australia/NZ:	**Canada:**	**Europe/Middle East:**	**U.S. and All Others:**
DECUS Australia P.O. Box 491 Crows Nest, N.S.W. 2065 Australia	DECUS Canada P.O. Box 11500 Ottawa, Ontario K2H 8K8 Canada	DECUS Europe 12, avenue des Morgines C.P. 510 CH-1213 Petit-Lancy 1, Geneva Switzerland	DECUS U.S. One Iron Way Marlboro, MA 01752 U.S.A.

TABLE OF CONTENTS

PDP-11 SOFTWARE RSTS

PDP-11 SOFTWARE RSX-11

A SIMPLE INDIRECT COMMAND FILE PROCESSOR

FOR RSTS/E

Norm Seethoff
John Fluke Manufacturing Company
Seattle, Washington

ABSTRACT

A few modifications to the BUILD program (distributed with all RSTS kits) are described. The modified BUILD program will execute commands from a file, saving the user from typing long or repetive input strings. This program is especially suitable for use with BACKUP, and an example shows how to automate BACKUP commands.

INTRODUCTION

BEGIN is a utility program designed to execute a named command file from any RSTS terminal. This capability permits a user to specify a file name from which commands are to be read and executed as if they were typed directly from the terminal. BEGIN is an alternative to typing in a lengthy series of commands for repetitive operations such as large assemblies, compilations and program linking. It is also very useful for applications in which another program can generate a complex sequence of commands to be executed by an inexperienced operator using BEGIN. In addition to command file exectuion, BEGIN presently offers the capability for one wild card string replacement and three private logical device assignments.

EXAMPLES

FORTRAN Compilation

As an example, assume that the file EXAMPL.CTL contains the following:

```
!Sample Command File for BEGIN
RUN $FORTRAN
O:TEST,L:TEST=I:TEST/L:?
^
Q LP:EXAMPL=L:TEST/NH/DE
```

When executed with the following commands (user entries are in bold type):

```
RUN &BEGIN
BEGIN Ver. 1.0 RSTS V06C John Fluke Mfg
BEGIN>EXAMPL,SRC,DR0,DR0,DR1
```

The following command dialogue takes place (commands generated by BEGIN are underlined):

```
RUN &BEGIN
BEGIN Ver. 1.0 RSTS V06C John Fluke Mfg
BEGIN>EXAMPL,SRC,DR0,DR0,DR1
^C
HELLO
RSTS V06C John Fluke Mfg Job 17 KB0
#100,100
Password:
Job 10 is detached under this account
Job numer to attach to?
1 other user is logged in
Ready
ASSIGN DR0:I
Ready
ASSIGN DR0:O
Ready
ASSIGN DR1:L
Ready
! Sample Command File for BEGIN
RUN $FORTRAN
*O:TEST,L:TEST=I:TEST/L:SRC
.MAIN.
*^Z
Ready
Q LP:EXAMPL=L:TEST/NH/DE
Ready
BEGIN Complete
```

System Disk Backup

BEGIN is a useful aid in the automation of disk backups. The following example shows the use of BEGIN to perform a system disk backup by executing a command file generated by yet another program. By using a separate program to generate the command file, a shell of commands can be constructed and the actual variable parameters (BACKUP setname, expiration date, etc.) can be filled in by the program. Use of such an approach for system backup minimizes the chances of operator errors during the backup. For clarity, the commands generated by BEGIN are underlined. User entries are in bold type.

```
RUN SYSBAK
Today's backup set tapes are:
     THUDR0
     THUDR1
To back up the system disks, enter
the following commands:
     For DR0 enter:
          BEGIN DR1:DR0
     For DR1 enter:
          BEGIN DR0:DR1
After you have completed the disk
backup, enter the following command to
QUE the listings and delete the scratch
disk files:
     BEGIN DR1:BACKUP
Have at it.................
Ready

BEGIN DR1:DR0
^C
HELLO
RSTS V06C John Fluke Mfg Job 17 KB0
#1/100
Password:
Job 9 is detached under this account
Job number to attach to?
1 other user is logged in
Ready
!     BEGIN control file for system
!     disk backup
!
!     Mount the first volume of backup
!     set THUDR0 on MT0:
RUN $BACKUP
BACKUP  V06C-03A
BAC[KUP] OR RES[TORE]?     BACKUP
WORK FILE NAME?            DR1:THUDR0.WRK
LISTING FILE<KB:>?         DR1:THUDR0.LST
FROM DISK<SY:>?            DR0:
FROM FILES<[1,100]*.*>?    [*,*]*.*
TO DEVICE<MT:>?            MT:
BEGIN AT<[*,*]*.*>?        [*,*]*.*
DELETE FILES<NONE>?        NONE
COMPARE FILES<NONE>?       NONE
*
BACKUP SET NAME<THUDR0>?   THUDR0
EXP DATE<26-OCT-79>?       01-NOV-78
DENSITY IN BPI<800>?       800
PARITY<ODD>?               ODD
MOUNT     DEVICE:          MT:
             ID:           THUDR0
           SEQ#:           1
        DENSITY:           800 BPI
         PARITY:           ODD
IDENTIFICATION WILL BE FINAL UPON MOUNT
DEVICE? MT0:
*
DISMOUNT DEVICE:           MT0:
             ID:           THUDR0
           SEQ#:           1
        DENSITY:           800 BPI
         PARITY:           ODD
EXPIRATION DATE:           01-NOV-78
PLEASE LABEL THIS VOLUME!
*
BEGIN Complete
```

USAGE

When run, BEGIN displays:

```
BEGIN>
```

as a prompt for entry of the name of the command file to be executed, an optional wild card string replacement, and three logical device names for device assignment. CCL entry is also supported. The format of the user entry is:

```
cmndfile,wildcrd,logdevI,logdev0,logdevL
```

where cmndfile is the name of the command file to be executed; wildcrd is the wildcard replacement string; logdevI, logdev0, and logdevL are the the physical devices to be associated with logical devices I:, 0:, and L:. All parameters specified must be separated by commas. Note that this prohibits the use of commas in the wildcard string. The default command file extension is 'CTL'. The default wildcard string is null. There are no default parameters supplied for device assignment. If a wildcard replacement string is supplied for substitution, this string will be used to replace to all occurrences of '?' in the command file being executed. If device names are supplied for logical assignment, they are assigned to the logical names I, 0, and L. This permits one wildcard replacement string and up to three logical device assignment names to be passed to the command file being executed. All occurrences of a single '^' in the command file will be replaced by a control Z character. A '^' followed by an ASCII capital letter will be converted to the equivalent ASCII control character. Null parameters are allowed. BEGIN is capable of detecting the entry of a control C on the terminal and terminating without executing the remainder of the command file.

OPERATION

The sequence of operations performed by BEGIN is nearly identical to those performed by BUILD. Normal operation of BEGIN is as follows:

1) Determine the job number and keyboard number of the job initiating BEGIN.

2) Issue command entry prompt, input command line, and parse to extract parameters (unless invoked by a CCL command and the parameters were passed in the CCL command).

3) Copy the command file to a temporary file. The temporary file is then closed, re-opened for input, and marked for deletion at completion.

4) BEGIN then detaches from the user's terminal.

5) Running detached, BEGIN then logs in the terminal (under the same account number) from which it detached.

6) Any logical device assignments specified in the parameter list are then forced to the terminal.

7) All commands contained in the command file are forced to the terminal.

   Command lines beginning with a '!' in column one are broadcast (rather than forced) to increase the processing speed of these comment lines. All occurrences of a single '^' are converted to control Z characters; all occurrences of '^' followed by a valid ASCII letter are converted to the equivalent ASCII control character. Execution continues at this step until the end of the command file is reached or a control C is entered on the terminal by the user.

8) If a control C is detected, an appropriate message is broadcast to the terminal and the detached job kills itself.

9) When the end of the command file is reached, a message is broadcast to the controlled job indicating completion and the detached job then kills itself.

AVAILABILITY

Copies of this document and the source of the APPEND file have been submitted to the RSTS SIG Library. The SYSBAK program used to generate the indirect command files for system backups using BEGIN has also submitted to the SIG Library.

TEDIT: A SIMPLE ALTERNATIVE

Donna Portz
Academic Computing Services
Arizona State University
Tempe, Arizona 85281

ABSTRACT

TEDIT is an ideal text editor for students and novice computer users. It's appeal is derived from its simplicity, ease of usage, line orientation, and minimal number of commands. Enhancements to the original version have made this editor more versatile without losing its simplicity. Users can learn it with minimal time investment and optionally utilize features characteristic of more sophisticated editors.

INTRODUCTION

Text editors should be at sale prices these days due to their abundance on many computer systems. For instance, the PDP 11/70 system at Arizona State University emerged with EDIT, TECO, SOS, TEDIT and, more recently, EDT, the DEC Common Editor. But numbers alone are secondary to the practice that every editor written tries to outdo its predeccessors in capabilities. Learning some text editors is like learning a higher level language, with READ, WRITE, and DO-loop command equivalents being just a taste of what lies ahead for the unsuspecting user.

WHY TEDIT?

At Arizona State University, three PDP 11/70 computers running RSTS/E are used primarily for classroom instruction with the majority of classes learning a programming language. Since ASU has no batch facilities into these machines, all input must be accomplished through timesharing terminals. Programs and data in Fortran or Cobol must be entered via a text editor. For many classes, use of terminals and text editing is a new experience from the batch processing procedures they previously used. Instead of learning to keypunch, students now must learn to edit. It was imperative that editing be an easy task and maintain a low profile as a means to the final end product, not burdening the learning process.

With all the editing power that was available, a choice had to be made. Which editor was the best for students and novice users? It had to be simple to learn, easy to use, preferably line oriented, and contain a minimal number of commands to do the job. TEDIT fit the requirements best.

INTRODUCING TEDIT

Still available through the DECUS library as EDIT, TEDIT, was written originally by William H. Blake of Purdue University. Enhancements were added by Brian Nelson of the University of Toledo, and Rick Catron of ASU to make the version of TEDIT currently used at ASU. The remainder of this discussion describes ASU's TEDIT.

Signing On

TEDIT is easy to enter when creating a file for the first time. In the example below the user response is underlined:

```
TEDIT
Tedit Version 6.5
Filename? LAB1.FOR
Creating file LAB1.FOR
*
```

When a previously created file is to be edited, the signon procedure is identical except that the output line "Creating file..." becomes "Editing file...". Thus the student always knows whether the filename given previously exists or not. This feature was added at ASU. The asterisk (*) indicates TEDIT is ready for commands.

Command Overview

TEDIT has seven commands that allow the user to accomplish any editing task. They are: INSERT, LIST, DELETE, SEARCH, CHANGE, REPLACE, and END. Some of the commands may be combined to imitate more sophisticated manipulations typical of larger editors. These features will be discussed later. Use of TEDIT commands is illustrated in Figure 1.

All TEDIT commands have the same general format to lessen the confusion:

```
command name          lines/filename
```

For example:

```
LIST  10
DELETE  7,9
INSERT  20/NEW.DAT
```

Command names may be the complete name or a one letter abbreviation which is the first letter of the command name, e.g. I for INSERT, L for LIST, etc. The minimal form of the command allowed is just a command name or abbreviation. The lines

and filename field are optional depending on the intended use.

Line Numbers

TEDIT line numbers may be specified in 3 ways:

a)	(blank)	no line number given
b)	n	a single line
c)	n,m	a group of lines

Format a) assumes all lines or the entire file. Format b) allows one line to be referenced, and Format c) an inclusive range of lines beginning with "n" and ending with "m". Line numbers cannot exceed the file's range with one exception. The user may append lines to the end of a file by specifying a line number one greater than the last line. In addition, the letter 'L' may be used to represent the last line number of the file, which prevents the user from giving an out-of-range line number and receiving an error, as in:

```
LIST  25,L
```

Both the last line plus one and 'L' specifications are enhancements to the original version. Use of the filename field will be discussed later.

Editing Modes

TEDIT has two operational modes: command level (or editing) mode and text insertion. The INSERT and CHANGE commands automatically put the TEDIT user in text insertion mode (see Figure 1). TEDIT provides line numbers as each line is entered, however, the line numbers do not remain with the file upon exit from TEDIT.

Positioning

TEDIT has no pointer to reference. It can be assumed that the user is positioned at the beginning of the file after each command is completed. Lines are renumbered immediately after a command is executed. There is no relative positioning; absolute line numbers are used to reach the desired line of text.

Command Functions

Please refer to Figure 1 for examples of command usage.

INSERT is used to enter new lines of text into the file. Only the command or abbreviation (I) is used when first creating text lines. When a line number is specified in the command, the new text is inserted before the line specified. After typing INSERT, TEDIT prints a line number and a greater than character ( ), referred to hereafter as a "prompt" character, and waits for the input line to be typed. TEDIT continues to solicit new lines by printing line numbers in sequential order followed by a prompt character. The process is terminated by a Control/Z. Note the Control/Z character is not entered in the file. Line renumbering occurs immediately and the user is left in command mode (*). As mentioned previously, new lines may be appended to the end of the file by specifying a line number one greater than the actual last line number.

LIST allows the contents of the lines specified to be printed at the terminal unless a filename is given in the command. The entire file may be listed by omitting the line number field. Control/O is used to terminate the output prematurely.

DELETE is used to eliminate lines no longer needed in the file. A user will generally utilize line numbers with this command. Without line numbers specified, DELETE will eliminate the entire file. To prevent accidental destruction, TEDIT asks the user to confirm this situation with a "YES" reply. If the file is deleted, TEDIT will terminate. Lines are renumbered immediately after selective deletion occurs.

CHANGE provides the TEDIT user with a combination DELETE and INSERT sequence. The lines specified are deleted first, then, TEDIT solicits the user for replacement lines (similar to the INSERT command). Insertions are terminated by a Control/Z. Line renumbering occurs immediately after.

REPLACE is used to replace character strings in a file with a new string. After receiving the command, TEDIT responds with "Old character(s)?". The user types the characters he wants replaced. Then TEDIT prints "New character(s)?" and waits for the replacement string. Each line in which a replacement occurs is printed.

SEARCH allows the TEDIT user to locate a specific character string. When the command is typed, TEDIT prints "Character(s)?" and waits for the user to supply the target string (including blanks if needed). The lines containing the matched string are printed at the terminal unless a filename was given in the command.

END command terminates TEDIT upon completion of editing. It is used without the line number and filename fields. If the file being used was just created in this edit session, TEDIT is exited immediately. If a file was created in a previous session, TEDIT responds with "Output file?". The user types a carriage return if he wants the updates to replace the previous text in the file originally edited. The original input text may be retained if the user specifies a different filename to hold the edited text. This is particularly useful if the user has made a bad editing error and wants the original text back. In this case the output file can be deleted. END should always be used for a normal exit since TEDIT edits a temporary file and copies the contents to the user's file when END is used.

Alternate File Usage

The filename portion of TEDIT commands enhances the capability of TEDIT. Depending upon the command, the filename given may act as an alternate input or output file. With the INSERT and CHANGE commands, it acts as an input file, that is the contents of the file are inserted into the file being edited before the line specified. E.g. INSERT 15/NEW.DAT would insert the contents of NEW.DAT before line 15. The filename field acts as an output file with LIST, DELETE, and SEARCH. No output lines are printed at the terminal when alternate files are used.

Special Commands

Four specialized commands have been added to the original version:

?L or ?LAST LINE	returns the line number of the last line of the file
?F or ?FILENAME	returns the name of the file currently being worked on
!HEADER	produces a heading showing column numbers. No abbreviations are allowed.
!TIME	displays the current time of day. No abbreviations are allowed.

The HEADER and TIME commands may be used at command level "*" or while inserting text.

Simulated Commands

A "move" and "ditto" command may be effected by using pairs of the basic TEDIT commands along with an alternate file specification. A move is accomplished, for example, by:

```
DELETE   25,30/MOVE.TXT
INSERT      50/MOVE.TXT
```

In the above example lines 25 through 30 were "moved" to precede line 50 with the aid of an alternate file. Similarly a "ditto" is performed by:

```
LIST   25,30/DITTO.TXT
INSERT    50/DITTO.TXT
```

In this example lines 25 through 30 were repeated just before line 50 in the file being edited.

Additional Comments

The following comments pertain to TEDIT usage:

- Editing a previously created file is best accomplished from the bottom up since line renumbering occurs after a command is performed.
- Control/Z may be used to return to command level "*" when in suboption level, e.g. when TEDIT is requesting character strings for the SEARCH and REPLACE commands. Also, Control/Z will return the user to command level when "Output file?" is solicited by TEDIT.
- When an input file lacks an extension, TEDIT searches the user file directory for a file with the same filename and a null extension. If a match is not found, the user's directory is searched from the beginning for a file of the same name but with a valid source file extension (APL, B2S, BAS, CBL, CMD, CTL, DAT, DOC, FOR, FTN, MAC, SRC, TXT). If a match is found, TEDIT opens the file for editing. If no match occurs, TEDIT will create a new file.

ASU TEDIT Details

TEDIT was converted to BASIC-PLUS 2 for increased efficiency. It executes approximately 40% faster than when it was run under BASIC-PLUS (Personal Communication Rick Catron). Storage requirements are about 11K under BASIC-2. Under BASIC-PLUS (extend mode) it required about 13K. TEDIT will handle a maximum of 256 characters per single line and allows variable line lengths with a maximum capacity of 246,000 characters.

CONCLUSIONS

TEDIT meets the requirements of an easy to use student text editor. Its simplicity allows even novice users to learn it quickly and readily. Because it is line oriented, the author thinks it adapts well to the type of usage it receives at ASU. It is less confusing than character oriented editors.

The author found the DEC Common Editor was too complex for student use. Its command formats and abbreviations are variable; some command functions are redundant and unclear in absolute function; and the subcommand level for string manipulations is far too sophisticated for novices. The character editor, EDIT, is awkward in handling buffers and keeping track of "Dot" requires as much time as the editing session itself.

TEDIT can be utilized in its most elementary form by means of seven basic commands. Additional editing power is an optional feature for those who choose to utilize it. A minimal investment in time is required.

REFERENCES

1. DEC EDITOR Reference Manual, Digital Equipment Corporation, Maynard, Mass., 1977.
2. PDP Information Packet, Copyright 1978 Arizona Board of Regents, Academic Computing Services, Arizona State University, Tempe.
3. RSTS/E Text Editor Manual, Digital Equipment Corporation, Maynard, Mass., 1977.

ACKNOWLEDGEMENTS

The author wishes to thank Jim Brodie of ASU for his helpful ideas and notes in preparing this manuscript. Thanks also are due to Rick Catron, formerly of ASU, for information on enhancements made to TEDIT and to Brian Nelson, University of Toledo for his contributions to TEDIT. Lastly, thanks are in order to William Blake of Purdue, without whom this paper would not have been presented.

```
TEDIT
Tedit Ver. 6.5
File name? GETTYS.ADD
Creating file GETTYS.ADD

*I
Type CTRL/Z to stop insertions.
     1 >FOUR SCCORE AND SEVEN
     2 >YEARS AGO OUR PHOREFATHERS
     3 >THIS IS AN EXTRA LINE
     4 >BTROUGHT FORTH ON THIS
     5 >CONTINENT A NEW
     6 >CONCIEVED IN LBIERTY AND
     7 >DEDICATED TO THE PROPOSITION
     8 >^Z
*END

Ready



TEDIT
Tedit Ver. 6.5
File name? GETTYS.ADD
Editing file GETTYS.ADD

*L
     1 >FOUR SCCORE AND SEVEN
     2 >YEARS AGO OUR PHOREFATHERS
     3 >THIS IS AN EXTRA LINE
     4 >BTROUGHT FORTH ON THIS
     5 >CONTINENT A NEW
     6 >CONCIEVED IN LBIERTY AND
     7 >DEDICATED TO THE PROPOSITION
*I 8
Type CTRL/Z to stop insertions.
     8 >THAT ALL WOMEN ARE CREATED EQUAL
     9 >^Z
*R 6
Old character(s)? bi
New character(s)? ib
*L 6
     6 >CONCIEVED IN LBIERTY AND
*R 6
Old character(s)? BI
New character(s)? IB
     6 >CONCIEVED IN LIBERTY AND
*R6
Old character(s)? IE
New character(s)? EI
     6 >CONCEIVED IN LIBERTY AND
*C 5
Type CTRL/Z to stop insertions.
     5 >continent a new nation^U
CONTINENT A NEW NATION
     6 >^Z
*
INVALID COMMAND
*R 4
Old character(s)? TR
New character(s)? R
     4 >BROUGHT FORTH ON THIS
*D 3
*R 2
Old characTer(s)? PH
New character(s)? F
     2 >YEARS AGO OUR FOREFATHERS
*R 1
Old character(s)? CC
New character(s)? C
     1 >FOUR SCORE AND SEVEN
*L
     1 >FOUR SCORE AND SEVEN
     2 >YEARS AGO OUR FOREFATHERS
     3 >BROUGHT FORTH ON THIS
     4 >CONTINENT A NEW NATION
     5 >CONCEIVED IN LIBERTY AND
     6 >DEDICATED TO THE PROPOSITION
     7 >THAT ALL WOMEN ARE CREATED EQUAL
*S
Character(s)? WOMEN
     7 >THAT ALL WOMEN ARE CREATED EQUAL
*R 7
Old character(s)? WO
New character(s)?
     7 >THAT ALL   MEN ARE CREATED EQUAL
*
R 7
Old character(s)?   MEN
New character(s)? MEN
     7 >THAT ALL MEN ARE CREATED EQUAL
*END
Output file?

Ready
```

FIGURE 1.

PERFORMANCE MEASUREMENT OF TERMINAL-ORIENTED SYSTEMS

Marc A. Dashevsky and Thomas G. Evans
Evans Griffiths and Hart, Inc.
Lexington, Massachusetts

ABSTRACT

Computer system performance measurement can play a number of useful roles, but to measure the performance of a terminal-oriented application it is necessary to be able to apply a terminal input load to a number of lines in a reproducible way. DIALOG is a RSTS/E program developed for this purpose. Its operation is discussed and some examples of its use are illustrated.

Computer system performance measurement can play a number of useful roles. It can aid in "tuning" a system running a single application or a set of concurrent applications for improved response time or throughput. It can be useful in determining hardware "bottlenecks" and pointing toward changes in system configuration to alleviate them. For a software product, it can provide valuable data as to the capacity of the product to cope with a specified workload in a specified operating system/hardware configuration environment. RSTS/E has a quite useful set of facilities to aid in the collection of such performance information, primarily the "monitor statistics" sysgen option and the corresponding STATUS utility for exhibiting the data thus collected in a convenient form. A useful introduction to these facilities is contained in an article on performance evaluation by Rich Marino in the RSTS-11 SIG Newsletter (vol. 5, No. 3) for May, 1978. RSX-11M has no such facilities available as part of the operating system, but there exists at least one "task accounting package" (ACCLOG, DECUS 11-329), which runs under RSX-11M and permits gathering at least a subset of the performance information available from RSTS/E.

Given these tools (and a great deal of care, patience, and cross-checking; interpretation of performance statistics can be a quite tricky task) what else is necessary to put an application through its paces in a controlled, reproducible way? Since many applications tend to be terminal oriented, we must have a convenient means of placing a specified terminal input load on a system. A number of facilities to permit such terminal input simulation have been developed for various computers and operating systems. For example, the SCRIPT program (part of the User Environment Test Package included on the RSTS/E V6C kit) is a facility for controlling programs through simulated terminal input contained in a sort of command file, or "script", and fed to the program being controlled via the RSTS/E pseudo-keyboard facility.

Our requirement, however, was for a mechanism by which one system could be used to simulate terminal input on a number of lines into another system on which only the application of interest is running, so that clear-cut performance results could be obtained. What follows is a description of a program called DIALOG which was developed for this purpose and some examples of performance results obtained with it.

DIALOG is a BASIC-PLUS program which uses the multiple-terminal feature of the RSTS/E terminal service to put characters onto output lines which are plugged into terminal ports on the system to be driven, and to receive as input on those lines the terminal outputs of that system. The specification of the characters to be sent out over the lines being controlled by DIALOG and those that are expected to be received is contained in "DIALOG text files" (.DTF), one for each line. (Note: the driving system and driven system may be the same. This may often be useful for test purposes, though it is normally quite difficult to disentangle the performance to be measured from the other activity on the system.)

DIALOG starts by looking at a specified "DIALOG control file" (.DCF) which indicates which lines (i.e. keyboard numbers on the driving system) are to be used, the name of the .DTF file to use for each, and an average input rate (in characters/second) which DIALOG should attempt to maintain for that line. A .DTF file consists of commands indicating what text and control characters are to be sent on its associated line by DIALOG, intermixed, as desired, with commands specifying that DIALOG should stall execution of the commands in the .DTF file at the current point until a specified pattern is found in the character stream that DIALOG is receiving on that line. This provides a means of synchronizing as required the execution of DIALOG and the program or programs being driven. There are also facilities for specifying both "overlapped" and "non-overlapped" delays, simulating several kinds of "user think time", though these delays are not used in the performance measurement experiments described below. The .DTF files may also contain repeat and go-to commands as well as commands turning on or off writing to a log file kept by DIALOG. Since a single DIALOG job uses one I/O channel for multi-terminal access, and another channel for both the .DCF file and the log file, this leaves up to ten channels on which .DTF files may be open. So, although DIALOG could conceivably control as many as 127 lines (the RSTS/E terminal limit), there is still a limit of ten unique "dialogues" that may be carried on over these lines. It is also possible to specify "begin" and "end" .DTF-type command files to DIALOG to be executed before and after the test run specified in the .DCF file and its associated .DTF files. This makes it possible to execute, say, SYSTAT or STATUS on the system being driven.

DIALOG has a variety of uses; the first version was developed at EGH to simulate an input load to a system from a customer's special-purpose terminals. More recently we have been using the current version to place an input load on KDSS, EGH's key-to-disk data entry software package, which exists in both RSTS/E and RSX-11M versions, and expect to put it to a variety of other uses. At the time of writing, we have done a number of experiments in driving KDSS under various terminal input loads on an 11/35 running RSTS/E or RSX-11M. DIALOG itself, as we have noted, runs on RSTS/E but there is, of course, no reason why the system being driven needs to be a RSTS/E one or even a PDP-11 as long as port-to-port connection is possible (by, say, a null modem connecting two EIA ports).

Performance measurements with DIALOG on KDSS are of interest to us for several reasons. First, it places us in a position to respond to performance inquiries with reliable data which is virtually impossible to obtain in any other way. Customer experience with KDSS performance is difficult to extrapolate to new situations because so many things vary, not least because KDSS is normally running concurrently with one or more customer-written applications with which we are unfamiliar. Second, a range of performance experiments permits us to vary a number of parameters and examine their effect, information that gives us a better feel for the performance costs and benefits of various ways of using KDSS. For example we conducted an experiment entering a number of records with a particular data entry format, following this with the same experiment modified only by removing from that format all the prompting text it normally displays, in order to determine by how much CPU load and disk accesses to the format library would diminish. In this case it turned out to be substantial, on the order of a 25% saving in both, which is the kind of information helpful to a KDSS user making the tradeoffs involved in format design.

A typical run from our experiments with KDSS under RSTS/E using DIALOG consists of (apart from setup of the system-to-system connections and other preliminary details) starting up the necessary jobs, getting a SYSTAT report, getting a STATUS report (itself of no special interest), executing the KDSS experiment specified in the DIALOG control files (e.g., simulating operators at four terminals, each entering a 50-record batch using a specified format and entering specified data into its fields), getting another STATUS report from the STATUS job which has been sleeping since the last one, getting another SYSTAT report, and getting KDSS's own batch status report, operator statistics report and log. The printout of these reports forms rather complete documentation of the run. The most informative portion of this data is normally the second STATUS report which provides information on CPU and disk usage, among other things, in the system being driven over the interval since the initial report from STATUS. During this period we insure that only KDSS is running so that we get "pure" performance numbers.

The procedure used under RSX-11M is similar to that just described for RSTS/E. ACCLOG is started and provided with the names of the tasks about which it is to collect statistics (i.e. the KDSS data entry task(s) and the KDSS file handler FILTSK). These tasks are run and data entry is begun. When data entry is completed, the tasks are terminated and ACCLOG is shut down by running ACCOFF. The file which contains the statistics accumulated by ACCLOG is printed, along with the KDSS batch status report, operator statistics report, and log.

Since, among other things, we want to compare the performance of the RSTS/E and RSX-11M implementations of KDSS it must be determined whether the two different methods of gathering performance statistics are indeed comparable. Both STATUS and ACCLOG report what percentage of the time interval being measured is spent by the CPU (1) executing monitor code, (2) executing user code, and (3) in an idle state (STATUS also breaks down the monitor usage into further categories). STATUS reports the frequency of disk accesses over this interval. Although ACCLOG does not report disk accesses, it collects the number of QIO's issued by each task being monitored. Since STATUS and ACCLOG both gather CPU usage information at each clock tick, it appeared likely that this information would allow valid comparisons. This was confirmed to our satisfaction when similar statistics were obtained by both methods after running identical CPU-bound test programs under the two operating systems. It should be noted that information collected only at clock ticks has the potential for being misleading when applied to jobs/tasks which get into or out of synchronization with the clock in some manner. Cross-checking results of experiments under various conditions for consistency is recommended before placing too much confidence in the results of any single run. That said, we'll conclude with results of two representative runs which are consistent with a set of other experiments under varied conditions.

The machine being driven is a PDP-11/35 with 120KW of memory, RK05 disks, and a DH11 terminal multiplexer. It runs unmodified RSTS/E V6C. DIALOG was run on an adjacent PDP-11/40, also with DH11, and also running RSTS/E V6C. The experiment was a simulation of eight operators simultaneously entering 100-record batches (all eight terminals being handled by one KDSS job) using a data entry format called EXAMPL, each typing in excess of 12,000 keystrokes/hour, a quite respectable data entry rate. We consider EXAMPL representative in terms of prompting text, field edits, etc., and in fact include it in the KDSS kit as a sample format. Standard KDSS version 3 was used. The only "tuning" steps taken were making sure the KDSS batch and format files on the 11/35 were contiguous and increasing the output buffer chain limit on the 11/35 for each of the eight lines being driven to 24 to avoid output stalls ("TT state") by the KDSS job. The lines were set to 9600 baud out from the 11/35 and 110 baud in to it in order to space the arrival of characters as much like a typing rate as possible. Briefly, the results shown by STATUS for the run suggest that even with the relatively slow RK05's on the 11/35 the process would run out of CPU capacity before disk capacity; only 2.3 disk accesses/second were taking place during the run. Under other circumstances, however, such as frequent access by KDSS data entry formats to auxiliary jobs which do heavy file lookup, this situation could well be reversed. As for CPU usage, the results were:

5%	User running
21%	SYS charged
9%	SYS uncharged
0%	Lost
65%	Idle
100%	Total

or, 35% of the CPU time was being consumed, 30% in the RSTS/E monitor on behalf of the KDSS job (presumably mostly in input and output processing associated with the eight terminal lines) and only 5% of it in execution of code (all in MACRO-11) in the KDSS job itself. Finally, the average total character rate over the eight terminals reported by STATUS for the period of the run was 27.8 characters/second input (simulated typing) and 189.8 characters/second output (echoing of typed input, prompting text, cursor positioning, etc.).

The above simulation was repeated with DIALOG driving the same configuration running RSX-11M. The experimental conditions were identical to those of the RSTS/E simulation with three exceptions. First, the RSX-11M operating system used was standard version 3.1 except that a set of modifications contained in the RSX-11M KDSS kit were incorporated into it at sysgen time, principally to permit buffering of type-ahead, an essential for high-speed data entry. Second, the notion of a monitor output buffer chain quota does not apply to RSX-11M, but since KDSS under RSX-11M handles its own terminal output buffering, the KDSS task's own buffer pool was set to a sufficiently large value (60 128-byte segments) to insure against output stalls. Third, ACCLOG rather than STATUS, was the performance measuring process monitoring system activity. CPU-usage results were as follows:

9%	User time
30%	Exec time
61%	Null time
100%	Total

Comparing these figures with those from the previous experiment it can be seen that overall CPU usage increased from 35% to 39% while user-mode CPU usage increased from 5% to 9%. The latter increase is not surprising considering that under RSX-11M the KDSS task is performing extensive terminal buffer management, especially on output, that under RSTS/E is handled in the monitor's terminal service. It is perhaps surprising, in view of this shift of responsibility, that we did not see some corresponding decrease in exec time.

These results should not be interpreted as a direct comparison of the performance of RSTS/E and RSX-11M. While data entry to KDSS under the two operating systems looks functionally identical to an operator and while much of the code being executed is identical, it was necessary to do a number of things differently because of differences in facilities provided by the two operating systems, so what is being compared is the performance of two functionally identical programs, each implemented with efficiency under its host operating system in mind, rather than the performance of two internally identical programs under the two systems.

RSTS/E APPLICATION LIBRARY: CONCEPTS IN STRUCTURE AND CONTENT

J. A. Hayes, Academic Coordinator
Computer Center
California State University, Northridge
Northridge, California

ABSTRACT

California State University, Northridge Computer Center supports nearly 5,000 RSTS/E users each term. There are two kinds of users, those who write elementary to intermediate level programs and those who make use of prewritten application library programs. For these library program users, there are a number of ways their interaction with the RSTS/E application library is made easy to use. For the RSTS/E librarian there are formal library procedures and a unique "library system" of programs to assist in library management.

INTRODUCTION

The focus of this presentation will examine some unique concepts in both structure and content of the RSTS/E program library:

1. The Use of a CCL LIB to Access Programs - To eliminate users from having to know the location of the programs, i.e., the PPN's of the file names.

2. The Concept of a Library System - A group of programs to provide library maintenance, user information programs (the on-line INDEX and SAMPLE EXECUTIONS) and library statistics.

3. Library Procedures - Formal library procedures to "keep track" of activity via logs and program history books.

4. The Content of the Library - High quality, user-proof programs provided by stringent, user-oriented Timesharing Library Program Standards and thorough testing procedures.

These concepts are applicable and transportable to different sites, both educational and commercial.

HISTORY

The California State University, Northridge Computer Center academic support staff has had more than 10 years experience with application libraries used by non-programming users. With a background in 2 timesharing systems (GE 435 TIMESHARING, dual CDC 3170 ITS TIMESHARING) and one batch (CDC 3170 MASTER) system where non-programming users have needed to access 140 to 300 prewritten library programs, we have, over the years developed well-defined concepts in structure, procedures and usability of these libraries.

With the arrival of the PDP 11/45 operating under RSTS/E in October 1976, we were faced with some unique problems and unique capabilities for which we developed solutions unique to the system's capabilities. Concepts in library procedures and content were directly carried over from the previously established timesharing systems. Certain user-provided information, such as a library index and sample executions, were put on-line for enhanced functionality and updating flexibility.

USE OF A CCL LIB TO ACCESS PROGRAMS

The Problem

On previous timesharing systems, library programs were entirely "independent" from the account number of where the programs resided. Users did not previously need to know account numbers of the program files. It was quite a surprise, of course, to discover that on RSTS/E the user would have to include the PPN of the library program name for access.

Background

By the time RSTS/E was installed, nearly all the programmers from our previous statewide timesharing system, which was supported on our site, had been transferred to the central facility to support both RSTS/E and the incoming new statewide system, a CDC CYBER 173. In all their infinite wisdom, the central facility had neglected to procure a support computer for their own staff while offering to support 19 campus RSTS/E operating systems and instructional support functions. The result was that our old friends, our ex-staff members used our campus PDP 11/45 RSTS/E system to do development work on. Work was done in the dark of night after our campus users were off the machine. My compliments go to Tom Hohmann who wrote most of the programs and to Glenn Dollar who did all the documentation[1]. Likewise, my compliments go to members of my own staff, Pat Kleinhammer and Steven Stepanek, who wrote other modules of this package. The design and resulting structure was done by the high volume, highly vocal, long argumentative method. The results have produced a stable, well-written, excellently documented package that runs through several versions of RSTS/E (V5C, V6A, V6B and V6C).

The Solution

The prime objective was to make life easy for the

non-programming user. The application library system for RSTS/E was designed and implemented to allow users to access the programs in the application library by name, eliminating the need for the user to know the location or PPN where the program resides. To run a program the user merely types the CCL LIB followed by the program name he wishes to access.

THE CONCEPT OF A LIBRARY SYSTEM

The secondary objective was to make life easier for the RSTS/E program librarian. The system of programs behind the LIB CCL:

- Provides the linking mechanism from the time the user types "LIB program-name" to the execution of the program.
- Provides library maintenance functions such as adding, changing and deleting of library programs.
- Provides a library utilization program to collect statistics on program usage.
- Provides an on-line index of library programs and sample executions of any or all programs for the user.

The library system consists of four programs, LIBRUN, LIBMAN, LIBIND, and LIBRPT and one or more data files. LIBPRG.DIR, is a directory to the application library and LIBIND.BLK is the index block used by LIBIND. LIBPRG.DIR contains the Call Name and a Chain specification for programs in the library and information used by the index program, LIBIND.

LIBRUN - The Linking Program

The LIBRUN program is called by the "LIB" CCL and performs the following functions:

1. Compares the program name entered by the user with the Call Name entries in LIBPRG.DIR.
2. Upon finding a match collects the appropriate statistics, if the statistics option is enabled, and
3. Chains to the program requested by the user.

Statistics, if enabled, are collected into a data file, LIBDAT.ymm, where y = last digit of the year and mm = a 2 digit numerical month. LIBRUN automatically creates this file at the beginning of every month. The file is pre-extended to 40 blocks allowing for 1209 entries. When LIBRUN normally records statistics, LIBDAT.ymm is opened in update mode. If LIBDAT.ymm is full, the file is opened in non-update mode and is extended 10 blocks. The statistics consist of an entry for each call to the library containing the program name, the date accessed, the time accessed, and the PPN of the caller.

To simplify the design of account independent programs LIBRUN puts the Chain specification into core common in a fixed format. This allows a program to retrieve the PPN and filename of needed data files or overlays.

LIBMAN - The Library Maintenance Program

LIBMAN, the library maintenance program, allows the librarian to make modifications to LIBPRG.DIR. LIBMAN in no way affects the existance of the actual application program. LIBMAN creates a temporary file, LIBPRG.VRL which is deleted by the EXIT function. All modifications are made to the temporary file. All input may be abbreviated to the first three characters. The following commands are available:

ADD

Allows the librarian to add a Call Name and an associated Chain specification to LIBPRG.DIR. The ADD function asks for a Call Name "CALL?" and a Chain specification "CHAIN?". The librarian should respond with the name used to call the program, maximum of 6 RAD50 characters, and the location of the program in the format of a legal file specification. The program then asks for Category, Language and Abstract Input File.

The Legal Categories are:

```
BIOLOGICAL SCIENCES
BUSINESS
CHEMISTRY
DEMOS AND GAMES
EDUCATIONAL APPLICATIONS
ENGINEERING AND COMPUTER SCIENCE
MATH AND STATISTICS
PHYSICS
POLITICAL SCIENCE
SCIENCES, OTHER
UTILITIES
```

The Legal Languages are:

```
BASIC-PLUS
FORTRAN IV
MACRO
```

The abstract (short descriptions of the programs) input file query may be answered with any legal file specification. A carriage return indicates the keyboard. If the abstract is input from the keyboard, it MUST be terminated with a carriage return and then a Control-Z. An input file must be in ASCII format. The maximum length of an abstract is 498 characters including carriage returns and line feeds.

An entry CANNOT be added unless ALL the information is entered.

CHANGE

Allows the librarian to change a Call Name or associated Chain specification. The CHANGE function asks for "Call Name?" to which the user responds with the current Call Name he wishes to change.

CHANGE then requests the following input:

```
CALL:       <call name>?
CHAIN:      <file spec>?
CATEGORY:   <category>?
LANGUAGE:   <language>?
ABSTRACT:   <abstract>?
ABSTRACT OK?
```

The librarian may change any parameter by typing

the new parameter or leave the parameter "as is" by entering a carriage return. If the librarian answers NO to the "ABSTRACT OK?" query, the program asks for "ABSTRACT INPUT FILE?"

An entry CANNOT be changed unless ALL information is entered.

DELETE

Allows the librarian to delete an entry from LIBPRG.DIR. DELETE asks for a "Call name to delete?" After the user responds with the program name to be deleted, DELETE repeats the Call Name and the Chain specification and then asks "Delete?" to which the librarian must respond Yes or No. DELETE removes the entry from LIBPRG.DIR and nulls the entry in LIBIND.BLK.

EXIT

Sorts the entries in LIBPRG.DIR alphabetically by Call Name, updates LIBPRG.DIR, deletes LIBPRG.VRL and then exits LIBMAN. EXIT reports the number of ADDs, CHANGEs, DELETEs, and total number of entries in the library directory.

HELP

Lists legal functions.

LIST

Allows the librarian to obtain a list of all Call Names and associated Chain specifications in LIBPRG.DIR. LIST asks "OUTPUT TO?" to which the user can enter any legal file specification. A carriage return in response to the query indicates the user's terminal. LIST also reports installation date, language, and category.

LIBIND - The Library INDEX Program

LIBIND, a library index program, allows users to selectively retrieve information about programs available on the application library. The user sees this program by the name INDEX. The "FUNCTION" query allows the user to specify the method by which the library information is retrievd. The second query allows the user to specify a program name, category name, language name, retrieval date, or output filename. An optional switch may be appended to the second response to specify what information to print; the default is /S (S for Short). A carriage return in response to any question will return the legal responses.

Legal Functions (only the first three characters are necessary):

CATEGORY

Lists programs by their category (i.e. Business, Math, etc.). The keyword "ALL" retrieves all programs. Switches are available.

DATE

Allows selection of programs by library entry date. User inputs the date in RSTS/E format; DD-MMM-YY, where "DD" is the day, "MMM" is the first three characters of the month, and "YY" is the last two digits of the year (i.e. 21-JUN-76). The DATE function is used in conjunction with any other function and is in effect for the NEXT FUNCTION ONLY.

EXECUTIONS

Allows printing of a sample execution for a program on the library. Switches cannot be used by non-privileged users. Sample executions are taken from the supplied files ??????.EXE which must reside in the same account as the associated program. An undocumented (see the Program listing) switch enables the librarian to print all sample executions at the printer. These can be made available to users as a reference document and will save valuable connect time.

EXIT

Exits the program.

HELP

Prints program instructions.

LANGUAGE

Lists programs by their language (i.e. BASIC-PLUS FORTRAN, etc.). The keyword "ALL" retrieves all programs. Switches can be used.

NAME

Lists programs by their library name. The keyword "ALL" retrieves all programs. Switches are available.

OUTPUT

Directs the output for the NEXT function to a file. The file specification cannot contain a device or PPN. The OUTPUT function is used in conjunction with any other function and is in effect for the NEXT FUNCTION ONLY. A Control-C will redirect output to the keyboard.

LEGAL SWITCHES

/N	Name:	Lists only the program name or names.
/S	Short List:	Lists program name, language, and category.
/L	Long List:	Lists program name, language, date, category, and abstract.

LIBRPT - The Library Utilization Program

LIBRPT, a library utilization program, produces a six month report of library usage statistics. LIBRPT asks for the starting month and year for the 6 month report. If a carriage return is entered then a 6 month report for the current half-year is generated. The program also asks "OUTPUT REPORT TO?" which requires any legal file specification in response. A carriage return indicates the librarian's terminal.

The generated report is alphabetical by program name and contains the number of calls for each of the six months and the last access date for each program. Also included is the total number of calls

for each month and a grand total. The program is also capable of reporting the number of calls by internal accounts if all internal accounts are less than a specific Project number and the variable "P7%" is changed on line 113 of LIBRPT to indicate the first non-internal account.

Library Installation Instructions

LIBRUN is called by the CCL "LIB" and therefore must be installed, in compiled form, in [1,2] with a protection code of 232. LIBMAN, LIBRPT, and LIBIND should be installed in [3,0]. LIBMAN will also create the data files in [3,0], where they are expected by LIBRUN and LIBIND.

The installation account or location of the data files can be changed by program modification. The statistics option is disabled upon delivery and requires a program change to enable. Changeable parameters are initialized and documented in the beginning of the programs. All programs must be consistent with respect to location of programs and data files.

If an installation does not wish to use the system, wishes to change the location of any program (library system or application program), or does not wish to install ALL programs then a new library directory and index block must be created using LIBMAN. The abstract files are needed for this reason only and are NOT needed by the system for any other reason.

The RSTS/E LIBRARY is designed to use the following accounts:

ACCOUNT	PURPOSE
3,0	LIBRARIAN'S ACCOUNT
3,238	COSAP
3,239	SCIENCE
3,240	DEMOS AND TUTORS
3,241	BIOLOGY
3,242	SOCIAL SCIENCES
3,243	MATHEMATICS
3,244	EDUCATIONAL APPLICATIONS
3,245	BUSINESS
3,246	ENGINEERING
3,248	CHEMISTRY
3,249	PHYSICS

These accounts must be created before the library is installed. The program and sample execution files must be installed in the same account.

LIBRARY PROCEDURES

Apart from the library system of programs, the librarian has formal documented procedures[2] to assist him with library maintenance. These aids help him "keep track" of what's going on. The formalization of library procedures has been a result of dealing with the management of large (140 to 300 numbers of programs on both previous timesharing systems and batch systems. With a small number of library programs this kind of structure may not be necessary. The intent is to document all activities having to do with an application library.

Library Program Activity Log Procedures

Whenever anything is done to a program (addition, modification, or deletion) in the RSTS/E Application Library the following entries are made in the Library Program Activity Log:

- Date
- Program name
- PPN of program
- Name of person performing the activity

Similar entries are made in the RSTS/E Application Program History Book.

Program History Book Procedures

The purpose of the Program History Book is to supply detailed information about each program and to record detailed activities with each program. There are two different kinds of information sheets for each program:

- Program History Data Sheet

  This detailed history of the program and its characteristics is filled out when installing a new program. Information on the History Data Sheet includes:

  - Program name
  - Entry date
  - Program description
  - Characteristics:
    - Source Language
    - Program category
    - Number of source statements
    - Core
  - Source of original program
  - Original computer and system
  - Conversion programmer
  - Comments

- Program Update Sheet

  Entries on the Program Update Sheet are made whenever a change is made to a program:

  - Date entered into the library
  - Documentation change?
  - New core
  - Source program changes made
  - Who entered changes
  - Reason for change
  - Current PPN

Library Utilization Procedures

On the last day of each month the librarian runs the LIBRPT program to generate a 6-month utilization report reflecting library use for the previous month and accumulative usage to date.

The LIBRPT program can either be run directly from the terminal or run via batch using a control file which can produce an output file and issues a print request.

Timesharing Program Testing Procedures

It is the function of the librarian to thoroughly test all programs being submitted to the library. His responsibility is to make sure the programs are user-proof and meet the Timesharing Library Program Standards[3]. The objective is to produce a high quality product that won't need fixing all the time. This is a particularly high cost and time consuming activity - it may take 50 hours of testing

and program change time. The advantages gained in making sure you have good working programs by thorough testing are:

- Happy users who know what to expect in the programs.
- Reduced consulting - very little of our consulting has to do with RSTS/E library programs.
- Reduced program "fixing" activity.

More information on testing procedures will be discussed in the following section: The Content of the Library.

Updating Library Source Tapes Procedures

Due to the size of the RSTS/E Application Library, the program sources are not maintained on the system disk. As programs are added or modified, the sources are temporarily stored in the library account where the compiled files and sample executions are stored. Periodically, the sources are archived on magnetic tape for back-up and to reduce disk storage.

Two series of tapes are generated: A complete back-up of the library accounts containing all program sources and individual back-up by program categories. Program sources on the system disk are then deleted. The complete back-up tape includes the librarian accounts where there are current activities in progress (holding areas, testing accounts, DECAL lessons, and on-line documents) as well as the files for the library programs themselves including the abstracts for the Index and Sample Execution file.

Library Program Tape Log Procedures

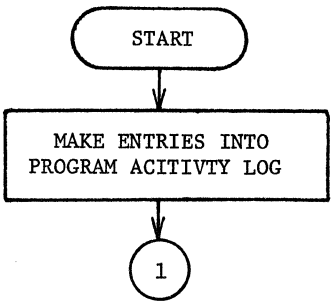
The Library Program Tape Log provides a tracking mechanism for the source tapes. Entries include:

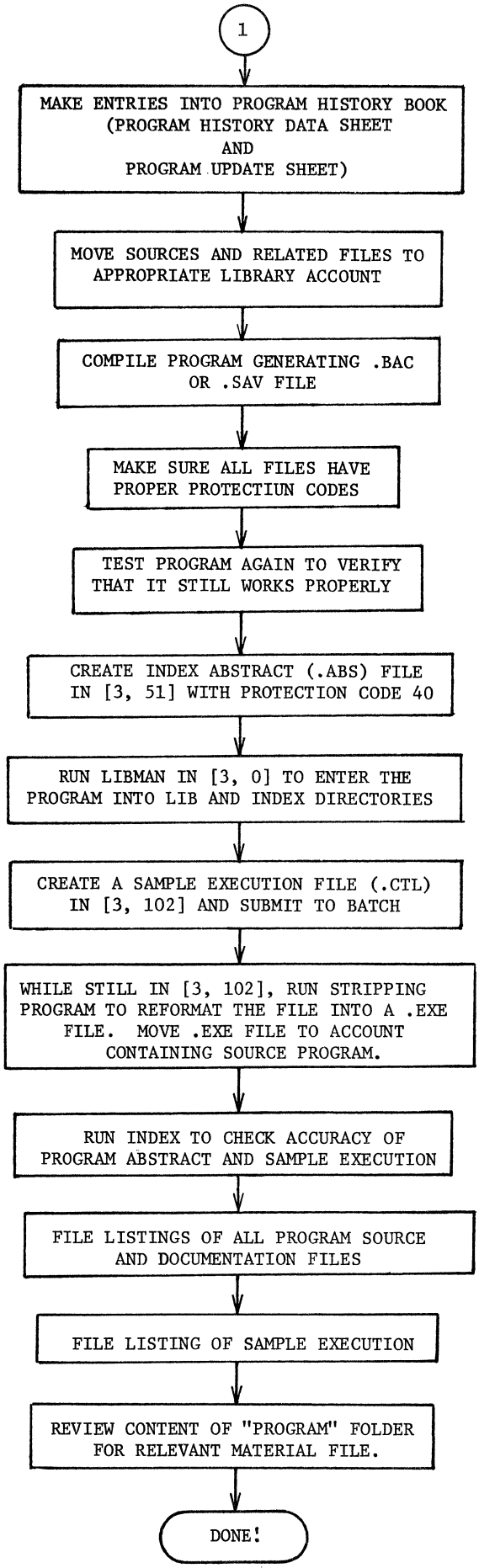
- Reel number
- File number
- Tape label description
- Date

Directories are kept with the log.

Summary of Library Procedures

By now, you think we'd run out of procedures. Some or all of these procedures can be used by a site to control the library activity. Some of the procedures perhaps are not critical - you can "fly by the seat of your pants" if you want to. Problems will arise when someone can't remember if something was done or when you change librarians, or worse, when library functions are performed by a number of people. To see the whole picture, a flowchart summary of some of the library procedures is presented (we are assuming that the program has been completely tested):





THE CONTENT OF THE LIBRARY

From our point of view, the library programs are not just any old programs some one wants on the library nor do we blindly just put programs on that we get from other sites. Essentially we're very picky about what goes on the library. Our goal is to provide high quality, good, user-proof programs that our users can rely upon.

Program Acceptance

The factors we consider before accepting a program for the library include:

- Predicted usage
  - Will the program be used in a class?
  - What is the potential number of users?
- Support
  - Is this a program we want to support?
  - Is the program submitter willing to offer support assistance?
- Standards
  - Can this program be brought up to Timesharing Library Program Standards?
  - Is the person submitting the program willing to bring the program "up to standards"?

Timesharing Library Program Standards

The document "Timesharing Library Program Standards" [3,4,5,6] is a crucial key to the quality of the individual programs that make up the content of the library. The standards were designed with the user in mind. These are not internal programming standards although we are now encouraging our programmers and users to abide by the DEC BASIC-PLUS Software Conventions[9]. Our user-oriented standards define what the user sees, the format of the program and how the program helps the user respond with the correct input information.

The structure of "Timesharing Library Program Standards" first presents standards concepts followed by system (and language) coding specifics. The primary topics include:

- Justification for inclusion in the timesharing library
- Documentation
  - Program name (header format)
  - Program description
  - Self-contained program instructions
  - Separate documentation
- Specifications
- Sample data and sample execution
- Programming conventions
  - Remarks
  - Credits
  - ANSI standards
- Requests for user responses
  - Syntax
  - Form
  - Checking the user's responses
  - Repeating boundary conditions or valid responses
  - Data entry from file to terminal, formatted or free format)
  - File handling
  - Preferred form of I/O statements
  - Program interrupt processing
- Program structure suggestions
- Program submissions procedures

Program Testing Procedures

It's not enough for a programmer to say the program is "up to standards". We've come to know better. The programmer's intent is good, but the thoroughness is lacking. The program librarian is responsible for complete checkout and testing. Sometimes very special student assistants are "trained" as program testers. This alleviates most of very time consuming "start-up" testing from the librarian and gives these student-assistants some very valuable human engineering program experience. It's one thing to write a program, and it's another thing to know how to thoroughly test it. Although program testers are used, the program librarian usually runs a complete test 2 or 3 times also. (Just checking.)

While the "Timesharing Library Program Standards" is the primary document a program is tested against, having student-assistant program testers meant that we eventually developed a document "Timesharing Program Testing Procedures". This is a "check off" list that the librarian uses when reviewing the hard copy of a program execution from a tester.

Library Content

So what's in our library? Currently there are 140 programs in the following categories:

- Biological Sciences
- Business
- Chemistry
- Educational Applications
- Engineering and Computer Science
- Math and Statistics
- Physics
- Political Science
- Other Sciences
- Utilities

Besides the very useful INDEX program which provides our users with a list and description of the library programs, there is another very special utility program. This program is called DOCUME[4,5] and allows users to select on-line publications to be printed either at the terminal/printer or on the system line printer. (During peak usage periods this program is disabled and is used only by in-house staff. Hard copies of these publications are available during these times.)[10]

Some programs have come from our users and faculty, but most of them have been converted from our previous timesharing systems. These programs have come from Dartmouth, the Huntington I and II projects, the DELTA project, from DEC, from DECUS and from other sites.

CONCULSION

This paper, "RSTS/E APPLICATION LIBRARY: CONCEPTS IN STRUCTURE AND CONTENT" has presented many ideas and helpful hints to managing a timesharing application library. Some concepts are peculiar to an educational environment but most of the concepts can

be transported to both educational and commercial sites. The use of a CCL LIB and the library system of programs can be very convenient and useful. Library procedures, even with their formality and forms, can provide you with a well-managed library. Program standards and testing procedures can enable you to produce good software products regardless of the application.

References

1. Dollar, Glenn and Academic Applications, RSTS/E LIBRARY SYSTEM, The Division of Information Systems, California State University and Colleges, and Computer Center, California State University, Northridge, revised July 1978.

2. Stepanek, Steven, RSTS/E APPLICATION LIBRARY SUPPORT, Computer Center, California State University, Northridge, July 1977.

3. Dollar, Glenn; Hohmann, Tom; Hayes, J. A.; TIMESHARING LIBRARY PROGRAM STANDARDS, Statewide Timesharing Data Center, California State University, Northridge.

4. Hayes, J. A., USER-ORIENTED PUBLICATIONS: ON RSTS/E AND FOR RSTS/E, Fall 1977 DECUS Symposium.

5. Hayes, J. A., USER-ACCESSIBLE PUBLICATIONS: HELP YOUR RSTS/E USER HELP HIMSELF, Spring 1978 DECUS Symposium.

6. Stepanek, Steven, PROGRAMMING STANDARDS; DO THEY REALLY HELP THE USER?, Computer Center, California State University, Northridge, Fall 1977 DECUS Symposium.

7. Hayes, J. A., HOW TO USE RSTS/E: A USER-ORIENTED TRAINING PACKAGE, Fall 1977 DECUS Symposium.

8. Hayes, J. A., HOW TO USE RSTS/E: HELP FOR THE USER, Spring 1978 DECUS Symposium.

9. BASIC-PLUS SOFTWARE CONVENTIONS, Digital Equipment Corporation.

10. Hayes, J. A., HOW TO PRODUCE AND DEVELOP YOUR OWN RSTS/E PUBLICATIONS, Fall 1978 DECUS Symposium.

HOW TO PRODUCE AND DEVELOP YOUR OWN RSTS/E PUBLICATIONS

J. A. Hayes, Academic Coordinator
Computer Center
California State University, Northridge
Northridge, California

ABSTRACT

The California State University, Northridge Computer Center has become well-known for both the quantity and quality of user-oriented RSTS/E publications. Educational and commercial sites have expressed strong interest in the development and production of documentation for users. This paper will describe the "how to's" involved in the production and development of readily-accessible, inexpensive publications.

The topics will cover:

1. Determining whether publications are needed.
2. Evaluation of cost-effectiveness.
3. Determining what publications should be developed.
4. Development vs. procurement from other sites.
5. Development of site-specific vs. transportable publications.
6. On-line publications vs. hardcopy publications.
7. Publication plans and production schedules.

INTRODUCTION

In the past ten years, the user population of the California State University, Northridge computer systems has grown from a few hundred users to well over 12,000 users per semester. The staff size has grown from 3 members to 6 full time staff who support these users in a multi-system environment. While there is a crew of 6 to 14 part time student assistants to assist in consulting services, there is no way to provide one-to-one assistance for the majority of the academic user population.

In order to provide users with the necessary information to use the four major computing systems, the academic support staff writes, compiles and produces a large number of publications to "help the user help himself". Although vendor documentation is readily accessible via free microfiched manuals and reference copies[3,4], the needs of the users have dictated the development of special user-oriented publications. Many of these are specially developed for the novice user. Some publications are designed as supplemental documents and still others provide otherwise undocumented information.

Of the 12,000 computer users in this university environment, approximately 5,000 are RSTS/E users. When the PDP 11/45 RSTS/E system was installed in October 1976 a simplistic, user-oriented publication, "How To Use RSTS/E Timesharing" was quickly developed and provided free of charge to first-time RSTS/E users. Within 3 weeks a "BASIC-PLUS" Instant was out of production. Knowing what the users needed and expected was determined from having experience with 2 previous timesharing systems.

Whenever a site, whether commercial or academic, installs a new computer all the problems of learning about the new system are there again. In commercial sites or small academic sites, programmers and sophisticated users can supposedly make their way through the voluminous and expensive vendor manuals. In larger academic communities with a high number of novice users each term, the learning of basics of the system must be quick and the costs must be minimized. As the number of new users grows at a much faster rate than support staff size, one means of providing these users with information is small, free or low cost publications designed to teach them basic functional commands, utilities and concepts. So where do you start? What do you consider? And how do you do it without extensive staff and development costs? The remainder of this paper discusses the answers to these questions.

DETERMINING WHETHER PUBLICATIONS ARE NEEDED

Learning With Only Vendor Manuals

In a commercial site, with professional programmers, you most certainly can buy them all the vendor manuals and let them dig out what they need to get started on the new operating system. Hopefully professional programmers know what they're looking for and have a good idea in which kind of manual to find it in. The considerations involved in this approach include:

- The cost of the manuals.
- Not knowing how many of which manuals to buy (not all the programmers need all the manuals).
- A lengthy learning curve to determine the important essentials and basic concepts.

Small academic sites can use the vendor manual only approach providing they are dealing with a small number of users. Small informal sessions can be used to "get the users started". After that they too can read the vendor manuals.

Alternative Approaches to Vendor Manuals

In large commercial sites and in large academic environments, using only vendor manuals becomes a very large problem. For academic sites some vendors (DEC and CDC, for example) are willing to negotiate reproduction rights to manuals applicable to your operating system.[2] Certain very popular manuals, like the BASIC-PLUS Language Manual can be reproduced in hardcopy form and sold at greatly reduced prices in local campus bookstores for student and faculty users. Alternately, vendor manuals can be reproduced in microfiche form and can be given away free or sold at a nominal cost (6¢-25¢) to campus users.

While both these approaches get the very expensive manuals into the hands of the users at a relatively low cost, there are still problems in reaching large numbers of users. Many faculty members are reluctant to require their students to purchase vendor manuals as class texts and there is still the cost of the hardcopy manual ($5.00-$7.00) that produces a certain amount of reluctancy on the part of a student. Microfiche, while certainly the lowest cost method of making manuals available to large number of users, carries with it the inherent problem of having microfiche readers available. While campus libraries have microfiche readers available and campus computer centers can purchase microfiche readers for user work areas, many users prefer to read manuals away from campus. High quality, inexpensive portable microfiche readers can be purchased for approximately $200 but hand held, small, even less expensive readers are still in the early stages of technology. In a few years there will be high quality, very inexpensive, very compact microfiche readers - and then student users can be expected to acquire a microfiche reader much as they do the small calculators today.

When You Need More Than Vendor Manuals

Getting vendor manuals into the hands of the users in a high turnover environment of academia still does not solve a very crucial problem: how to get the user on a system with only the information he needs to know immediately. This is the decision point in providing special user-oriented information that is a "distillation" of the huge amounts of information found in the many manuals available for only one operating system. The objective is to provide the new user with only the information he needs right away, such as:

- logging on and logging off,
- basic system commands,
- simplistic access methods to the language processors,
- simple file concepts,
- the simplest form of file manipulation capability,
- easy access to the application library programs[5] and
- any site specific peculiarities.

The types of user-oriented publications that provide this information are the system specific primers, tutorials, or what we call the "how to use" publications. At California State University, Northridge we have "HOW TO USE RSTS/E TIMESHARING" and "HOW TO USE RSTS/E EDIT" ("HOW TO USE RSTS/E EDIT" is now "AN INTRODUCTION TO TECO").

Another very popular type of publication are the pocketsized "instants". Initially we brought up a quickie titled "BASIC-PLUS INSTANT". Later we developed a "RSTS/E FORTRAN INSTANT". The "BASIC-PLUS INSTANT" has now been replaced by the DEC RSTS/E POCKET GUIDE (the RSTS/E documentation people finally got the hint).

Other types or categories of publications can be provided when the need warrants. In a previous paper, User-Oriented Publications: On RSTS/E and For RSTS/E, there is a thorough discussion of:

- Auxiliary Publications which provide supplemental or site specific information.
- Stand-Alone Documentation which is developed when there is no readily available documentation from the vendor or which provides specific program documentation.
- Indexes and Sample Executions which provide users with information on library programs.

EVALUATION OF COST EFFECTIVENESS

The Cost To Your Support Staff Without Publications

Without small packets of "distilled" information about the systems, the academic support staff in medium to large university environments, is faced with having to repeat the basic information, such as logging on, access to languages, etc., over and over again either on a one-to-one basis or in a seminar context. Each term the users ask the same questions of the staff. As mentioned earlier, the user population grows faster than the staff who must serve them. The cost is a time cost to your staff. As the number of academic users grow, certain other academic computer support services are needed. The staff must be used to support these new and fast growing service needs leaving less time for the staff to introduce new users to the systems. While services such as providing users with multimedia shows[4] can alleviate the repetitious, time consuming one-to-one or seminar sessions, the users still need something they can take away and read or something they can reference while at a terminal. Small publications "fit the bill" in this aspect and free your consulting staff to help students debug their programs, assist faculty members in course development, and provide a multitude of other academic services.

The Cost to Your Users

The prime intent of most courses that use the computer as part of their curriculum is not to spend an excessive amount of time learning how to use a particular computer system. Courses are designed:

- to teach computer concepts in general;
- to teach specific computer languages;
- to use the computer as a tool to teach concepts that would require a lot of time if done by hand or calculator (i.e., statistics, engineering applications, geographical applications, etc.); and
- to "assist" the instructor in the teaching of specific topics such as English, numerical analysis, graphics, etc. (CAI).

To this end, the users in an academic community should not be required to spend the majority of the term just learning the basics of using an operating system which might happen if just vendor manuals were relied upon. The answer to this problem is again, to provide users with just the information they need - a small publication, information that is distilled and written in a way they can understand it quickly. A lot of vendor manuals have improved in their readability, but the size in number of pages is formidable to the novice or casual user.

The Cost of Producing Publications

Every computer center manager will question the cost of producing publications for the users. The most common thing you'll hear is "we're not in the publishing business". Certainly the cost of publications must be weighed against the computer center budget and the personnel time required to produce good, useable, user-oriented publications. The objective is to reduce these costs as low as possible. On the other hand, when the cost of using staff to train large number of students and faculty about the computer systems each term becomes noticeably high in terms of time and manpower, it's time to look into alternative means to provide this service. Other academic support services may require more and more time and manpower.

The cost factors that must be considered include:

- Development costs - Writing publications "from scratch" or just bringing together appropriate materials (i.e., compiling a publication) requires time and people.

- Typing costs - Once put together, material must be typed by secretaries, clerical assistants or put on-line by staff or student assistants.

- Publishing costs - The cost of paper has increased a great deal in the last few years and reproduction facilities can be limited in some academic environments.

- Management costs - With a few publications, this is a minimal cost. Managing larger numbers of publications (10-60) requires management cost in decision making, scheduling and planning.

Needless to say, the cost impact of producing your own publications must be carefully considered. It is very easy to get in over your head very fast. Producing publications out of a campus computer center must be identified as a specific academic support service and warrants the full support of the management.

DETERMINING WHAT PUBLICATIONS ARE NEEDED

The Planned Approach

One method of determining what publications are needed is planning. Evaluate the entire situation in terms of the cost to the user, the cost impact to the staff and the budget. The users may desperately need small publications to learn about the system and the staff may need publications to "keep the users off their back" but the budget may limit the whole concept very severely. In this case, begin with only the absolute essentials that can be produced for the lowest cost to the staff and to the budget. On the other hand, if the budget is not a severely limiting factor and/or costs can be recovered through sale of small publications, then a "full blown" set of publications can be determined.

The essential considerations, after cost, are the needs of the users. Find out what are the most frequent problems or what information is requested most frequently:

- Most users will need to know how to operate the equipment they will be using, therefore a modest beginning in producing publications may be providing terminal or keypunch instructions.

- After that, the biggest problem may be access to the operating system - how to log on or how to set up JCL cards. In a multi-system environment, users may be required to access more than one system per term for a given class. The solution to this problem is to provide brief, compact, distilled information in the form of a primer or guide publication for the operating system. These are the "how to use" publications.

- When your consulting staff gets deluged by questions on a certain utility or language, it's time to consider the "instant" form of a publication. This is usually a result of too much information in a vendor manual, poorly written vendor manuals, or having to search through several manuals on a specific utility or language. Instants can be one-pages, folding cards or pocket size booklets. Sometimes the "how to use" form may be needed to supplement the "instant" form. For example, although there is a TECO Manual and a TECO Pocket Guide, an "INTRODUCTION TO TECO" may also be needed.

- When vendor documentation is not readily available (either cost prohibitive or incomprehensible) or site specific implementions differ from vendor manuals, publications of the "auxiliary or supplementary" category are warranted. Language textbooks teach the language but site specific information to access the language may be needed to use it. Certain sites may have special site specific functions or routines that are used with vendor software. Auxiliary publications are required for the users in this case.

- When there is no vendor documentation available, "stand-alone" publications may be needed. Application programs (i.e., prewritten or canned programs) for the non-programming user require stand-alone documentation also.

- When a site has a library of application programs, users need to be informed of what programs are available. An "Index" of the library, listing program names and brief descriptions, solves this problem. Users may want to see the specific program features or see how to run the program - the category of publication is "Instructions and Sample Executions". If connect time is at a premium on your site, these may be hardcopy documents. These two publication documents inherently have an updating problem if the application library has a medium to high growth rate; in this case on-line versions may be a more efficient method. Restrictions built in to programs where the user accesses either Indexes or Sample Executions (which may be voluminous) can allow the user to drop off either on a terminal or the line printer only program information that he is interested in.

Invariably user needs may suggest publications that do not fall into the above categories. What is important is to determine what is needed and to develop a publication to meet the need.

The Random Approach

The alternative to the "planned" approach is the random approach. It may just so happen that you don't consider publications as something that means you sit down and plan out. Repeated consulting problems or requests for information may mean that one day you decide that a small publication would solve the problem. Perhaps a certain instructor is giving out erroneous or misleading information and it looks like a brief publication would solve the problems for everyone. The random approach is randomly driven by random needs.

With this approach, computer center managers may find themselves suddenly "in the publishing business". Secretaries may suddenly have entirely new kinds of things they're typing - for which they have no experience in doing or that impacts the normal secretarial typing functions. Current staff loads may be overburdened by in-house copying and collating on a mimeograph machine. Or there may be heavy budgetary demands if publications are "sent out". Programmers may become writers and proofreaders instead of providing other academic services.

The random approach, unfortunately, is how most of us start. If your installation can accommodate the impact over a period of time, you'll be lucky. If you are seeing a detrimental impact on your site or if you are considering publications as a way of helping the user for a new system, the planned approach is recommended.

DEVELOPMENT VS. PROCUREMENT FROM OTHER SITES

Now we get down to the "nitty-gritty" of publication production. The needs have been determined, the cost-effectiveness has been evaluated, and which publications are needed have been determined. We're now at the point of making suggestions for "how to produce and develop your own RSTS/E publications" which is the title of this paper. The first step at this point is to determine whether you do your own development or whether you use available materials from other sites.

Developing Your Own

Certainly you know the needs of your own users and you know the capabilities of your own staff (the programmers and the typists). When you consider developing your own publications whether you "write them from scratch" or whether you collect materials and put a publication together, you need to consider the following factors:

- Technical expertise - Do I have staff programmers with the technical expertise? Do they know the system? If RSTS/E is a brand new operating system to your site, you may have a problem. If your technicians learn fast, you may have less of a problem, but with a new system you have to allow for a certain amount of time to determine what are the basics or essentials that are most needed.

- Writing ability - This is a bigger problem than you realize if you are just beginning to get into publications. As we all know, programmers are known to be notoriously bad writers. If you are lucky, you have one or two people who can communicate computer technical jargon into something users can read. If you have time, you can send someone who has shown some writing ability off to be trained. If you don't have time or money for this, the DEC pocketsize booklet, "WRITING FOR THE READER" is recommended.

- Development time - Developing new materials or compiling existing materials requires time. Judicious project management should allow you to fit in publications into staff projects. If your staff doesn't have time, consider part-time, student assistants or contract out the writing of the publications. Both situations require analysis of content and a well defined outline to work from.

The advantage to developing your own publications is that they are custom fit to your installation needs. You may also find that such publications are not available from other sites or the vendors, in which case you do it yourself.

The disadvantages are personnel time and cost for the writing, managing, and analysis. Another important consideration is quality and effectiveness of the publication.

Procurement From Other Sites

We're all into "not reinventing the wheel" if we can possibly do so. In terms of RSTS/E publications, this means getting materials from another site that are useable. Procurement of existing RSTS/E publications is done in the following way:

- Search techniques - Start asking people what they have. User conferences are a good place to begin. Ask your local vendor representative for names of educational sites. Get a list of RSTS/E EDUSIG users and start writing letters.

- Acquisition - After you find contacts, find out what they have. Ask if you can have a copy and if you can reproduce it or extract from it. If you can get "on-line" copies of documents they will allow you greater flexibility in customizing them to your own site.

- Evaluation - After acquisition, review the document to see if it fits your needs. If the publication is an extraction from a vendor manual, proofread it. Make sure the content is what you want your users to have.

- Customizing - If necessary, modify the publication to suit your installation's needs. Hardcopy publications may require additions and deletions. The document may be retyped or you can do clever "cut and paste" jobs if the copy was in high quality print. Photo-ready copies, rather than production copies may be available. On-line publications give you a greater degree of flexibility for customizing. With a text editor or a word processor package (RUNOFF, RNO), modifications are done easily.

The advantages of using already developed publications are pretty obvious. You've reduced the development time. You're not doing something that's already been done. Chances are that's it's reasonably readable for users and you haven't had to coerce programmers into being writers. You've traded these advantages for search and acquisition time and editing time.

Trading of RSTS/E publications carries with it a snow-balling effect. In your search for a particular kind of publication, you may find that the site you are making inquiries from has a lot of other items you can use.

DEVELOPMENT OF SITE SPECIFIC VS. TRANSPORTABLE PUBLICATIONS

This can be a hard decision. Your objective is to serve your users and to meet their needs. Once you get into producing user documentation, you'll find that site specific information has a tendency to change more rapidly than the standard products provided by the vendor. For example, in our early "how to's" we included hours of operation, facilities available and amount of equipment in user work areas. This information changed each term, so it was removed from the "how to" publications (which are relatively stable) and put on a one-page "Facilities Sheet" that is updated each term. The general rule is: If certain information changes frequently, separate it from the technical information on a system, utility or language publication. This will save you the cost of updating "large" publications.

When considering site specific content against the transportability of the publication, remove such items as terminal operation instructions. With the large number of new terminals on the market, you'll probably change terminals before you change computers. In this particular case, individual instruction publications for each kind of terminal is recommended. Instead of revising a large general user's guide for a system whenever you get new terminals, you simply quit producing one kind of terminal instructions and bring up instructions for the new terminal. Additionally not all users may have access to all the terminals. This way they only need to have instructions for the terminals they can access.

ON-LINE PUBLICATIONS VS. HARDCOPY PUBLICATIONS

On-line publications versus hardcopy publications will be discussed strictly in terms of advantages and disadvantages of each. The decision of providing either or both these methods of production and accessibility is dependent on a number of factors.

Hardcopy Publications - Disadvantages

- A direct noticeable cost when "sent out" to be reproduced. Small, low volume publications reproduced "in-house" tend not to have an impact on the budget.

- Increased numbers of publications and increase in size of publications predict the eventuality of a publication plan and scheduling of production.

- Printing of large numbers of copies may mean an inventory problem.

- Reduced updating flexibility. Without small stand-alone word processing units or ability to access on-line publication files, modifications and changes mean the whole publication must be retyped to update it.

- Timeliness of updating. Linked tightly to publication inventory and the ease of updating, the timeliness of updating is a crucial factor. Keep your publications current and correct.

Hardcopy Publications - Advantages

- Readily available to users - something they can easily get at and take home with them (immediate gratification).

- Can be available free or sold at a low cost in the campus bookstore.

- Highly recommended for "large" publications or frequently needed publications.

- Can make use of special typing or typesetting techniques; i.e., bold face lettering, use of italics or color inks.

- Recommended where large quantities are needed.

- Large publications can have heavy stock cover pages which increase their lifespan. (Users can be asked to "return" publications when they are finished with them.)

- Printing of large quantities reduce the cost per publication.

On-Line Publications - Disadvantages

- Large sized publications require lots of connect time to run off at a terminal. The way around this is to give the user a choice to access on-line documents either at the terminal or via the system line printer. Terminal access to large documents may be restricted to certain times of the term or times of the day.

- Impact on the system line printers. DEC furnished printers do not seem to be designed for high volume usage. Additionally, other users may want to print large program files on the printer. Large publications or those frequently accessed should be made available in hardcopy form to alleviate this problem.

On-Line Publications - Advantages

- Ease and flexibility of updating.

- Updating can be done by a programmer or a trained typist.

- More changes, "large" changes, and large insertions are more easily accommodated using computer text editors and word processing utilities. Small individual word processing units have limited capacities.

- No inventory problem. Copies are available when needed.

- Available to the user while he is at the terminal.

- Can be accessed from a user terminal/printer or can be printed on the system line printer.

- Timeliness. Because of the ease and flexibility in updating, on-line publication can be changed in a timely manner. For example, library indexes can be updated when a new program is installed.[5]

- Recommended particularly for large size, low volume publications. Certain publications may only be required by a small number of users or alternately, the size of the user body cannot be determined beforehand.

- Can be hardcopied. Once document is developed on-line or modified from another site, large number of copies can be reproduced in hardcopy. Photo-ready copy may be run out on the system line printer (upper and lower case capability should be considered) or special hardcopy, impact printer terminals (DTC, Diablo) may be used. When using the system line printer a new ribbon and white paper should be used. When using a printer terminal (not dot matrix, please), use a new print wheel or ball with carbon ribbon. Many of these terminals can be programmed to double print each character to produce exceptionally fine photo-ready copies.

PUBLICATION PLANS AND PRODUCTION SCHEDULE

Once you have more than 3 to 5 publications, you're in the publishing business. If mismanaged, you'll have irate managers, overburdened secretary-typists, frustrated programmer-writers and dissatisfied users to deal with. First, to avoid these problems, the service of providing publications to users must be an acknowledged and supported service. Secondly, a publication plan and production schedule is needed.

The Publication Plan

Everyone involved in your computer center publications should get together and lay out a publication plan. The elements of the plan should consider:

- What publications to produce.
- Frequency of updating.
- Frequency of production.
- Whether publications are typed or done on-line.
- Reproduction facilities.
- Size of publications.
- Who writes or puts together the publication.
- How long will the publication be in existence.
- Numbers of copies required for a given time period.
- Inventory storage if large number of copies are produced.
- Distribution and inventory checking.

Production Schedule

Once the size of the problem (publications are always a problem) has been defined, the key people involved such as the head secretary and the "publications project leader", can determine a production schedule. A schedule for the entire year is recommended.

Scheduling considerations should include the following:

- Publication review - Evaluation of whether it's still needed, evaluation of content, check for accuracy of content. If changes are required, schedule time to do it.
- First draft due date for typing.
- Proofreading of the first draft.
- Second draft due date for typing corrections.
- Proofing of second draft.
- Final typing.
- Final proofing.
- Schedule reproduction facilities.

CONCLUSION

It is hoped that the many facetted concept of "how to produce and develop your own RSTS/E publications" has been covered in this paper. The recom-

mendations, the suggestions and awareness of the problems have been the result of ten years experience by the California State University, Northridge Computer Center.

To provide you with a beginning in your search for existing RSTS/E publications, this paper concludes with a list of sites and publications available either on RSTS/E or for RSTS/E:

1. Computing Facility
   University of California at Irvine
   Irvine, California

   Publication:

   PDP-11 PRIMER, September 1975

2. Academic Computing Services
   Arizona State University
   Tempe, Arizona 85281

   Publication:

   PDP INFORMATION PACKET
   (TIMESHARING MADE E-Z)
   January 1978

3. Waters Computing Center
   Rose-Hulman Institute of Technology
   Terra Haute, Indiana

   Publication:

   COMPUTING AT ROSE, December 1977

4. Computer Services
   The University of Toledo
   2801 West Bancroft Street
   Toledo, Ohio 43606

   Publication:

   TIMESHARING USER GUIDE, January 1978

5. Computer Center
   Central State University
   Edmond, Oklahoma

   Publication:

   COMPUTER CENTER USER'S GUIDE, Aug. 1977

6. Computer Center
   California State Polytechnic University, Pomona
   3801 West Temple Avenue
   Pomona, California 91768

   Publications:

   LOCAL TIMESHARING INSTANT

   RSTS/E FORTRAN INSTANT (Publishing and distribution site)

   LOCAL TIMESHARING PROGRAM OPTIMIZATION GUIDE FOR BASIC-PLUS

   TELERAY TERMINAL INSTRUCTIONS (from C.S.U., Northridge)

   On-Line Publications:

   SAMPLE EXECUTION FOR FORTRAN

   SAMPLE EXECUTION FOR BASIC-PLUS

7. Computer Center
   San Francisco State University
   1600 Holloway Avenue
   San Francisco, California 94132

   Publications (Developed On-Line):

   SYS SYSTEM FUNCTION CALLS, Winter 1976

   RSTS/E RECORD I/O, June 1978

   INTRODUCTION TO TECO, Fall 1978

   INTRODUCTION TO RNO, Fall 1978

   INTRODUCTION TO EDT, Winter 1978

8. Computer Center
   California State College, Bakersfield
   9001 Stockdale Highway
   Bakersfield, California 93309

   Publications:

   HOW TO USE RSTS/E TIMESHARING (modified, from C.S.U., Northridge)

   HOW TO USE RSTS/E EDIT (modified, from C.S.U., Northridge)

   HOW TO USE RSTS/E FORTRAN

   HOW TO USE RSTS/E RUNOFF (modified, from San Francisco State University)

   EASY EDIT

   CALIFORNIA STATE COLLEGE, BAKERSFIELD COMPUTER CENTER RESOURCE MANUAL

   On-Line Publications:

   FOR HELP

   ED HELP

   HOW TO USE THE EDITOR (modified, from San Francisco State University)

9. Computer Center
   California State College, Stanislaus
   800 Monte Vista Avenue
   Turlock, California 95380

   On-Line Publication:

   COSAP I INSTANT

   Publication:

   C.S.C., STANISLAUS COMPUTER CENTER USERS MANUAL

10. Computer Center
    California State University, Sacramento
    6000 J Street
    Sacramento, California 95819

    Publications:

    RSTS/E FORTRAN IV INSTANT (modified, from DEC materials and from California State University, Pomona)

    C.S.U., SACRAMENTO LOCAL TIMESHARING INSTANT (modified, from California State Polytechnic University, San Luis Obispo)

11. Computer Center
California State University, Fresno
Shaw and Cedar Avenues
Fresno, California 93740

Publications:

HOW TO USE RSTS

USER'S BROCHURE FOR RSTS, Winter 1978

TELERAY TERMINAL INSTRUCTIONS

GUIDE TO REQUESTING COMPUTER SERVICES FOR RSTS

BASIC PROGRAMMING STANDARDS

12. Computer Center
California State College, Dominguez Hills
1000 E. Victoria Street
Dominguez Hills, California 90747

Publications:

TELERAY TERMINAL INSTRUCTIONS

DECWRITER INSTRUCTIONS

DEC EDIT

On-Line Information:

Various on-line help files.

13. Computer Center
California State University, Hayward
24800 Carlos Bee Boulevard
Hayward, California 94542

Publications:

TIMESHARING TERMINAL OPERATION

LOCAL TIMESHARING (RSTS/E)

LOCAL TIMESHARING FORTRAN

On-Line Publications:

BASIC - BEGINNING USER GUIDE TO RSTS/E BASIC

BMDP

BPCREF

COSAP

CREF

CVTFNS - GUIDE TO CVT FUNCTIONS

DIRECT

ED - COSAP DATA EDITOR INFORMATION

EDFOR - HELP FILE FOR EDFOR (THE FORTRAN EDITOR)

EDIT - HOW TO USE RSTS/E EDIT (from C.S.U., Northridge)

SYS SYSTEM FUNCTIONS (from San Francisco State University)

FORTRAN INSTANT (from CSU, Northridge)

HELPER - BASIC-PLUS SOURCE EDITOR (modified, from DECUS)

MACRO - BEGINNING USER'S GUIDE TO THE MACRO ASSEMBLER

MIOCS - MACRO INPUT/OUTPUT CONTROL SYSTEM (from California State Polytechnic University, San Luis Obispo)

NORTON (from DECUS)

TRAN - COSAP TRANSFORMATIONS

14. Computer Center
California State University, Northridge
18111 Nordhoff Street
Northridge, California 91330

Publications:

HOW TO USE RSTS/E TIMESHARING (with V6C Addendum)

TELERAY TERMINAL INSTRUCTIONS

FLOPPY DISK INSTRUCTIONS

RSTS/E EDIT INSTANT (V6A)

On-Line Publications:

RSTS/E FORTRAN INSTANT (V6A) (Development site)

BASIC-PLUS INSTANT (V6A)

HOW TO USE RSTS/E EDIT (V6A)

RSTS/E RECORD I/O (from San Francisco State University)

SYS SYSTEM FUNCTIONS (from San Francisco State University)

TECO V29 (V6C) (from DEC)

TIMESHARING LIBRARY STANDARDS

RUNOFF (modified to V6A, from DEC)

COSAP I INSTANT (from California State College, Stanislaus)

On-Line Publications (Non-RSTS/E):

HOW TO USE NOS TIMESHARING (CDC CYBER 174)

SPSS 7.0 INSTANT (from California State University, Bakersfield)

REFERENCES

1. Hayes, J. A., USER-ORIENTED PUBLICATIONS: ON RSTS/E AND FOR RSTS/E, Fall DECUS 1977.

2. Hayes, J. A., USER-ACCESSIBLE PUBLICATIONS: HELP YOUR RSTS/E USER HELP HIMSELF, Spring DECUS 1978.

3. Hayes, J. A., HOW TO USE RSTS/E: A USER-ORIENTED TRAINING PACKAGE, Fall DECUS 1977.

4. Hayes, J. A., HOW TO USE RSTS/E: HELP FOR THE USER, Spring DECUS 1978.

5. Hayes, J. A., RSTS/E APPLICATION LIBRARY: CONCEPTS IN STRUCTURE AND CONTENT, Fall DECUS 1978.

JAH:spw
10/24/78

RSTS/E System Calls for Pascal and FORTRAN

David M. Vann
Oregon Minicomputer Software, Inc.
2340 S.W. Canyon Road
Portland, Oregon 97201
(503) 226-7760

ABSTRACT

A set of routines has been developed which allows access to capabilities of the RSTS/E operating system from Pascal and FORTRAN. These capabilities have previously been available only through BASIC-Plus SYS() functions or MACRO assembly language.

INTRODUCTION

The RSTS/E operating system provides a large number of capabilities necessary for a large time-sharing system. These capabilities include: assigning and releasing special devices; setting terminal characteristics; user login, logout, and accounting; file directory maintenance; inter-job message send/receive, and many others. Some of these capabilities are used by application programmers; the more specialized features are used only by system programmers.

Programmer access to these system dependent features has been provided by extensions in the BASIC-Plus language, most notably the SYS() function calls. The SYS() calls are well documented, but not especially readable; for example, the following line returns the current job number:

```
J%=(ASCII(SYS(CHR$(6%)+CHR$(9%)))/2%)
  AND 127%
```

BASIC-Plus is also an interpretive language, and is not particularly efficient. The latest release of RSTS/E therefore includes some CUSPs (notably PIP.SAV) which are written in MACRO assembly language. Unfortunately, documentation on RSTS/E system calls from MACRO is not easily acquired; MACRO programming also requires highly skilled programmers, and can be very sensitive to operating system changes.

FORTRAN AND PASCAL

Compilers for Pascal and FORTRAN are available for RSTS/E systems; these languages are much more efficient than BASIC-Plus. Both languages also offer reasonable facilities for creating libraries of subroutines, and can access MACRO capabilities. Oregon Software has developed libraries for Pascal and FORTRAN which allow efficient access to the RSTS/E system features in a readable fashion.

Examples:

(acquiring the current job number, as above)

```
Pascal:   J:=Job;

FORTRAN:  J=JOB
```

(Deassign a device by name)

```
BASIC:
  X$=SYS(CHR$(6%)+CHR$(10%)+STRING$(6%,0%)
   +RIGHT(SYS(CHR$(6%)+CHR$(-10%)+DEV$),9%))

Pascal:   Deassign(DEV);

FORTRAN:  CALL DASSGN(DEV)
```

A more complex example shows accessing a file in update mode, waiting for other users to release the desired record:

```
BASIC:    100 ON ERROR GOTO 120
          110 GET #1%, RECORD R%
          120 UNLOCK #1%   GOTO 200
          130 IF ERR=19% THEN SLEEP 2%
          140 RESUME 110

Pascal:   IgnoreIOerror(true);
          UpdateFile(fileid,2);
          Seek(fileid,recordnum);
          IF IOerror THEN FatalIO;
          Unlock(fileid);

FORTRAN:
   110 READ(1'IRECORD,END=999,ERR=120) VALUE
       CALL UNLOCK(1)
       GOTO 200
   120 IF (IOERR .NE. 19) GOTO 999
       I = ISLEEP(0,0,2,0)
       GOTO 110
```

Examples of system calls:

Action	Pascal	FORTRAN
Echo off	Echo(false);	CALL ECHO(.FALSE)
Rename file	Rename(oldf,newf);	CALL RENAME(OLDF,NEWF)
Dismount disk	Dismount(disk);	CALL DISMNT(DISK)
Advance magtape	MTskip(fileid,count);	CALL MTSKIP(UNIT,COUNT)
Get keyboard number	K := KBnum;	K = KBNUM
Do CCL command	CCL(Command);	CALL CCL(COMMAND)
Is file a TTY?	IF TTYtype(fileid)	IF (TTYTYP(UNIT))
Get project number	J := ProjectNumber;	J = PRJNUM

EXPERIENCE TO DATE

Our experience with these routines is rather limited, but generally favorable. Several application programs (in Pascal) use the routines for simple functions such as enabling single character terminal interaction. Our timesharing accounting system has been rewritten, also in Pascal, with marked gains in clarity. Probably most interesting is a Pascal version of the directory reordering program ($REORDR) which reorders our RP06 disk in about 5 minutes, compared to 90 minutes for the BASIC-Plus program.

AVAILABILITY

The Pascal and FORTRAN libraries, with supporting documentation and some sample programs will be submitted to the DECUS library in early 1979.

X.25 PACKET SWITCHING NETWORK AND RSTS/E TIME SHARING

L. R. Irons
Tested Time Sharing Ltd.
Calgary, Alberta, Canada

ABSTRACT

The objective of this paper is to present an overview of X.25 Packet Switching Network and outline some methods of accessing this type of network. The body of this text evolved as a result of implementing a package to access an X.25 based network. Although the X.25 portion of the interface is rigidly defined, the rest of the package is easily configured for specific applications.

Specifically I will use the Canadian Packet Switching Network, "DataPac"(1) as an example of an X.25 type Packet Switching Network, although any X.25 based network would provide equally as good an example. It is the interface between DataPac and DEC(2), RSTS(3) system which will be my main concern.

As the software required for X.25 protocol is not readily available from computer manufacturers at present, there may be some question of its practical value. On the other hand, with the increase in number of X.25 based Packet Switching Networks internationally, and the current and future interconnection of these networks, it seems that they are here to stay. Currently the Canadian and American Packet Switching Networks are connected to one another via a Gateway Interface (X.75). The United States Network is also interconnected to the British International Packet Switch Network. The list of X.25 based networks is growing to include Japan, Australia, France and an Inter-European network. With these factors in mind it seems appropriate to have a closer look at X.25 based networks.

The Canadian DataPac network provides prospective customers with a nation-wide network and a network interface facility for a variety of terminals. The interface between a host and the network has been left to the user. This has led to the development, by many companies, of a variety of useful network interfaces, allowing a prospective DataPac user an interface which not only connects him to the DataPac network but may also increase the efficiency of his communications hardware and software. There are trade offs, such as the introduction of another vendor between the telephone company and the computer.

Packet switching is not the answer to all communications requirements, it is just another alternative. Factors such as response, throughput, access and methods of billing must be taken into consideration, as they would for any communications application.

One could describe time division multiplexing as a number of users sharing a single communications line with each user being given a specific time slot on the line. To use a similar analogy, packet switching could be described as a number of users sharing a single communications line, where each user is required to uniquely identify his data. With time division multiplexing, if a time slot is not used it is lost, while with packet switching the network can control the number of users on any given line and optimize the use of that line.

The goal of X.25 then is to uniquely identify user data. This is accomplished through four basic levels of protocol. The first of these levels of protocol is the physical level and it is composed of a synchronous interface and modem which are connected to the network via a full duplex 4-wire line. This interface will allow communications to the network at speeds up to 9600 baud. Included in this physical level is a synchronous protocol either Binary Synchronous (BSC) or High-Level Data Link Control (HDLC).

With the HDLC protocol, which is a bit oriented protocol, data sent on the synchronous link will be delimited by a flag bit sequence, which is a string of six consecutive 1 bits (01111110). HDLC also requires that after the last data bit and preceding the terminating flag there be 16 bits of circular redundancy check, $(x^{15}+x^{12}+x^{5}+1)$. HDLC allows for the rejection of any data sequence by transmitting an abort sequence which consists of seven consecutive 1 bits (1111111).

As previously mentioned HDLC is a bit oriented protocol and thus possible random occurrences of six or seven consecutive 1 bits must be avoided. This is accomplished by a procedure appropriately called "bit stuffing". Bit stuffing entails analyzing data to be transmitted: when a sequence of five consecutive 1 bits is encountered, regardless of word or byte boundaries, a zero bit is inserted into the bit stream. The reverse is true for the receiver (bit unstuffing), which requires deletion of a bit after the occurrence of five consecutive 1 bits in a data sequence. The procedures for HDLC and BSC are standard and fortunately performed very well by a DEC DUP-11(7) synchronous line interface.

The data stream between flags in HDLC contains the next level of protocol, namely the frame level

protocol. The minimum length of a frame is two bytes or 16 bits. The frame level is responsible for establishing, maintaining and clearing the link to the network. A frame contains an 8-bit address field, an 8-bit control field and an optional data field. The address field of a frame can be either "A" or "B", and will determine which side of the full duplex link is being controlled - for example, from the host point of view, reception of a command frame with an address of "A" will require transmission of the appropriate response frame with the address of "A". Similarly the host will transmit its command frames with an address of "B" and expect to receive the appropriate response with a "B" address. The control field defines the type of frame transmitted or received.

To establish the link, the host will transmit a "set asynchronous response mode" (SARM) and wait a specified timeout period for an "unnumbered acknowledge frame" (UA) from the network. This establishes the link from host to network. The network establishes its side of the link by sending a SARM frame: on receipt of this frame the host has a timeout period in which to transmit a UA response which will completely establish the link. The link between host and network is cleared in the same manner, with the use of a disconnect frame (DISC) rather than a SARM frame.

Once the link has been established, data transmission can commence. At this point the host will set two frame-related flow-control variables to zero. We will call these two variables V(S) send variable, and V(R) receive variable. These variables are used to keep track of data transmitted or received and will cycle from 0 - 7, or modulus 8. All data is transmitted through information frames or I-frames which will be transmitted with the values of V(R) and V(S) in their control field. The variable V(S) contains the number of the next I-frame to be transmitted by the host, hence if an I-frame is being transmitted the current value of V(S) is loaded into the control field of that I-frame and V(S) is incremented to the number of the next I-frame to be transmitted. The variable V(R) contains the number of the next information frame we expect to receive from the network. If we receive a valid I-frame from the network the number of that I-frame will match our V(R) value (it was the one we expected) and V(R) will be updated to the next expected I-frame.

In response to receiving an I-frame the host or the network must acknowledge by transmitting either a "received ready frame" (RR), "received not ready" (RNR) or its own I-frame - all of which have V(R) contained in their control field. This V(R) acknowledges the V(R)-1 information frame received. The RNR frame requests the remote to suspend transmission of I-frames. I-frames, RR, RNR, or "reject frames" (REJ) are the only frames which contain this flow-control information.

The remaining frame types are used for link recovery. In the event the number of a received I-frame does not match V(R) (it is out of order) we will issue a "reject frame" (REJ) in response. This reject frame will contain the value of our V(R) in its control field and will request that the remote begin retransmission of I-frames whose V(S) value was equal to the V(R) value contained within the received REJ frame. The "command reject frame" (CMDR) acts as a catch all for most other errors. It will be transmitted with the current values of V(R) and V(S) and error cause information. Reception of a CMDR usually implies reinitialization of at least one side of the link.

To complicate things slightly, X.25 allows a number of information frames to be transmitted before any acknowledgement of receipt of these I-frames is received, so the updating of V(R) and V(S) must take this into account. The number of I-frames allowed to be outstanding is referred to as window size. It should also be mentioned that the frame level protocol is time dependent. In other words, command frames (SARM, DISC, I) must receive responses (UA, RR, RNR, REJ, CMDR) within a given timeout period.

The next level of protocol is the packet level. All of this packet level protocol is contained within information frames.

The packet level protocol is much the same in principle as the frame level, although it is more comprehensive. It is the responsibility of the packet level to connect, control and clear a number of calls across the host-network link maintained by the frame level.

A packet contains 16 bits of addressing information referred to as a logical channel number, or LCN. This LCN is only meaningful between the host and local network node. Furthermore, the packet also contains 8 bits of packet "type" information. The packet may also contain information used only to control the call or user data for transmission.

A call is established when either end receives a call request packet and responds with a call accepted packet. The call request packet will contain a unique LCN for this host-network link, the address of the calling host, the address of the called host and information about the call such as user data field size, which host is to be billed for the call, packed level window size, and transmission throughput parameters.

A call may be rejected by transmission of a "clear request" packet which will contain an 8-bit field which specifies the reason for refusing the call. Similarly an established call may be cleared by transmission of a clear request packet. On receipt of a clear request packet a "clear confirm" is transmitted.

Calls are distinguished from one another by the LCN. Once a call is accepted all packets transmitted or received for that call will contain the specific LCN given in the LCN field of the call request packet. The LCN is freed upon receipt of clear confirm or clear request packets.

Once a call has been established, data on the packet level can be transmitted across the host-network link. As with frame level, we will need to maintain variables to govern data flow. These may be called PV(S) and PV(R) - packet send variable and packet receive variable respectively - and are maintained in the same manner as V(R) and V(S), with the appropriate adjustment for window size. A unique PV(R) and PV(S) variable is maintained for each established call. Data, "received ready" and

"received not ready" are the only packets which will contain the PV(R) or PV(S) flow control information.

There are two forms of data packets - level 0 and level 1. Level 0 data packets contain data destined for the user while level 1 data packets are used for X.3, X.28, X.26 interactive terminal protocols (ITI) which will be discussed later. Different data packet types are distinguished by a qualifier bit (Q bit) in a restricted sub field of the LCN. Data packets are allowed a user data field of 128 bytes, for priority, or 256 bytes for normal traffic where the shorter packets are given transmission priority within the network. When the data field is completely filled the "more data" bit (M bit) is set in the type field indicating a logical continuation of data in the next expected data packet. The M bit is peculiar to the data packet.

On receipt of a data packet a response will be required. The response may be a data packet, RR or RNR packet. This follows the same principles outlined for the frame level.

Recovery of errors such as out-of-sequence packets is facilitated by resetting the PV(R), PV(S) variables to synchronize both ends of the link on the packet level for a specific call. This is accomplished by transmission of a reset indication packet. In response to a reset indication packet a reset confirm packet is transmitted. The call will remain connected and transmission can resume. In the event of major packet level problems, a restart indication packet is transmitted clearing the entire packet level calls as well as packet variables. The response to a restart indication is a restart confirm packet.

Control of the packet level during a call is performed by transmission of level 1 data packets or interrupt packets. The interrupt indication packet contains an 8-bit data field and is given transmission priority over data packets. It is ideally suited for control C or break indication. The response to an interrupt indication is an interrupt confirm packet. It should be noted that only one interrupt indication may be outstanding.

This very basically describes the packet level except for level 1 data packets, which are used mainly for the interactive terminal interface protocols.

The interactive terminal interface protocol allows the host and network to establish the terminal characteristics of the remote user. The actual set up of the ITI will depend on the remote user's application as well as his hardware.

The ITI protocol is established by exchange of level 1 data packets which contain a parameter list and parameter values or acknowledgements within the data field. The ITI protocol is set before data transfer and is set for the link between the remote user and the network. There is nothing to prevent the change of ITI parameters during any one call.

Until computer vendor software for interfacing to packet switch networks becomes readily available, users are left in the realm of the communications processor or black box. The basic functions of a communications processor are to provide the protocol for data destined for transmission and to strip protocol from received data. The protocol in this case is X.25 and data is in a format acceptable for RSTS input/output.

When related to X.25 these functions can easily be reduced to eight processes, namely:

1. physical level decode (DUP-11 receiver handler);
2. frame level decode;
3. packet level decode;
4. host transmitter handler;
5. host receiver handler;
6. packet level encode;
7. frame level encode;
8. physical level encode (DUP-11 transmitter handler).

Keeping in mind the previously described levels of protocol and the idea of command response, these processes can be evaluated with little problem.

The physical level receiver will synchronize on flags. Data between flags is "bit unstuffed", stored in a buffer and used in the circular redundancy check calculation. On completion, the buffer is passed to the frame level decode operation. Any errors which might occur cause this process to repeat. With a DUP-11 this complete process is easily written as a device handler.

The frame level decode process will be active only when buffered data is passed to it from the physical level decode process. When active, frame level decode will analyze the address and control field of the buffered data and arrive at one of four possible conclusions:

A. the frame was an error;
B. the frame was a command requiring only a response.

In both these cases a request is made to frame level encode for the appropriate response. The buffer is released.

C. the frame was an expected response in which case the appropriate variables are updated and the buffer is released;
D. the frame was a command requiring further processing (information frame) in which case a request is made to frame level encode for the appropriate response, all affected variables are updated and the buffer is passed to packet level decode.

The packet level decode procedure is activated only by the frame level decode and its actions are in principle much the same. Packet level decode will analyze the logical channel number and type fields and as a result perform one of five different actions:

(i) the packet was in error, in which case packet level encode is requested to issue the correct response; release buffer;
(ii) the packet was a command requiring only a response;
(iii) the packet was an expected response, in which case update applicable variables and release buffer;

(iv) the packet contained ITI pertinent data, in which case update all relevant variables, issue a request to packet level encode for response and release buffer.

(v) the packet contained data destined for user in which case update all applicable variables, issue request for packet level encode for the correct response, replace LCN and type fields with job number, pass buffered data to host-level transmit.

Host-level transmit procedure is comprised of a device handler which will transmit the buffered data and release the buffer.

Data received from the host via the host receiver handler procedure is again stored in a buffer. It must contain some identification to indicate its source. When this data string is considered complete due to some packet forwarding formula, the buffer is passed to the packet level encode procedure.

The packet level encode procedure is activated by either a request from packet level decode for a response or buffered data from the host receiver. If activated by a request for a response, packet level encode will capture a free buffer, load appropriate LCN, type and data and pass this buffer to the frame level encode procedure. If buffered data is received from the host receiver, its identification is exchanged for a valid LCN, a type field is loaded and it is passed to frame level encode.

Frame level encode may be activated by response requests from frame level decode. In this case a buffer is captured and the requested response is built in this buffer. The buffer is then passed to the physical level encode procedure. Frame level encode may also be activated by buffered data from the packet level encode procedure. Frame level encode will supply the needed frame level information and pass this buffered data to physical level encode.

Physical level encode is activated by buffered data from frame level encode. This data is preceded by a flag sequence, followed by circular redundancy check and a terminating flag. Finally bit stuffing is performed and the resultant data transmitted.

The physical level decode and host receiver handler are self-driven. Initially a read request is made to each of these processes. When a request is completed the process will reactivate itself with another request.

The four processes, physical level encode/decode and host transmit/receive, once active are driven by hardware interrupts. The remaining four processes are driven by software interrupts initiated by related processes. These four processes will execute to completion once they are entered, except for higher priority hardware interrupts. Each of these processes also contains a software status register to show the condition of that process. This register is accessible by all other processes.

A common buffer pool is declared for use in passing data between processes. Buffers are captured or released by the processes via updating a bit map of the buffer pool.

Buffer pointers and requests are passed between processes through an RT-11[4] type queue structure. Each process has a queue which is maintained as a linked list. However processing of the requests within a queue is not necessarily serial as in RT-11.

The actual implementation of the two host related processes is subject to many changes as there is a wide variety of devices suitable for these processes. The physical device could be almost anything from a DH-11[5] to another synchronous interface depending on the type of host or the desired format of data. By using different devices and handlers for these two processes the scope of the communications processor can be completely changed.

There are many other possible procedures one could follow to perform the same functions as described above. In the same light there is no specific type of hardware required to perform this function.

This type of application does not require a great amount of processor power or memory, although a lack of either will limit its capabilities. These communications processors have been implemented on machines varying from 8-bit microprocessors to PDP 11/40's[6].

For the user or manager who has a phobia of "more computing power", the manufacturers of these devices can cleverly disguise them to "set your mind at ease". Whether they are marketed as communications processors or fancy-type modems, one brand might be more suited to your application.

Packet switched communications poses some problems to a time sharing environment like RSTS. The main problem encountered is the question of forwarding the data packets to and from the remote user.

In the case of file transfer, a majority of data packets can be filled, however, in the case of an interactive remote user the efficiency of a packet switch system will suffer, as not all packets can be completely filled. Consider the Basic Plus[7] instruction "input 'password' A$;" in some arbitrary program. The string "password ?" must be output to the remote user and his response obtained before execution of the program will continue. Two relatively empty data packets must be transferred, but how are we to know if the data packet is to be sent or if more data is available to fill it?

One solution to this problem is to forward either full data packets, or data packets that have not been altered for some arbitrary time. The time of course would be dependent on whether input is expected from a user on a 300 baud terminal or input from the host machine at very high speed. In practice, this method works fairly well, although it causes an additional delay. RSTS/E however provides an even better indication of when a data packet should be forwarded by entering a 'keyboard wait' state.

Another major concern from a timesharing point of view is that of end to end delays. Depending on packet level window size, what a remote user is

receiving on his terminal at any given moment may be only a portion of the output, which is already on the network destined for him.

The undesirable effects of this delay are demonstrated through input/output control commands such as control (C, S, Q, O). A control "S" for example may be immediately forwarded to RSRS, which will cause RSTS to suspend output. Output is not necessarily suspended to the user terminal, however, as the remainder of the current data packet and any subsequent data packets already held by the network will be sent to the user's terminal. There is currently no absolute solution to this problem, although its effects can be minimized by varying packet level window size, data field size within data packets and ITI parameters.

These constraints should be considered when appraising the value of a packet switch facility or when writing software which could possibly be used on such a network. For many applications the benefits of packet switching outweigh the drawbacks.

Currently, as previously mentioned, most DataPac access is provided in the form of a communications processor which is physically located between the host and the network. In general these interfaces do not require any modification of the host, hardware or operating system, they simply interface line for line to a number of the host communications ports, hence an interface which will connect 16 EIA ports to the DataPac network should handle 16 calls over the DataPac network. The processing abilities of these communications processors may be used to better advantage for specific applications.

As a case in point, the communications interface can very easily prefix data from any given X.25 logical channel with a job number suitable for RSTS multi-terminal input/output. All data can then be passed to a RSTS multi-terminal type job through a single keyboard line.

Alternatively a RSTS controlling job can pass the data on via send/receive to individual jobs on the system. This can provide substantial advantage for remote data entry applications.

There are other methods of communication to RSTS which are even more convenient. The X.25 handling code could be included in the host and access provided to users and jobs via a run-time system. If this were done correctly the integrity of the main operating system could be maintained. The end result is specific to this operating system.

Of course the RSTS operating system could be modified to accept this type of communication interface, although the system is no longer as easily supported.

The method of implementation is not really of basic importance as long as one is aware that there are different methods and trade offs for each.

In conclusion, packet switch communication has provided us with a very attractive alternative which complements RSTS time sharing very well. We have expanded our possible customer base while reducing total communications costs. The benefits may not be the same for every application but packet switch communications is an alternative which should be evaluated.

REFERENCES

(1) Trans Canada Telephone System Trademark

DataPac Standard Network Access Protocol Specification, Trans Canada Telephone System, Computer Communications Group.

(2) Digital Equipment Corporation Trademark

(3) Digital Equipment Corporation Trademark

(4) Digital Equipment Corporation Trademark

(5) Digital Equipment Corporation Trademark

(6) Digital Equipment Corporation Trademark

(7) Digital Equipment Corporation Trademark

SCHEDULING STUDENT ASSISTANTS IN THE COMPUTING LABORATORY

J.D. Rose
California State University, Hayward
Hayward, California

ABSTRACT

Scheduling of a part-time student workforce must take into account the class schedules of the workers. For maximum stability, it should also take into account their study habits and other demands on their time. The job may be complicated by any limitations or rules imposed by Federal, State, district or campus policies. Workload patterns throughout the week and the academic term can affect the staffing level required, and hence the scheduling process. Changes in all these factors may require work schedules to be prepared frequently. Unless the workforce is quite small, this frequent scheduling can become quite tedious and time-consuming. This paper discusses the approach used to solve these problems in scheduling Student Assistants at the Computing Laboratory facilities of the California State University campus at Hayward. Special attention is given to a package of computer programs developed to automate much of the process. Written in BASIC-PLUS, they should be adaptable to other educational institutions with similar characteristics and requirements.

INTRODUCTION

California State University, Hayward, supports instructional computing in part through the operation of Computing Laboratory facilities. These contain equipment (such as terminals, card machines, graphics devices, etc.), user work areas, documentation, and personnel to provide assistance. The work force largely consists of part-time student assistants, with staff supervision. The scheduling of these student assistants must take into account their own schedules and preferences, budgetary considerations, workload variations, and the schedule horizon. To prepare manually a "good" schedule at frequent intervals is a time consuming job, and using the computing facilities themselves to automate at least partially the scheduling procedure seems an obvious approach to the problem. The philosophy used in the development of a preliminary package of programs to achieve partial automation of the scheduling process is described below. (Development is continuing, and improved versions are planned.)

SYSTEM PHILOSOPHY

Student Assistant Inputs

Recognizing that student assistants are students, we must not schedule them to work during class times, and should not schedule them during times they prefer to use for studying. This principle can be extended to a scheduling system that weights heavily the assistants' own preferences in general. The preliminary version of the scheduling package specifically allows each assistant to specify time preferences (but not job or location preferences) by designating a "1" for each "first-choice" hour, etc., down to a "6" for an hour in which work is impossible due to a class conflict. Figure 1 illustrates a sample preference matrix, prior to entry into the scheduling system.

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
0800	1	4	1	4	3	4	5
0900	1	4	1	4	3	4	5
1000	1	3	1	3	3	3	5
1100	1	3	1	3	3	3	5
1200	6	3	6	3	3	3	5
1300	6	3	6	3	3	3	5
1400	6	3	6	3	3	3	5
1500	6	3	6	3	3	3	5
1600	2	2	2	2	3	3	5
1700	2	2	2	2	4	4	5
1800	2	6	2	6	4	4	5
1900	2	6	2	6	4	4	5
2000	6	4	6	4	4	4	5
2100	6	4	6	4	4	4	5
2200	3	4	3	4	4	4	5
2300	3	4	3	4	4	4	5
2400							

Mary, Week 3

Figure 1. Assistant's Input (Preferences)

Management Parameters

There may be workweek limitations imposed or implied by Federal, State, or institutional policies. For example, a campus may limit student assistants to, say, 20 hours of work per week. Or, a student may be receiving "Work-Study" funds,

with an entitlement that averages 15 hours per week during the term. The fact that such funds may be derived in part from the Federal government rather than local sources may be an important factor. And regardless of the source of funding, there may be various rates of pay for different jobs, skills, or educational levels.

Seasonal workload patterns may exist across the year, over a single term, or even shorter periods. Thus, the scheduling process may need to take into account a pattern of staffing densities consistent with workload fluctuations.

Although some of these management parameters are collected and stored by the scheduling system, these data are not actually used by the preliminary version. Instead, they simply are displayed for the convenience of the scheduling supervisor.

The Schedule Horizon

Classes at Hayward are scheduled in weekly patterns, hence student activities tend to follow weekly patterns. The result is that our student assistants find it convenient to specify their preferences in weekly patterns, and that class-generated workload also tends to exhibit weekly patterns.

Thus, if the weekly patterns continue throughout the term, a single schedule with a one-week horizon is appropriate. However, the factors frequently change, as when student assistants add or drop classes in the early part of the term, or alter their study-time requirements in the latter part of the term as examinations approach. Here again a weekly horizon seems appropriate, but new schedules may have to be prepared with some frequency.

An example of a one-week schedule for a particular facility is given in Figure 2.

THE SCHEDULING SYSTEM

Information Flow

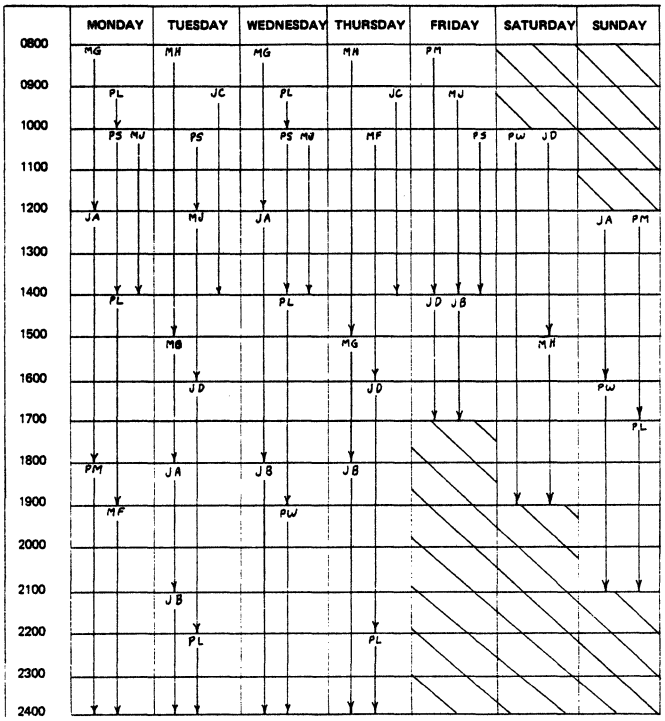
The supervisor enters various "parameters" into the system data base when initializing for a new academic term. The assistants initially prepare their "inputs" on forms (Figure 1) and then enter them into the system via interactive terminals, allowing the selective retrieval and display by the supervisor as illustrated in Figure 3.

Software

A simple data base structure provides for the hourly preference information ("inputs") for each assistant, plus some management parameters. In order to allow for the entering of schedule information which would become effective at a later date, the data base is divided into two duplicate file structures - one for the current information and one for the future.

The data collection programs select the proper (current or future) file structure according to effective-date information (entered by the assistant), and extract that person's information from the file (or request it from the user if not on file). Editing and display functions are provided, and the data base is then updated.

The supervisor's package creates, updates, displays, or erases the data base records for personnel or parameter changes. Current preference data may be



Computing Lab Assistant Schedule: Week 3

Figure 2. Supervisor's Output (Schedule)

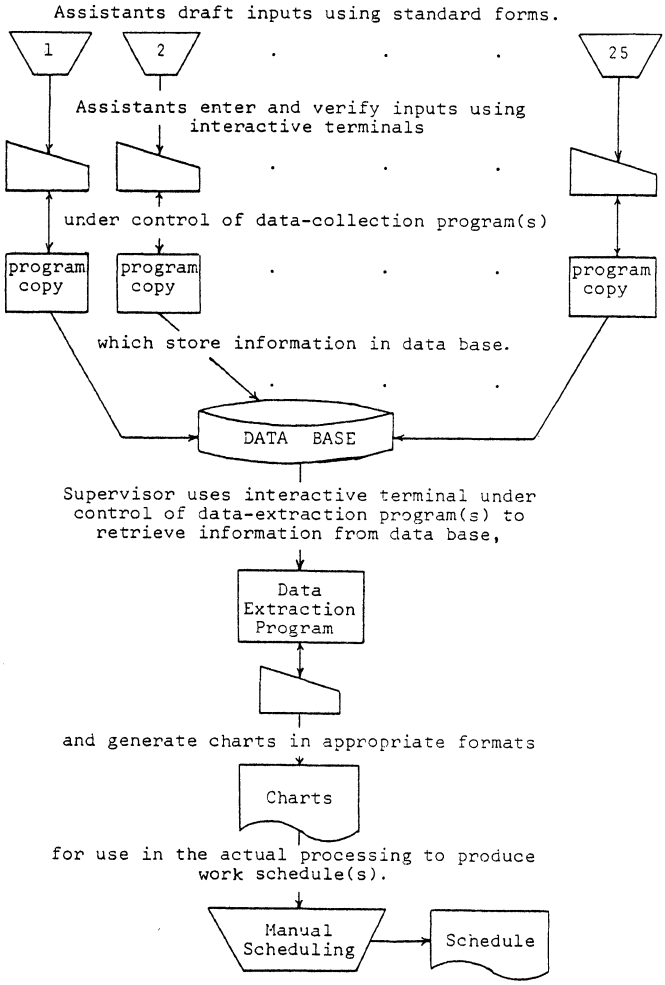


Figure 3. Scheduling Process

updated from the "future" file if requested and the dates are appropriate, and displayed in formats found to be most helpful in constructing a schedule. The displays can be for specified subsets of the workforce for additional versatility.

The various programs which implement the system are written in BASIC-PLUS, and are running under RSTS/E on an 11/45. Although coded in an "ad-hoc" manner, they should be adaptable without difficulty to another campus facility having similar characteristics, requirements, and computing system. With additional modification, they should also be usable with other hardware or software systems.

The scheduling problem investigation is continuing, and (both structurally and functionally) improved software will be available at a later date. A mathematical model of the actual scheduling process has been derived (1), but additional research will be needed in order to incorporate it into the system.

REFERENCE

1. Economides, S., and Rose, J.D., "Workforce Scheduling in a State University Computing Laboratory Facility," paper presented at the American Institute for Decision Sciences 10th Annual Convention, St. Louis, November 1, 1978

ACKNOWLEDGEMENTS

The author gratefully acknowledges the contribution of Spyros Economides in directing the research effort into the formal modeling aspects of this project, and the assistance of Bonnie Benzinger in implementing and testing the current scheduling system.

CURRICULUM INTEGRATION AND USER SUPPORT OF RSTS IN A SMALL BUSINESS COLLEGE
"What do we do now that it works?"

Arthur K. Lash
Nichols College
Dudley, Massachusetts

ABSTRACT

Installing an academic timesharing system requires more than having an operational system. The users and manager of a new timesharing system (specifically a RSTS system) are quickly confronted with a variety of support and curriculum decisions. The experiences at Nichols College (a RSTS user since September 1977) can serve as a guide for the novice, and more mature, RSTS installation. Nichols College is a 700+ student four-year college with an evening graduate school whose primary emphasis is on business and public administration.

This paper details the problems and their solutions, the approaches and techniques Nichols has used since becoming a RSTS site.

BACKGROUND

Nichols College is a four-year institution whose primary emphasis is on business and public administration. In addition to the 700 undergraduate students, a part-time MBA program, with 100 students is offered. The College acquired its first computer (an IBM 1130) in 1968. In 1977 a replacement system was purchased to bring the College's administrative and academic capabilities into an interactive mode.

System Resources

A PDP 11/34 (purchased through EDUCOMP Corp.) with 96K and running RSTS/E was selected to replace the batch-oriented 1130. One administrative and four academic terminals, in addition to a LP05 line printer, a TS03 tape drive, and dual RP02 disks comprise Nichols' present configuration. While most academic work is currently performed on the 11/34, the 1130 continues to handle administrative needs until a phased-in conversion to the RSTS/E system can be completed.

Curriculum

Since 97%of Nichols students are in the business administration curriculum, a fairly structured set of courses is required of most freshmen and sophomores. Liberal arts requirements (English, history, social sciences, math, statistics, science) comprise the bulk of the first two years. Introductory managerial and financial accounting as well as the beginning of the business "core", round out the requirements. Junior and senior courses primarily follow major concentration. The introductory computer course (which uses BASIC) is not normally taken until the third year.

A systems emphasis is offered within the management concentration. Courses in systems analysis, computers in modern organizations, applications programming (COBOL), quantitative methods, and a capstone management information systems offering, constitute the systems emphasis.

Support Problems

Limited resources of all types are a problem which plague many organizations, and Nichols is no exception. Significant energies had to be directed toward system selection, acquisition approval, and the site preparation completion. Since no individual was assigned these tasks as his/her sole responsibility, long range preparation for the actual running and support of the system was limited. Once the joy and frustration of installing the system was over, the stark realization of providing services to users totally unfamiliar with any interactive system, struck. Compounding the situation was the fact that the start of the 1977 Fall semester coincided with the installation of the system.

It was in this context that NIC (Nichols Interactive Computer) came into existence. Since the author was primarily responsible for system operation, support, education, as well as teaching full-time, the constraint of time became a major one. However, the solutions to the problems of supporting all types of users, and curriculum integration under this constraint, can serve as a model to other sites. In addition, these solutions serve as testimony for inter-installation cooperation and DECUS.

CURRICULUM INTEGRATION

A primary task of NIC is to support and enhance the offerings at Nichols. Two distinct areas exist. The systems courses described are those where original program creation takes place. Support facilities for these types of courses differ from non-

system courses where the use of canned library programs is necessary.

Systems Courses

RSTS/E lends itself well to introductory students. The introductory computer course is split between dealing with computers in modern organizations and programming in BASIC. An account for each introductory student (approximately 80 per semester) is created within the same project number. The project manager's (XXX,0) account was used extensively. Illustration programs, assignments, data, and limited operating instructions are made available from this account. While a programming text for strictly BASIC-PLUS is not used, the differences between BASIC-PLUS and other BASICs are covered in class lectures and in the Nichols Interactive Computer User's Guide (see User Support for a description of this guide). Introduction to Computer Programming with the BASIC language by Harvey Deitel (Prentice-Hall) is the language text. The computer applications text is Business Data Processing by Mike Murach (SRA).

Oriented toward business data processing-type applications, the introductory students' programs increase in complexity from simple report production using FOR/NEXT and READ/DATA statements to simulated file maintenance programs using arrays. Simulation, two-dimensional arrays and string manipulation are also covered, but actual BASIC-PLUS files (ASCII, virtual arrays or Record I/O) are not part of the introductory course. Thus, the material needed to produce these types of programs can be adequately found in a BASIC text supplemented by in-class lectures (the DEC BASIC-PLUS Language Manual is a must for the instructor, however).

The applications programming course poses a different problem. Because of the wide spread use of COBOL by businesses, it appears appropriate to offer students a course in the language. Nichols decided against using DEC COBOL for one major reason: cost. COBOL would place a considerable operating load on the system in a student environment because work would be primarily in a program development (compile) mode. WATBOL from the University of Waterloo (Canada) was selected to provide COBOL experience for students at a considerably lower cost and strain on system operations. Although providing a less extensive version of the language than the DEC compiler, WATBOL serves its purpose for student use. The speed and ease of compilation, combined with extensive error diagnostics made WATBOL an excellent compromise for Nichols' purposes. The text used for the course, An Introduction to COBOL with WATBOL: A Structured Approach (WATFAC) by Cowan, Dirksen, and Graham, together with a WATBOL-11 Users Guide serve as the primary documentation for WATBOL under RSTS/E.

The final systems course which utilizes NIC is the capstone offering titled "Management Information Systems." File structures are discussed and actual programming of file manipulation operations are required. Unfortunately, this area is not supported as adequately as other areas of BASIC under RSTS/E. The Users Guide contains detailed information on the construction and programming of sequential ASCII files. Virtual array and record I/O files are not in the current version of the User's Guide primarily due to its perceived inappropriateness in a general document. A Guide to Programming in BASIC-PLUS by Bruce Presley, et. al., of the Lawrenceville School contains a chapter on Virtual arrays. This paper's author is developing a guide for the more advanced file handling facilities (virtual arrays and record I/O, available in the Spring 1979). While these topics receive adequate coverage in the BASIC-PLUS Programming Manual, this publication (and other system reference manuals) are not appropriate teaching/learning aids. Therefore, supporting these types of features for student users requires more material than DEC supplies.

Non-systems courses

Typically, a student does not encounter his/her first systems course (the introductory computer course) until the junior year. A great deal of computer integration is possible and necessary prior to junior year. However, this area offers the widest range of possibilities for system usage and concurrently an area for potential problems. Adequate documentation and user support is essential in this area since these users are unintiated to the workings of NIC and the BASIC language.

Two major facilities have been established to service the non-system courses:

A. Public library - a library of over 400 canned programs resides on the system (in a system library account). These programs have been acquired in three ways:
   1. EDUCOMP starter kit - as a system supplier, EDUCOMP has compiled most of the DECUS library educational programs. These cover accounting, business, economics, statistics, math, social and natural sciences, BASIC tutorials and games. Documentation varies from none to adequate. Many programs have originated from the Huntington Project or Project SOLO. Documentation through DEC is available for some of these programs.
   2. Other sites - through the DECUS library and contacts with other installations, many additional educational programs have been placed in the program library.
   3. Original programming - students have been encouraged to submit programs for inclusion in the library. While most programs seem to be categorized as game or novelty, a trend toward more useful applications appears underway.

   Communicating the contents of the library to the users is a substantial problem. While other installations have developed category search programs, Nichols has found written documentation to be sufficient. A text file of each library category was established. The file contains the name of the program and a brief explanation of its purpose. These files are listed and placed on the walls of the terminal room for reference. Any changes, additions or deletions made to the library mandate modification of these text documentation files. Faculty demand more detailed instructions and sample runs of programs before assigning these to classes. A current project underway is to extablish such documentation.

The public library has seen limited actual use in courses to date. No pressure has been placed on faculty to integrate these programs into their offerings. Only those faculty who have a personal desire to integrate system facilities into their courses have actually done so. This has actually been a blessing in disguise. Due to limited personnel resources and a limited number of terminals, a slow increase in system usage beyond system courses has allowed for a controlled, manageable growth in demand.

B. Course library - in order to accomodate those courses using the system, another library has been established to hold those programs. Decreased access time and reduced confusion was the primary motivation for establishing a separate library. Most of these course-related programs have come from the public library. Each course is given a separate project number with sufficient programmer numbers to satisfy the particular application. A simulated payroll program producing checks, stubs, register and general journal entries based on student inputs is required of all freshmen accounting students. This type of program characterizes current usage. A/statistical package was of primary concern. The 1130 had such a package and one on the new system was essential. STAT11 from the DECUS library (DECUS #RSTS11-110) was selected due to its cost, ease of use and demand on disk space. The STAT11 package contains a RUNOFF version user's guide which was modified to meet NIC characteristics and made available to instructors and students who needed details beyond those provided in class.

A slow and deliberate approach has characterized Nichols' approach to integrating system usage in the curriculum. While the system has only been operationas since the Fall of 1977, the system-related courses are well integrated with computer usage. Non-systems courses are beginning to incorporate system usage. This phased-in approach (most coincidental) has been helpful in allowing system shakedown and development of adequate support facilities for the truly novice user.

USER SUPPORT

While most of the users on the system were enrolled in systems-related courses, it was easy to explain system operations. Although it placed a large burden on the instructor, it was the only recourse without adequate documentation for system operations and usage. To a user at any level, support must be provided. Ideally, the goal for user support should be personal assistance when required, combined with written tutorials and reference material. Nichols has taken significant steps toward reaching this goal.

User's Guide

A problem which plagued any efforst to provide user support was the lack of any adequate written materials for novice users (and in many cases the system manager). Programming manuals are not appropriate learning texts. They are not written as such and can not be expected to serve the purpose. A guide for novice users which would serve as a combined statement of RSTS/E as implemented on NIC and BASIC-PLUS was essential.

Attempts at writing a user's guide were inconclusive. The time required for such a project is formidable. Such time was not available to a system manager trying to learn system operation and teach on a full-time basis. The author became aware of a user's guide which had been written by James Condict and James Krupp at Middlebury College. They made available a copy of their 11 chapter guide in RNO.TSK format (for a nominal fee). Each chapter was edited to conform to the Nichols environment. A chapter on FORTRAN, in the final version, was omitted since the language is not taught in any course. The final copy of the manual was produced on a daisy-wheel printer to achieve a document which is easy to read and sells for $5.00.

The guide focuses on three main areas:

1. User orientation - Directed toward a novice user, the guide leads a person through what a computer is, how to sign on to RSTS, executing canned programs and various ways of inputting data to a program.
2. BASIC programming - a concise presentation of BASIC-PLUS through matrix commands and sequential files.
3. System utilities - the commands necessary for creating and maintaining BASIC programs as well as various system utilities (e.g. PIP) are explained.

The guide is useful to the novice BASIC user, a user of canned library programs, experienced users who need to become familiar with RSTS terminology and function and for programmers who will be writing in FORTRAN and COBOL. This one document alone has provided significant support for all system users.

Hands-on assistance

Regardless of the breadth of written documentation provided for system users, there comes a point where personal assistance is required. The four academic terminals at Nichols are located next to the computer room. In order to provide assistance, student consultants are employed to operate the terminal room. These individuals are usually work-study students who have an interest and above-average knowledge of the system.

The consultants serve three main functions. Primarily, they assist users in distress (e.g. recovering a program which had not been SAVEd). They ensure both the quality of the physical treatment of the terminals and the enforcement of a few essential rules for terminal use (e.g. Running games is prohibited when there is a waiting line, monitoring use when a waiting condition exists to keep aimless work to a minimum and the flow of users constant). Lastly, the consultants provide a ready force of interested programmers for simple programming projects. The Nichols' experience indicates that the consultants should staff the room continuously during operating hours (8 AM to 10 PM) until the introductory programming students have completed their first round of program creation, saving, modification, and running (usually 2 weeks at the start of a semester). Following that time, the computer center staff and other users can handle most problems during the day. Consultants, however, are used for approximately 4 PM to 10 PM daily, except

Friday and Saturday when the terminal room is closed. Without the use of student terminal room consultants a tremendous burden would be placed on the computer center staff, a high level of student frustration would exist, and the operating hours would be sharply curtailed since the security of computer and terminal rooms could not be insured.

EXTENDED SYSTEM RESOURCES

The need for a system to support a curriculum and all of its users exists. With limited financial and personnel resources, a burden is placed on those responsible for insuring that these needs are met. Given these time, financial and personnel constraints, Nichols has taken the approach of utilizing the tremendously cooperative nature of most RSTS/E sites in acquiring resources which extend system capabilities and ease of operation whenever possible.

Account Creation and Accounting

Since account creation is one of the first tasks with which a new system manager is confronted, it can be a time consuming operation. After several different approaches, Nichols now uses a modified REACT program written at Middlebury College (DECUS #RSTS11-109). Creation (and deletion) of groups of accounts is easily accomplished. Most courses have individual student accounts assigned. The creation of any number of individual project, programmer numbers (PPN) is easily accomplished with this program. In addition, accounts are assigned random passwords and a listing file is created for the instructor's use in actually assigning the numbers in class.

An added benefit of this modified REACT program is the creation of a virtual array file of all PPNs which is used in modified MONEY utility. The drawbacks of the standard MONEY program become glaringly obvious the first time the listing is viewed - no tables, by project or overall! The modified MONEY program allows for a more usable system accounting program. For non-privileged users, MONEY allows project managers to see their entire project and programmers their own utilization. The Middlebury MONEY, and others existing accounting programs achieve the same results, are a necessary modification to the standard RSTS/E in order for system management to monitor usage.

Language Processors

The curriculum orientation of Nichols dictated the acquisition of two additional languages; COBOL and FORTRAN. FORTRAN was necessary for compatibility with packaged academic software which might be acquired and written in the language. The justification of a COBOL processor and the WATBOL compiler in particular, has been detailed above. An added feature of the WATMON (the monitor under which WATBOL executes) is the existence of a SORT and PRINT utility which is very helpful for student programmers.

While these added languages are needed to support the curriculum needs of the College, they must be supported by adequate documentation. Again, the DEC supplied FORTRAN manuals can not be viewed as tutorial or even reference for most student users. Fortunately, the Middlebury user's guide contains a chapter on the FORTRAN implementation of RSTS/E. When the Nichols user's guide was constructed, this chapter was excluded. It was, however, utilized as a stand-alone FORTRAN user's guide. The same approach can be taken for any chapter or a related set of chapters in producing specifically oriented user documentation.

WATBOL documentation was not as concisely available. As mentioned earlier, a text exists which is WATBOL oriented (An Introduction to COBOL with WATBOL (WATFAC) by Cowan et al.). However, there are slight implementation discrepancies with RSTS/E. The University of Waterloo provides a WATBOL User's Guide and File Utility Guide which can be reproduced to create a fairly concise set of documentation. Minor modifications are needed to orient the support materials from a batch to an on-line orientation.

Text Editor

One of the major facilities of RSTS/E with which Nichols found faults was an easy to learn and use text editor. Both FORTRAN and WATBOL acquire the use of an editor to create source programs. Creating and modifying documentation (including modifications to the user's guide) require an easy to use editor. A line-oriented editor was obtained from Babson College. Written by Doug Platte, this editor has proven to be easily learned. Besides line-oriented commands (add, delete, list, replace), it has string-oriented operations (insert, delete, find) and possesses file merge capabilities. A concise, well-written, manual is also available.

This editor is fairly small(27 blocks when compiled) and has become the editor used at Nichols for all editing operations.

Other facilities

A number of other facilities have been added to the system to provide easier operation of the system for the typical system user.

Modified LOGOUT - These modifications come from the Middlebury package of utilities. It allows for a user to respond with a "P" for the "Confirm" prompt. "P" means, "Please log me out even though I have exceeded my quota. All such requests are logged into the GRIPE file where they can be reviewed by the system manager. This facility is helpful since a novice user may be confused as to why he is above quota, advanced users may in fact create programs beyond quota, or a condition might arise where some backup files exist and a user is not sure whether they could be deleted. The philisophy here is "better safe than sorry." When the computer center staff or instructor is available, these conditions can be resolved.

Detached Batch - This is another Middlebury utility which operates as a detached job and does not need OPSER, QUEMAN or the SPOOLER to be operational. When batch processing is desired, the program detaches until the job has been completed. Functionally, this batch processor is equivalent to the standard DEC program but operates with considerably less overhead.

Password Changing - Password changing programs lift a considerable burden from the shoulders of the computer center staff and/or instructor. Once the randomly generated password has been assigned (through the modified REACT program) each student is responsible for remembering and maintaining the password. The system manager, however, has the ability to create a file of project numbers where this capability is not desired. The public access number and a course's number (where many students use the same PPN) fall into the category of PPNs where this change capability is not allowed. The password maintenance facility and a fixed disk quota instill a sense of control for each user over his/her account. In addition, the responsibility for several aspects of account upkeep is clearly placed in the user's hand.

SUMMARY AND CONCLUSIONS

Faced with the related problems of supporting novice users and curriculum integration of a RSTS system, the Nichols approach has been an evolutionary one. Initially, only those courses which were directly related to computer usage (the systems courses), utilized the system. Novice users in these courses were supported by instructor-provided aids and guidance. Support schemes (terminal room assistants, operational guidelines, manuals, etc.) were developed using the students who were most directly involved with computer usage. In order to extend the integration of the computer into non-system courses, these support facilities should be in place. A gradual integration of these non-system courses has been occuring. In this way, each new application can be observed and stabilized prior to a new undertaking. This approach seems mandatory with limited time, personnel, and monetary resources.

Resources beyond the "standard" RSTS system were also incorporated. Again, the limiting resources in a small college dictated extensive use of materials produced at other colleges. User application programs and RSTS utility programs were acquired at nominal cost from a variety of sources to complement system operation. A comprehensive user's guide for BASIC-PLUS and RSTS relieved the burden of having individuals serve as the primary resource for instruction and assistance. Once completed, efforts at supporting other system features could then be undertaken. System accounting, an easy-to-use text editor, WATBOL, and various utilities related to the handling of accounts, were added to enhance the system resources.

Nichols has been able to provide user support and curriculum integration through combined use of in-house efforts and shared development from other sites.

ON-TARGET,
An Effective Bottom-Up Approach to Job Shop Control

Norman J. Viehmann, President
Viehmann Corporation
Woburn, Massachusetts

ABSTRACT

Job Shop Managers are measured by the results of delivery performance, cost performance, and the value of work-in-process inventory. These objectives conflict with one another. Traditionally, managers of job shops have had to base their decisions on summaries of incomplete data. To obtain useful information at an appropriate level of detail it is necessary to process very large volumes of data. The computer can provide such detail. It also makes possible measurement of a shop manager's results separately from those of the materials manager, the production control manager and engineering manager. The great change in level of management sophistication made possible by the computer is not always practical to implement in one giant step. This paper describes a bottom-up approach for providing job shop managers timely, detailed information that has successfully used the computer to help improve results.

INTRODUCTION

At all levels, managers of job shops are faced with complex decisions. Typically the reports used by job shop managers lack the detail necessary to support good decisions. As a result, work-in-process becomes excessive, serious bottlenecks develop, and overall delivery performance remains beyond management control in the attempt to balance overall production with order input levels.

Traditional reporting includes Total Shop Load by Week; Number of Jobs In-house/Released; Current Backschedule (Load and Number of Jobs); Hot List of Top Priority Jobs; and Open Master List. Planned capacity is reviewed weekly. Gut feeling of workload contributes more to decisions on overtime, hiring and layoffs than data in these reports.

Policy for accepting incoming orders is oversimplified. Orders are accepted for ship dates requested by customers unless the shop is shipping significantly further behind schedule than normal. In that event, schedule dates are negotiated. The effect on load is rarely checked-out during the order acceptance process.

What is needed is a system of reporting that will do for job shop managers what time-phased requirements planning did for material managers. However, an even more complex reporting scheme is required. To control deliveries of the hundreds of orders with tens of thousands of operations in a job shop, it is necessary to control 1) the size of the queues of jobs waiting at each work center, 2) the load by week for each work center, and 3) the corresponding effective work center capacities.

Sophisticated computer-generated reports can pinpoint needs for management action. In practice, these reports cannot be introduced directly to replace the old manual system. Intermediate steps are required to allow management to get the feel for the new reporting tools.

This paper describes an "entry level" Shop Floor Control System that has been introduced in two job shops. The first system was installed in batch mode and has operated successfully without change for ten years. The second system, described here, features on-line entry and update. It is operating under DEC's RSTS time-sharing system on a PDP-11/70 and is being installed in a manufacturing job shop on a PDP-11/34 under CTS-500. The programming language is BASIC PLUS 2.

We have called the system the "ON-TARGET Job Shop Control System". It provides the means for accumulating and storing all the data that is needed for basic detailed decision-making on the shop floor and successive management levels. Before looking at the system design let's take a closer look at the problems and decisions that occur daily in every job shop.

THE JOB SHOP ENVIRONMENT

A typical medium-sized job shop has 50 to 100 work centers for performing machining, assembly and/or hand operations on hundreds of different product items. Each product item has a unique engineering drawing or sketch and a unique routing. The routing defines the sequence of operations and it specifies the work center, the setup time, and the production rate for each operation.

Shop orders are created to satisfy either inventory or direct customer orders. Upon acceptance, a shop order is assigned a scheduled ship date. It may also be assigned a priority code that defines

its importance relative to other orders. Many job shops accept orders for products that require a new routing to be prepared by methods engineering. It may be necessary to procure raw material for the specific order. An order can logically be released for production only after methods engineering is complete and the material is available.

Machine feeds and speeds used in production often differ considerably from what is specified on the routing. Sometimes the tooling and occasionally the work center actually used are different from the routing. Once a computer system is installed it is important that routings be updated to match actual practice.

MANAGING THE JOB SHOP

Purchasing has the responsibility to furnish material. Methods Engineering is responsible for specifying how each product will be manufactured. Manufacturing Engineering is responsible for specifying and maintaining the jigs and fixtures and machines. Manufacturing must see that it completes shop orders within cost and delivery objectives that it has accepted.

The effects on delivery of late material, incorrect methods specifications and machine failures can be quantified and reported. They seldom are. Most often, manufacturing bears full responsibility for deliveries and is expected to make up time for late material, experimentation with methods and facility downtime.

It is not uncommon for Manufacturing to find itself faced with a surge of new load when its backschedule load has already gone out-of-control and it has insufficient lead time to hire and train new employees. Delivery performance decreases rapidly. Overhead costs rise. Work piles up on the shop floor. Lots are split to make partial deliveries. Jobs are lost in process and expediting effort and overtime soar.

The most direct solution to this very common situation is to provide a set of reports that can promote the liklihood of making the best decision at the right time.

The shop management decisions that directly effect delivery performance are, from the bottom up:

Shop Floor Scheduling:

What job should be run next on this work center?

Loading/Overtime:

How many man hours should be assigned to each work center today; this week?

Capacity Planning/Hiring-Layoff:

How many man hours should be scheduled for the entire shop next month?

Master Scheduling:

How much load shall we plan (or accept) by period?

In order to make these decisions the manager needs to have the following information:

For Shop Scheduling:

1. What jobs are available to setup on this work center?
2. What jobs are about to become available for setup?
3. What is the Schedule Date; the Priority; the Hours required on this work center and number of operations remaining for each job waiting at this work center?

For Loading:

1. How many standard hours are in queue at this work center?
2. How many actual hours are required from this work center to complete back schedule and current weekly loads?
3. What is the average production output of this work center?
4. What is the current planned capacity of this work center?

For Capacity Planning:

1. Has the queue been increasing, remaining stable, or decreasing for this work center, this department, the shop?
2. Is the anticipated load increasing? Is there a load swell or depression moving through the profile of loading weeks? Is load decreasing for this work center, this department, the shop?

For Master Scheduling:

1. If I accept this job will it create or encounter an overload on any facility?
2. What ship Schedule Date can be assigned to this job within the limits of current shop load?

A system that provides this information requires data on every operation performed on every job. This same information can be used to record job costs.

HOW THE ON-TARGET SYSTEM CAME INTO BEING

Design of the system was initiated in response to what were viewed as serious problems in a job shop that manufactures heavy equipment. Jobs are both make-to-order and for inventory.

1. Approximately 75 jobs were being rescheduled each week. This represented about ten percent of the released jobs.
2. The critical ratio values shown on the daily shop schedule presented priorities that did not coincide with the knowledge of shop managers. The report was not used in the setup decision-making process.

It was decided to involve key production people: managers, supervisors, methods engineers, lead men, expediters in an intensive study to reveal problems and suggestions. Within one week a workable priority policy was developed and a system design concept completed.

The strategy of the system design was to provide only the essential facts needed, where and when needed. It was determined that there was a real need for a reliable shop floor scheduling report.

This report would be used if priority information was credible. The priority system that was agreed on reflected solid facts that seldom changed as a job progressed through the shop.

DESIGN CONCEPTS

General

The Job Shop Control System is part of a total business system that includes inventory control, order processing, general accounting, payroll and sales applications. This business system is operating on both in-house systems and on time-sharing services, using from small PDP-11/34 configurations to large 11/70 systems.

The software design allows maximum flexibility to accomodate the needs of individual organizations, A single company can tailor separate Job Shop Control Systems to the specific needs of different shops or divisions. For instance, report files and transaction files are maintained independently of time periods. Each user can select when to run his reports and when to clear his files.

Each user has his own unique access code. This limits him to affecting his own files and protects his data base from access by other divisions or departments.

The on-line editing feature facilitates assignment of the data entry responsibility to the department that is the source of the data. Experience has shown that this improves the accuracy of the data. It reduces the cost of data entry by greatly reducing editing and review effort. On-line updating of multiple files provides information that is more current by one to six days.

Every application function is accessable by a direct command that is contained in the menu. The application functions fall into the following categories:

Master File Maintenance and Display
Shop Floor Transactions
Add/Modify/Display Jobs
Reports

Data Entry

Data entry is in conversational mode. Key information such as job number, clock number, work center number and part number are edited on-line. When a value is entered that is not recognized by the system an error message is displayed. For certain fields the user may enter a "HELP" command to cause a list of valid values to be displayed. The data entry approach fulfills the self-teaching design strategy and allows an individual to use simplified procedure as he becomes familiar with the system.

File Structure

The master files and the Open Job file are keyed index files. A single record is retrieved with two accesses. Records in the Shop Transaction file are chained internally by Job and externally to the corresponding operation detail record in the Open Job file.

Reports

Reports are requested from terminals via the Report Menu. There are four output options: 1) Display at a CRT terminal, 2) Print at a remote printer terminal, 3) Print at the central system printer, and 4) Create a "spool" file and hold for printing later.

The print programs are designed to be independent of periodic cycles, so each user can specify his own reporting cycles. The result is that one user can vary his reporting cycles as special needs arise.

The Job Shop Control System reports are designed to provide the facts necessary for making well-timed decisions that will improve in-process inventory, shop efficiency, costs and deliveries. For instance one user established the following initial reports:

Daily: Shop Floor Schedule
Transaction Register
Performance Exceptions

Weekly: Work Center Utilization
Direct Labor Analysis
Scrap/Reject Analysis

On Request: Job Status - Selected Categories
Job Cost
Shop Load

DESIGN FEATURES

Direct Inquiry

The system provides direct inquiry to Customer, Employee, Work Center and Routing master records; to the Standard Rate file, to Open Job records and to Open Job Routing detail records. Inquiries can be displayed or printed at remote CRTs or printer terminals.

The master files and the Job file are up-to-date as of the last transaction entered to the system. Inquiry always provides the latest information.

Job Processing

New jobs are added to the system via the Job Menu. The data entry clerk need only key in the product code, ship schedule, order type, customer number, customer P. O. number, lot quantity and priority. The system assigns a sequential job number and an initial status code. This code can be overridden. For instance it may be known that material is already available when the order is entered.

Besides entering new jobs there are many other important day-to-day informational functions that need to be transmitted to the Job Shop Control System. These include:

Change ordering information,
Change the job routing,
Split the job,
Release jobs, and
Cancel jobs.

All of these functions plus "Display Job" can be accessed through the Job Menu.

The CHANGE JOB command is used to change priority, ship schedule, or customer purchase order number. The lot quantity, status and estimated material cost can also be changed provided the job has not been released to production.

It is not uncommon for the methods department to change the routing after an order has been released to production. ON-TAGET recognizes this reality of maintaining accurate shop floor schedules and load. The system makes it possible to change routings and have the change take effect while a job is actually in process. It is possible to change operation time, to insert or remove or change the sequence of operations which follow the last operation for which labor has been reported.

The SPLIT command recognizes another reality of job shops, the split lot. The new quantities, priorities and ship schedules are assigned to each split lot as specified. Actual costs are split proportionately between the jobs to the operation where split.

The RELEASE and CANCEL functions provide a simple means for changing the status of many jobs with a single transaction.

Transaction Processing

Shop transactions can be classified in the following categories:

Production Labor

- On-standard
- Substitute or Added Operations
- Setup
- Rework

Non-Production Labor or Daywork
Material
Merge Lots

Transactions can be entered either from a shop floor terminal or from a display terminal located in the office. The shop floor terminal can capture data directly from the machine operators. Prepunched cards save additional effort and can be used to enter employee clock number and job number. The shop floor terminal captures and enters actual hours automatically.

Only six or seven items are entered for standard labor transactions:

Clock number
Job number
Operation number
Actual hours (via office CRT only)
Quantity good
Quantity rejected
Operation complete, yes or no.

If off-standard, the work center number needs to be entered. One manufacturer experiences an average of only 21 keystrokes per entry.

The MERGE transaction brings the actual cost of one lot that is combined with another lot into the cost accounting trail of the second lot. The final job cost report then accurately reflects the cost of the combined lots.

Non-production transactions capture actual hours in up to ten daywork categories. These are reported daily by employee and weekly by category within department.

Scrap quantities are entered on rework transactions. The direct labor cost per unit that has been accumulated through the operation where scrapped is used to compute the scrap cost.

The Transaction Register can be printed as needed. The user simply specifies the period for which it is desired. Transactions are removed from the file by using a copy function and specifying a "reorganization" date. Transactions for all jobs that have been completed on or before that date will be removed from the file.

Job Costing and Status

With the direct inquiry capability a user simply enters the job number to obtain the latest status. Immediately the current shop floor location and the accumulated cost will be displayed. It can also be printed. This data is current through the last operation complete.

Current information about an operation in process can also be displayed and printed. The detailed information about an operation is updated as each shop transaction is accepted.

The Job Cost Analysis lists the detail of each type of transaction at each operation. Labor variances for standard production and for non-standard production are clearly shown. If desired, setup can be separated from "run" performance by using setup transactions. Rework labor and scrap costs can be itemized. The Job Cost Analysis clearly shows the sources of individually controllable costs.

Master File Maintenance

All master file maintenance functions are performed on-line. Tutorial messages are used extensively. A full list of employees, customers, work centers, standard labor rates or standard routings can be displayed with the HELP command. If an attempt is made to add a record with a duplicate key, an informative message will be displayed. For some fields only values that correspond to pre-specified ranges will be accepted. Where rigid specifications are not practical, warning messages are provided.

Master file maintenance routines are accessed from the Main Job Shop System Menu. Master file data that is not affected by shop transactions can be modified. New operations can be inserted and operations can be removed from the Routing Master.

Reports

The Job Shop Control System reports are designed to support the five primary functions that determine shop performance:

Shop Floor Scheduling
Resource Monitoring
Loading

Capacity Planning
Master Scheduling

In addition to the reports described here, the Job Shop Control System can provide the data for the almost infinite variety of reports that users will request to support individual management styles.

Shop Floor Scheduling

Released each morning and current as of the end of the second shift the night before, the Daily Schedule lists by work center all the jobs that are at the work center and all jobs that are just one operation before that work center. The jobs are listed in sequence by scheduled ship date within priority number. The standard load hours are given for the job at the work center; also the number of operations remaining; and for "prior operations" jobs, the work center that they are now at.

Resource Monitoring

These reports are designed to provide feedback on labor performance and machine utilization in time for effective investigation and corrective action.

Daily direct labor performance exceptions are reported on the Transaction Register. Foremen can check on causes for large variations in labor efficiency in time to correct such problems as poor setups, insufficient training, etc.

The Weekly Direct Labor Performance report gives a clear picture of relative performance and allows a foreman to spot significant changes in an individual's performance.

The Weekly Machine Utilization report presents clear facts about each work center that quantify effective capacity, underutilization and downtime.

The Weekly Scrap/Reject report identifies sources of unusual scrap and reject activity and quantifies the costs relative to overall shop activity.

The three "Weekly" reports can be created for any period desired.

Loading

The Job Shop Control System allows a manager to observe the effect of alternate plans for releasing and scheduling orders. Load is expressed in standard labor hours. The Weekly Labor report lists by work center by week, the total in-house load, and the released in-shop load. It also shows the change in load from the prior week.

The load is determined for each operation on each job when it is entered to the system. The load-week is determined by backing off from the scheduled ship date. An estimate of the waiting time the job will experience at each operation is determined from a Queue Table that is accessed via the work center file. Each center is assigned one of several Queue Tables. The Queue Table expresses estimates of waiting times as a function of a Job's priority and operation load hours. If backward scheduling results in a starting load week that is earlier than the current week, the job is loaded "forward".

Capacity Planning

Capacity Planning is the process of comparing planned load to existing capacities and making decisions to change capacities.

The effective capacity of a work center reflects the cummulative results of operator efficiency, machine and tool downtime, and manning.

The Job Shop Control System compares effective capacity, planned capacity and load. The Weekly Work Center Utilization report develops the effective capacity and compares it to the planned capacity. The latter is maintained on the Work Center file. The Weekly Load report compares load to planned capacity.

Master Scheduling

The term Master Scheduling is used un at least two contexts. When a Materials Requirements Planning system is in use, the Master Schedule refers to the planned orders for the end products that are entered to the MRP system for the purpose of generating orders for inventoried items and developing capacity requirements.

When MRP is not used, the term Master Scheduling has many different meanings. It usually refers to a planning process that is not short term.

The Job Shop Control System facilitates Master Scheduling. Jobs classified as "Planned Orders" can be entered to the system, the Load report run, and then these jobs removed or replaced with an alternate plan. The resulting Load reports will give the required capacity by period for 13 periods into the future, and will include the existing backschedule load.

ON-TARGET IS PART OF A TOTAL BUSINESS SYSTEM

The Job Shop Control System can be implemented independently from its sister applications:

Inventory Control
Order Processing/Billing
Sales Analysis
Payroll, Purchasing, and
General Accounting

If desired, the orders generated by the Inventory Control and/or Order Processing systems can be processed directly to the Open Jobs file. The material cost transaction can be created and processed directly from an inventory withdrawal transaction, and so on as appropriate for such other functions as billing, payroll, general ledger and machine maintenance.

The Job Shop Control software was designed by Darryl Johnson of Interactive Management Systems, Inc. It conforms to the standards and interlocks with the other business applications maintained by that organization.

REPORT NO 14 WEEKLY LOAD REPORT PAGE 1
APRIL 3, 1978

WORK NUMBER	CENTER DESCRIPTION	LOAD WEEK	PLANNED CAPACITY	EST HRS LOAD: INSHOP	& TOTAL	TOT LOAD CHANGE	HOURS CURRENT	HOURS OVER FLO	NO OF JOBS
1050	SPIN	CU&BK	44.0	65.0	68.0	10.5-	68.0	.0	14
		13		25.0	45.5	16.4-	40.0	5.5	1
		14		24.5	39.6	5.1	39.6	.0	11
		15		22.1	34.5	4.5	34.5	.0	9
		16		16.0	38.7	8.6	35.0	3.7	12
		17		11.5	29.2	1.2	29.2	.0	10
		18		10.1	28.1	1.7-	28.1	.0	9
		19		0.0	21.5	6.5	21.5	.0	7
		20		5.5	15.1	5.1	15.1	.0	6
		21		0.0	21.2	13.1	21.2	.0	6
		22		0.0	15.5	8.0-	15.5	.0	4
		23		0.0	16.5	.7	16.5	.0	4
		24		0.0	5.5	5.5	5.5	.0	1
		BAL		0.0	0.0	5.5-	0.0	.0	0
1050	SPIN	TOTAL		199.7	378.9	1.5			94
1060	BENCH	CU&BK	40.0	38.0	42.5	10.0-	42.5	.0	18
		13		37.5	44.1	1.1	44.1	.0	21
		14		16.1	29.0	6.0	29.0	.0	16
		15		18.5	30.1	5.2	30.1	.0	17
		16		25.0	33.5	4.5-	33.5	.0	20
		17		24.7	33.0	3.0-	33.0	.0	19
		18		13.8	21.0	6.0	21.0	.0	15
		19		16.7	21.9	5.0	21.9	.0	16
		20		15.0	25.0	7.5	25.0	.0	14
		21		11.1	11.9	.0	11.9	.0	11
		22		.0	9.4	1.4-	9.4	.0	10
		23		.0	4.1	2.0	4.1	.0	5
		24		.0	3.9	1.4	3.9	.0	5
		BAL		.0	1.2	1.2	1.2	.0	1
1060	BENCH	TOTAL		235.0	204.1	12.5			188
	FDRY	TOTAL	84.0	434.7	561.0	14.0			

Figure 1

REPORT NO 5 DAILY SCHEDULE PAGE 1
DEPT 1013 20 1978

WORK CTR	JOB#	DESCRIPTION	OPN#	QUAN TITY	PR NO	SCH DATE	EST HOURS	TOTAL HOURS	PREV W.C.	#OPNS REMNG	HOURS REMNG
1031	1767	WM-5531	2	50	2	11-30	6.5			8	6.7
	1824	WM-8118	2	50	2	12-01	4.0			6	8.0
	1821	WM-8001	3	10	2	12-02	1.0			7	10.2
	1858	14003	2	75	3	11-29	15.2			5	1.1
	1845	DMS1655	3	15	3	12-02	4.0			10	20.1
	1803	DRMC-1781	2	25	3	12-04	8.7			11	6.0
	1871	WM-8110	3	5	4	12-03	3.0			17	.9
1031					TOTAL AT WORK CENTER			42.4			
	1759	WM-5557	2	150	1	11-29	20.0		1110	8	18.1
	1951	WS-328	2	400	2	11-28	28.5		1110	2	6.4
	1904	WS-1065	20	50	2	12-09	2.0		2118	10	35.0
	1909	WS-1067	20	75	3	12-10	3.6		2116	8	39.5
					TOTAL PENDING			54.1			
1133	1650	WM-5532	30	100	1	12-04	2.0				
	1784	WM-8119	35	150	1	12-11	3.0				
	1675	WM-8114	61	60	1	12-11	1,0				
	1706	WS-1055	25	800	2	11-30	12.5				
	1781	WR-9251	35	750	2	12-04	11.5				
	1802	WM-5514	30	25	2	12-11	1.0				
					TOTAL AT WORK CENTER			31.0			
1133	1709	WS-1011	25	1000	1	12-11	14.0		125	5	120.5
	1815	WR-9225	15	100	2	12-04	2.0		1015	9	36.8
	1792	WR-9415	30	200	2	12-11	4.0		1015	7	44.5
	1764	WM-1855	20	50	2	12-18	1.0		1010	5	13.6
	1780	WS-1069	25	85	4	11-30	1.0		1115	10	21.0
					TOTAL PENDING			22.0			
DEPT TOTAL AT WORK CENTER								73.4			
GRAND TOTAL AT WORK CENTER								465.4			

Figure 2

REPORT NUMBER 6 JOB COST ANALYSIS PAGE 1

PART NUMBER WS-4555
QUANTITY ORDERED 25 JOB NUMBER 8881 DATE COMPLETED 03-08-78

TR CD	OP NO.	WKCTR NO.	QUANT COMPL	UNIT-COST $EST	UNIT-COST $ACTL	* LOT VARIANCES * STAND.	TEMP/SUBS	REWORK $LABOR	REJECT $LABOR	SCRAP $LABOR	SCR/ REJ	COST BRTFWD	OPN DATE	MATL $	CUM ACTL LABOR $
1	01	522A	25	10.00	16.00	150.00							01-11	125.00	400.00
2	2A	F402	25	33.00	36.02		75.50		215.00		5		01-15		1,085.50
2	4B	25A	20	2.50	6.04		70.80						01-21		1,206.30
1	05	311B	20	2.50	1.99	10.20-							01-31		1,246.10
1	06	7651	20	1.00	.96	.80-							02-09		1,265.30
8	06	BENCH	6		1.00			6.00					02-09		
6	07											200.00	02-17		1,465.30
1	07	181		2.75	2.70	1.25-							02-20	30.00	1,532.80
1	08	54	25	5.00	9.80	120.00							02-21		1,777.80
8		311B	5					8.00					02-21		
8		F402				45.10			215.00-	215.00	5		02-22	30.00-	
LABOR $/UNIT				56.75	69.76										
FINAL TOTALS			25			212.75	146.30	14.00	.00	215.00		200.00		125.00	1,777.80

VARIANCE ANALYSIS:

VARIANCE ON STANDARD OPERATIONS	212.75	
VARIANCE ON TEMP/SUBSTITUTE OPNS	146.30	
TOTAL VARIANCE		359.05
STANDARD LABOR 25 x 56.75		1,418.75
MATERIAL $		125.00
GRAND TOTAL DIRECT CHARGES		$1,902.80

OVERHEAD CHARGES:

SCRAP	215.00
REWORK LABOR	14.00
TOTAL	229.00

Figure 3

REPORT NUMBER R-1

OPEN JOBS LIST
SELECTED BY JOB NO. FROM 30651 TO 30700

APR 1, 1978

JOB NO.	LST PFX	ST	PR	CUSTOMER P. O. NO.	CUST#	JOB DUE DATE	** QUANTITIES ORDRD	GOOD-LOC	REJECT	** OPNS REMG	LST OPN	CUM S.U. HRS EST.	ACTL	CUM PRODN HRS EST	ACTL	CUM LABOR $ EST	ACTL
S30651		5	2	AJ506	20110	3-28-78	500	510	0	20		3.0	2.0	10.0	9.0	95	87
30652		5	2		11051	3-10-78	25	25	0	2	70A	15.0	14.1	86.6	79.9	865	801
30653		5	3	P14A-301	8502	3-10-78	600	595	5	3	65T	40.0	10.0	40.0	42.5	400	435
T30654	S	5	3	21602	7525	3-18-78	1100	1200	50	4	36	105.5	105.5	1017.7	1016.6	10850	10995
30659		5	1	41710	10106	3-31-78	100000	99800	200	2	20A	10.0	6.0	56.0	51.0	525	499
30667		4	3	13223	10233	4-04-78	150	0	0	10		.0	.0	.0	.0	0	0
30676		5	2	1046	20101	4-28-78	500	510	0	9	50	2.9	3.4	19.9	22.5	211	235
30677		5	2	56707	10545	4-25-78	100	100	0	7	50	2.1	2.2	25.0	22.5	240	237
30682		5	2	45756AX	10355	4-04-78	1000	1010	40	4	70	17.5	10.2	66.5	60.2	700	625
30686		5	1	19009	9875	4-13-78	1500	1525	25	2	65	12.7	11.5	125.0	107.6	1275	1017
30695		5	3	8564P	0235	4-04-78	100	100	5	1	80	9.5	9.8	190.0	180.5	2115	1995
30696		3	5	AP1013	8242	3-31-78	50	0	0	10		.0	.0	.0	.0		
30697		2	5	X4350	10174	4-07-78	25	0	0	10		.0	.0	.0	.0		
30698		5	4	19876	10125	4-25-78	150	150	0	9		.8	.6	4.0	4.0	48	47
30699		4	3		10974	4-18-78	1100	0	0	15		.0	.0	.0	.0		
30700		1	3	75100-AY-112	10435	5-05-78	250	250	0	0	12	.0	.0	.0	.0		
GRAND TOTALS												213.0	175.3	1640.7	1598.3	17324	16773

LOAD HOURS REMAINING-SU 60.5
LOAD HOURS REMAINING-RUN 965.0

Figure 4

QDMS: A DATA MANAGEMENT SYSTEM UNDER RSTS/E

P. Tofil, C. Darling, T. E. Moriarty, R. B. Enders, P. J. Cruson
Quodata Corporation
Hartford, CT 06103

ABSTRACT

The QDMS data management system was designed to meet two needs. First, it gives non-technical users access to the computer with an easy-to-use report writer. Second, QDMS provides programmers with a 'tool box' that substantially reduces the time necessary to create applications software.

A data dictionary provides seventeen attributes for each field. The file structure includes separate descriptor, index, sequential pointer, special index, and data files. QDMS provides multi-keyed ISAM, special/selective pointer files, file inversions and multiple views of the data file. Up to five files will be linked via common keys.

Actual sorting is handled by a runtime sort. Sorting with or without selection can create new index and pointer files. A multi-file version is used to create a linked pointer file for use in reporting. The report generator may be used for 'ad hoc' as well as reusable report creation. Batch and on-line modes are allowed for the report description process.

Software development under RSTS/E can be greatly enhanced with a ready set of tools. The tools take the form of BASIC-plus functions, which provide a standard means of associating the logical interfile relationships and file accessing required. Thus, programmers can concentrate their efforts on application needs, rather than the mundane data base requirements.

INTRODUCTION

As minicomputer systems have gained wide acceptance in the user community throughout the seventies, the use of computer technology is no longer reserved for that elite group of technicians residing in corporate data centers. Top management is demanding and expecting to receive an extension of the computing resource to non-technical personnel in their organizations. Put another way, management is refocusing responsibility for corporate data resources back to the actual users of such data. These users may be clerks, managers, and in some cases line executives who use data and the information it yields to operate on a day-to-day basis.

The challenges are clear. Technicians of today will be called upon to produce highly sophisticated software which will be operated in what they will perceive to be highly unsophisticated environments. QDMS, running under the RSTS/E timesharing system is a step toward putting responsibility for management of information back into the hands of the user. The key feature of QDMS that brings information closer to the user is the report writer, which is used by non-technical personnel. It also provides a sophisticated report generation facility that replaces programming in many cases.

SYSTEM STRUCTURE

QDMS can logically be divided into three major segments for the purpose of more clearly understanding the total functionality. System modules have been designed to provide capabilities to three varieties of user personnel.

1. Managerial (system managers)
2. User (clerical or operating personnel)
3. Programmer

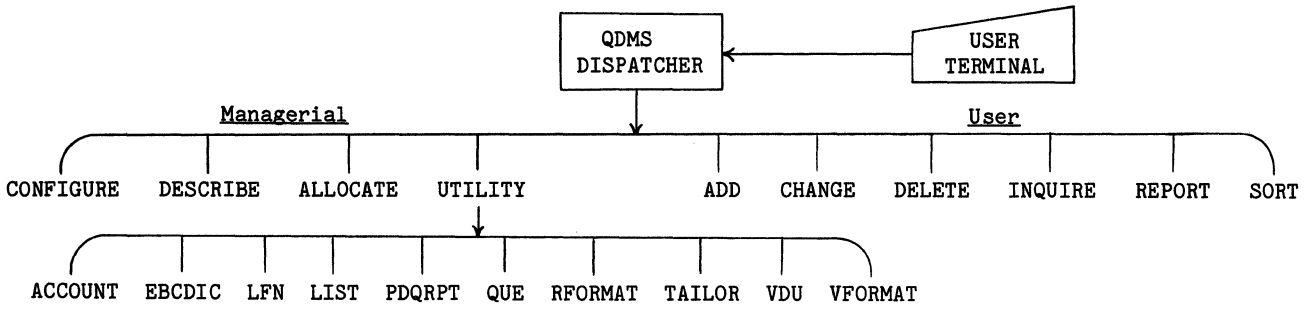
The various system components are illustrated by functional area in Figure 1. Note that access to all modules of the system is controlled by the dispatcher. The dispatcher functions as an interface between the user and the various QDMS command levels. This dispatcher concept keeps the system relatively easy to use for the user by funneling user requests through 'tiered' levels of the QDMS command structure. By issuing EXIT commands the user can return to the dispatcher which will then be ready to receive the next command request.

Managerial Modules

Certain functions within QDMS are the domain of the system manager or person designated as the data administrator within a given PDP-11 site. Figure 1 shows these to be the **CONFIGURE, DESCRIBE, ALLOCATE,** and **UTILITY** modules of the system. These modules

Figure 1

QDMS SYSTEM STRUCTURE

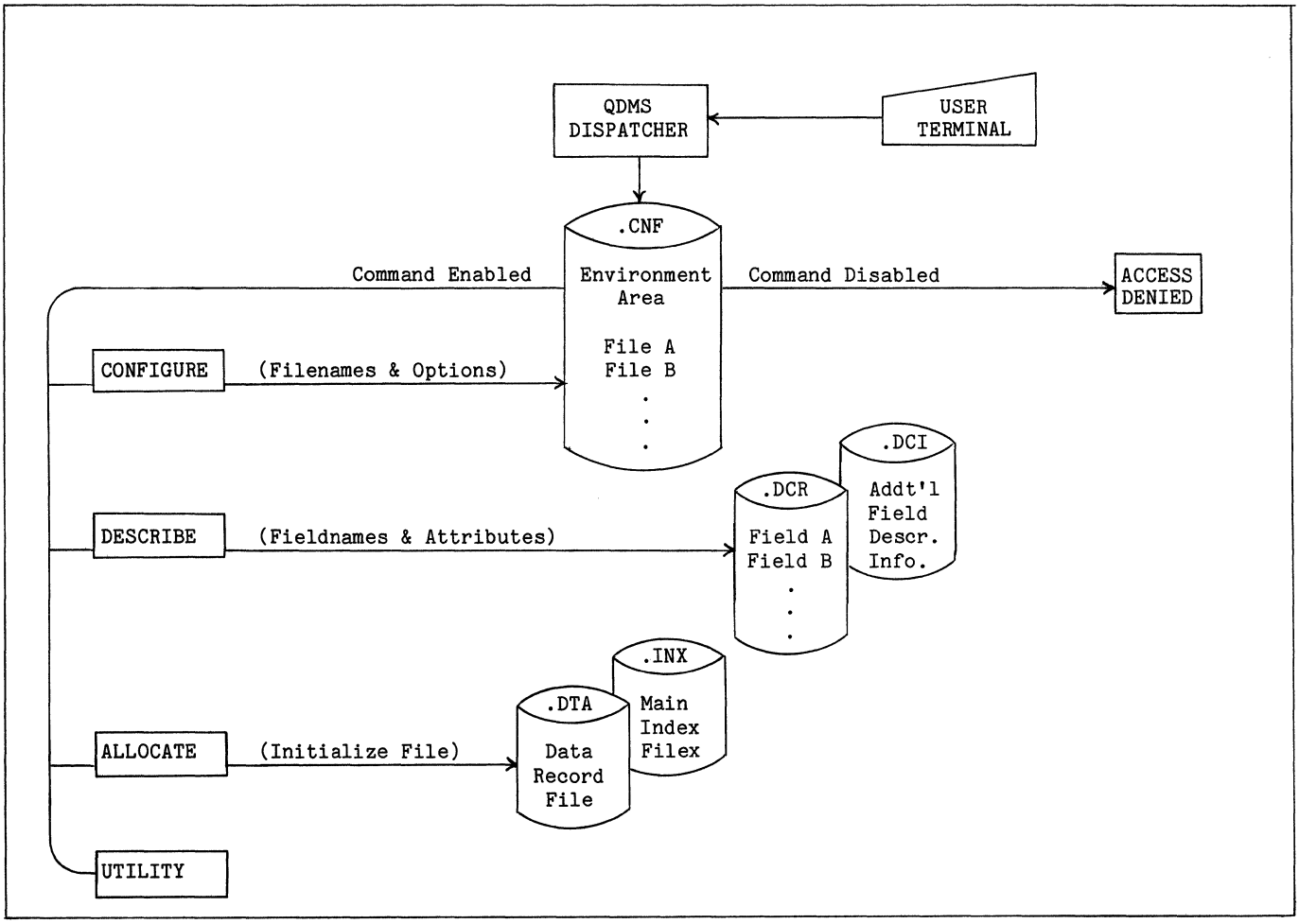


Programming

The **Building Block Library** consists of a comprehensive set of callable pre-programmed functions in source format. The functions are for use by applications programmers when designing specific software modules or when interfacing same to a QDMS data base.

Figure 2

QDMS MANAGERIAL COMMANDS



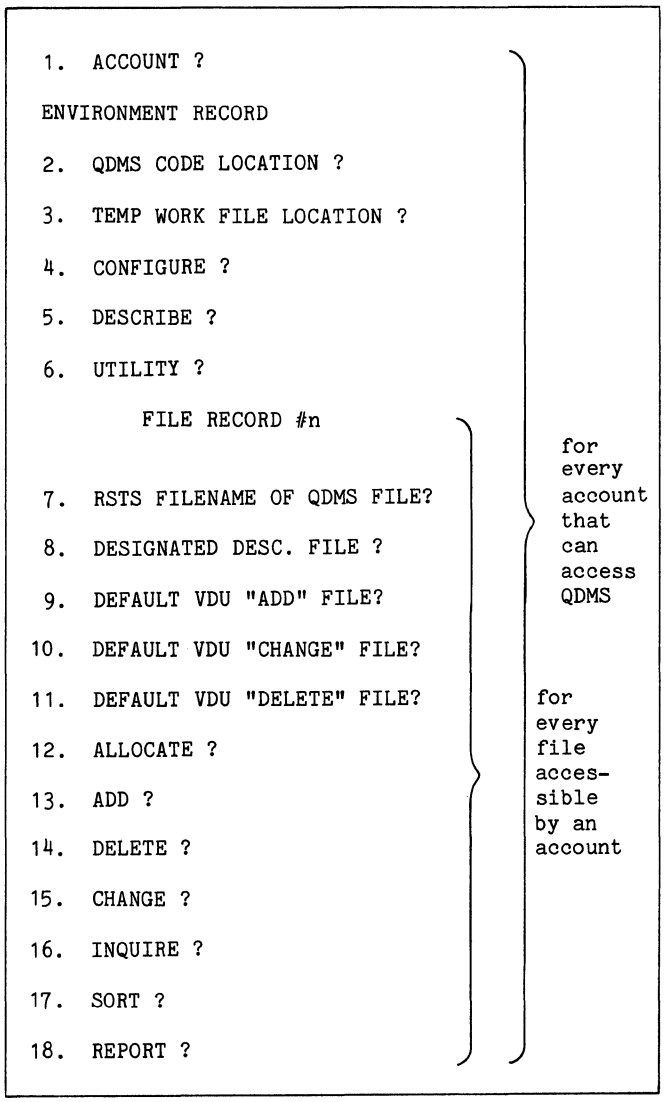
might also be accessed by programmers or analysts, but will most likely be restricted from access by non-technical users.

The **CONFIGURE** option (Figure 2) establishes accounts that will be allowed access to QDMS commands and files. Each user (account) on the system must have a configuration file. This file offers the next level of system security beyond RSTS which limits access to files if required, and controls the commands that can be employed by any given user when interacting with QDMS files. If a user succeeds in logging into QDMS, but does not have the appropriate configuration file, further access is immediately denied.

As each user is configured to run QDMS the files that are allowed to be accessed are designated. In addition the various command modules are either made available to that user or disabled on a selective basis. This provides further protection against unauthorized use of the data base. Configuration files are built and can be changed on an interactive basis. Figure 3 is an illustration of the dialogue necessary to configure a QDMS user.

Figure 3

CONFIGURE QUESTIONS HIERARCHY



The **DESCRIBE** option allows 'windows' into a given data base for each user. Any combination of fields in a data base may be defined in a user's descriptor file. A number of users may have the same data elements in their respective descriptor files or they may be tailored uniquely for each user. In this manner it is possible for several users to maintain different subsets of the main data base. This also provides an additional level of security by omitting those elements in the main data base from a user's descriptor file that he or she is not privileged to see. There is the capability when creating a descriptor file to assign seventeen different attributes to each QDMS field described.

Figure 4 is a listing of the first two fields in a twelve field customer master file. As items fourteen and fifteen in the field attribute table indicate, there is read and write protection at the field level which provides further data security. Within the seventeen attributes there are such capabilities as range checking, default value, edit checking, and an on-error value which is used when loading data in batch mode. The descriptor files like the configuration files can be created interactively and dynamically altered by privileged personnel should the need arise.

The **ALLOCATE** command is used to reserve necessary disk space required for each file. This command is used when the file is initially constructed or can be invoked at any time to extend an existing file. Space is allocated either by specifying the number of logical records or indicating the number of disk blocks to be used. Logical record sizes can range from 32 bytes to over 2000 bytes. Maximum possible size of a QDMS file is 33,553,920 bytes.

The **UTILITY** options are a series of modules designed to assist all three categories of users. Each utility performs a specific task when invoked. The utility command is accessed via the dispatcher at which time the specific utility is accessed by name.

Figure 5 shows a list of the utility commands and the function of each. Of particular importance in the utility repetoire are the RFORMAT and PDQRPT modules. It is via these modules that the user obtains access to the comprehensive report writer capabilities of the system. Although classified under managerial modules, these components, especially PDQRPT, will be heavily used by non-technical personnel for report generation.

The preceding components of the system constitute the tools available to people charged with maintaining the integrity and security of the QDMS environment. Some will be employed by the system manager while others will be of use to technical supervisors responsible for the quality, content, and security of all QDMS files on the system. The next section deals with those components available to the end-user or non-technical person.

User-Oriented Modules

The dispatcher accesses certain programs which provide end users with the necessary file maintenance,

sorting, and reporting capabilities for maintaining and using data files independent of technical personnel. These modules are

ADD
CHANGE
DELETE
INQUIRE
SORT
REPORT

Figure 4

DESCRIBE COMMAND LISTING

```
QDMS DESCRIPTION FILENAME ? QDMS:CUSTMR.DCR

***  QDMS:CUSTMR.DCR IS CURRENTLY DESCRIBED

OPTION ? LIST

 2.  OUTPUT TO ? $KB:


FIELD # 1                          ID.NO

ID NUMBER

 3.  STARTING CHAR POSITON         1
 4.  FIELD LENGTH                  3
 5.  DATA TYPE CODE                3
 6.  COMPARE TYPE                  NUMERIC
 7.  EDIT TYPE                     NUMERIC
 8.  CHECK LIMITS                  NO
 9.  LOWER LIMIT FIELD VALUE
10.  UPPER LIMIT FIELD VALUE
11.  DEFAULT VALUE                 999999
12.  ON ERROR VALUE                999999
13.  SORT ELIGIBLE                 YES
14.  READ PROTECT                  NO
15.  WRITE PROTECT                 NO
16.  KEY FIELD VALUE               1
17.  PRINT MASK


FIELD # 2                          L.NAME

CUSTOMER'S LAST NAME

 3.  STARTING CHAR POSITION        4
 4.  FIELD LENGTH                  12
 5.  DATA TYPE CODE                7
 6.  COMPARE TYPE                  STRING
 7.  EDIT TYPE                     NO
 8.  CHECK LIMITS                  NO
 9.  LOWER LIMIT FIELD VALUE
10.  UPPER LIMIT FIELD VALUE
11.  DEFAULT VALUE
12.  ON ERROR VALUE                *****
13.  SORT ELIGIBLE                 YES
14.  READ PROTECT                  NO
15.  WRITE PROTECT                 NO
16.  KEY FIELD VALUE               0
17.  PRINT MASK


Note:  Underscored text is entered by user.
       $ indicates ESCAPE key pressed.
```

Figure 5

UTILITY COMMAND SUMMARY

UTILITY	FUNCTION
ACCOUNT	Provides system statistics including block usage, record length, number of records on file, and dates and times of last access.
EBCDIC	Converts EBCDIC magnetic tape files to ASCII disk files to be accessed by QDMS.
LIST	Lists ASCII formatted files or copies files.
LFN	Maintains the logical file name table enabling users to access files by mnemonic name.
QUE	Connects the user to the RSTS/E QUEMAN system.
VDU	Maintains a keyboard table indicating which terminals are video display units.
VFORMAT	Used to format a VDU screen display. Once formatted, displays can be stored and recalled at will.
RFORMAT	Creates control files used by the report command control files, determines fields to be used and output formats of reports generated. This is a major system utility.
PDQRPT	Links to the RFORMAT utility when an ad hoc or one-time report is desired. Functions as a short form version of the dialogue necessary to generate a report eliminating much of the interaction through system defaults.
TAILOR	Changes the size of the user area available for QDSORT user. Specifies sort area size between 7 and 28K.

ADD - This module functions in either the batch or on-line mode. It is used to add records to an existing file or to a newly created file. The batch add will accept data from as many as four different files either in ASCII format or as RSTS/E Type 1 files. The batch add feature minimizes conversion time of existing files into QDMS format. In order to take advantage of the batch add, the user must have previously configured, described, and allocated a QDMS file. In addition, it will be necessary to create a batch control file. The batch control file may be constructed interactively prior to running the add program.

The on-line add program is entirely interactive with terminal prompts to guide the user. A selective add process may be predetermined at the beginning of a terminal session. Prompts will be given only for those fields requested in each record. During the add process data will be checked against edit

parameters as defined in the descriptor file for each field. Unqualified data will be rejected and the user will be prompted accordingly. A verification option is included if record verification is desired.

The user may add data via sequential conversational field prompts or construct a video display screen format. Screen formats may be altered at any time. They can be stored and recalled for future sessions. At termination of an add session, pointer files to the main data file are automatically rebuilt reflecting the most current condition of the file.

CHANGE - This module is employed to alter the contents of fields within records on a QDMS file. Invoking the change command will retrieve the first record on the file. Further record retrieval is based on an Index Sequential Access Method (ISAM). A record may be accessed by key field or by its relative position to the 'current' record. Valid change commands are GET, SHOW, CHANGE, NEXT, and MARK. The GET command will retrieve a record by specified key. The SHOW command will display requested fields. The CHANGE command alters the contents of a field. The NEXT command will retrieve the next record on the file. NEXT plus or minus N, where N is an increment of records, will move the user backwards and forwards in the file. MARK allows the user to change delimiters within a field. VDU screen formats may be used with the CHANGE command.

DELETE - One or more records may be deleted from a QDMS file. Valid commands are DELETE, NEXT, GET, SHOW, and MARK. The latter four commands have been described under the change function. A series of records may be deleted relative to the current record either forward or backward in the file for N records were N is the count of records to be deleted. The video display option may be used in conjunction with the DELETE command. Records are not physically removed from the file. The deleted records are flagged and physically deleted when the file is sorted and reordered.

INQUIRE - A file may be examined on a record by record basis. Valid functions include GET, NEXT, SHOW, and MARK as described in the change module. The inquire command is most commonly used for low volume, limited inquiry. A more comprehensive form of inquiry is offered using sort selection and PDQRPT described in a succeeding section.

SORT AND REPORT - These commands are sufficiently powerful to warrant a separate section and will be treated there.

Summary - In general the user interface offers a series of easy-to-use command modules to the end-user. Terminal sessions are tutorial in nature. Ample use of help text is employed. At any point in the system the user may type 'help' or depress the line feed key to obtain help messages. Through the use of prompts, help text, and the user manual, the end-user should be able to accomplish full data management functions with a minimum of assistance from technical personnel.

Programming Tools

Analytical and programming personnel represent the third category of users for whom QDMS was designed. There exists within the system structure a comprehensive set of software development tools referred to as building blocks. The building blocks represent nearly three dozen general routines that are common to software systems. These routines have been hard-coded and combined in a library from which each is accessible via a one line function call. The functions accept a fixed number of parameters each time they are called, and return a single value when completed.

The purpose of the QDMS Building Block functions is to reduce the programming required to access a QDMS data file. Using index and/or pointer files and the QDMS Building Blocks, most file access can be programmed as a series of function calls. This means that the programmer never has to worry about using BASIC-Plus to open files, read or write blocks, deblock buffers, translate encoded data, etc. All these things can be accomplished by simple function calls. The functions are divided into two general areas. There are database functions and index functions. A list and brief explanation of each is contained in Figure 6.

Figure 6

QDMS BUILDING BLOCKS FOR PROGRAMMERS

Database Functions

FNA5	Add Record
FNB5%	Retrieve Start of Field
FNC5%	Close File
FND5%	Open Descriptor File
FNF5$	Extract Field
FNG5%	Get Record
FNH5%	Decrement Header Record
FNI5%	QDMS File Initialize
FNJ5$	Justify Field Image
FNK5$	Build Key String
FNL5%	Retrieve Length of Field
FNM5%	Delete Record
FNN5%	Retrieve Number of Named Field
FNN5$	Retrieve Name of a Field
FNO5%	Open File
FNP5%	Change Field in Buffer
FNT5%	Retrieve Field Data Type
FNU5%	Replace Record on Database
FNV5	Access Pointer File
FNX5$	Translate into ASCII
FNY5$	Translate from ASCII
FNZ5%	Utility Function

Index Functions

FNA6	Add Record
FNC6%	Close File
FND6	Delete a Record by Key
FNI6%	Create and Initialize
FNK6$	Retrieve Entire Key
FNN6	Retrieve Pointer to Next Record
FNO6%	Open File
FNP6%	Put Record
FNR6%	Change Main Pointer in Current Record
FNS6	Search for a Key
FNT6%	Position Pointer to File End or Beginning

SORT AND REPORT WRITER

THE SORT

A significant part of any data management system continues to be the sorting and reporting capability. These QDMS modules were designed to provide optimum power and flexibility which allow end-user personnel to obtain selected output in any format without having to consult programmers or other technicians.

The QDMS sort operates as a RSTS runtime system which minimizes sort times. The sort creates para-files consisting of pointer and index files which point to actual data files or selected subsets of data files. The user can create multiple pointer files which will redefine the order and content of data subsets. The pointer files are accessed by the report writer at report generation time. There is one index file associated with the main data base. However, the sort may be used to create additional index files to be used by programmers in conjunction with the QDMS functions or building blocks.

In most instances the sort functions as a key or tag sort. A reorder option allows physical sorting and reordering of the main data file optimizing space utilization and speeding data access. A comprehensive selection module provides up to fifteen levels of Boolean selection criteria with six relational expressions. Sort and selection dialogue may be saved in a sort control file and invoked for subsequent runs. The sort/selection process provides the report writer with input flexibility for data sets to be displayed during report generation.

REPORT Writer

Most users judge software systems by the ease with which output can be obtained. Careful attention was given to the report writer segment of the QDMS system. The design called for a completely interactive report system directed at two categories of users. The report generator was built to offer maximum flexibility in features and operation for programming personnel. The purpose of this design goal was to virtually eliminate the need for creating application print programs. An equally important provision was the short form 'ad hoc' interface which allows non-technical users complete access to the report writer facility.

The two interface modules known as RFORMAT and PDQRPT generate report control files which may be saved and recalled for subsequent report processing. Existing report control files generate corresponding batch files which may be altered via any editor to modify existing report formats or generate unqiue formats with a minimum of interaction.

The report writer references pointer files created by the QDMS sort to allow report production in any order of any subset of the data base generated through the sort and select option. Output from the report writer is device independent, and in fact, may be directed to a peripheral other than a printer. This option provides a vehicle for the generation of QDMS sub-files from the main data base. The table transformation feature and calculation modules combine to provide for the generation of data items not contained in the main data base. Literals and special keywords augment this facility.

In summary, English-like declarative statements, liberal use of embedded help texts, and numerous default responses make the report writer easy to use as well as readily acceptable by non-technical users. Programmers on the other hand will find the long form version (RFORMAT) a useful tool in generating sophisticated reports without having to code programs. Figure 7 contains a summary of report writer features.

Figure 7

QDMS REPORT WRITER FEATURES

1) Batch and on-line description process (batch input file automatically created via on-line session)

2) Any combination of various line types (headings, details, controls, footings, and grand totals)

3) Report Control File creation for reuse of same report

4) Table transformations (on a one-to-one or a range basis)

5) Complete variable horizontal/vertical spacing (including overprint)

6) Four types of source fields allowed (QDMS fields, literals, calculations, and special keywords)

7) Miscellaneous features:

   - Auto-totalling
   - Cross field calculations (any valid algebraic expression)
   - N-UP reports (mailing lables, form letters, etc.)
   - N-level control breaks
   - Automatic pagination (page numbering and date stamping)
   - Averaging
   - Numeric, Alphanumeric, and floating point print masks
   - Device independence
   - Scroll control option
   - Operator information (forms setup, cover page, and report disposition information)
   - Ad hoc interface (PDQRPT)

Concluding Remarks

As conceived, developed, and herein presented, QDMS represents a 'tool box' approach for extending file management, record retrieval, and data reporting techniques to the unsophisticated user. Through the use of managerial and programmer oriented modules, work effort is at least considerably reduced, and at best almost eliminated. Future releases of the system will embody such features as linked file technology allowing a user defineable file linkage schema which operates across as many as five different files for sorting and reporting purposes. This version (V3) is in final test and scheduled for release in early 1979.

Multiple key access has recently been added to the **CHANGE, DELETE,** and **INQUIRE** modules in the form of a LOAD command. This feature allows the user to access records via an alternate index file. Thus, the user may select any field or combination of fields, execute the sort to create a new index file, and then retrieve records based on these new key values.

Future release specifications are today being assembled which will allow applications development under RSTS in virtually a programmerless environment. Care will need to be taken to effect the proper economy-in-scale providing RSTS users with an understandable, manageable, and affordable alternative to their data base requirements while maintaining upward compatibility with future releases of RSTS/E.

OVERLAP SEEK DISK DRIVER

Paul J. Wirtz
Cytrol, Incorporated
Edina, Minnesota

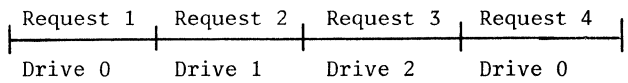
The Overlap Seek Disk Driver (OSDD) is a software module that replaces the disk driver in the RSX-11M operating system for the PDP-11. Separate modules exist for the RK06/07, RM02/03, and RP04/05/06 disk subsystems. In addition to the functions performed by the DIGITAL supplied module, the Cytrol drivers make use of hardware features already in the disk controllers, but not utilized by RSX-11M. The additional features include:

1) Initiation of I/O requests for all drives in parallel.

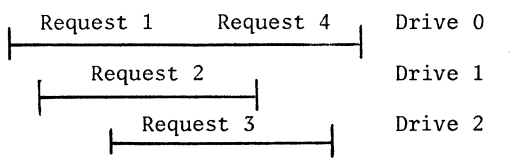
2) Use of the search command to reduce controller busy time (RM03, RP04/05/06).

3) Accommodate hardware arbitration of dual-ported drives.

The parallel initiation of requests to drives on the same controller can substantially increase the rate at which the disk subsystem can process I/O. It does this by overlapping the seek (head movement) operations on several drives at the same time. Because, on the average, the seek time is about 75% of the total request completion time, seeking drives in parallel increases the number of requests that can pass through the disk system. It is a lot like having several checkout lanes at the supermarket. Even though every customer goes out through the same door, having several checkout lanes increases the number of customers that the store can handle and reduces the average waiting time per customer. The current disk drivers for RSX-11M process all disk requests serially, waiting for each request to complete before initiating the next one. The OSDD allows requests to different drives to start at the same time, even though they will finish one at a time.



Processing disk requests with the standard RSX11M disk driver



Processing disk requests with OSDD

The maximum achievable throughput depends on the number and type of drive in the system.

Subsystem Type	Number of Drives 1	2	3	4	5	6 or more
RK06	100	185	240	265	265	265
RK07	100	185	235	260	260	260
RM03	100	190	280	365	440	510
RP04	100	190	280	360	430	430
RP05/06	100	195	285	365	440	500

Maximum disk subsystem throughput (percent of non-overlapped)

The actual throughput depends on a number of factors including:

1) The distribution of accessed data within and between drives.

2) The number and kind of active tasks.

3) The way in which FILES-11 capabilities are used. (Some choices can cause significant serialization before requests reach the driver.)

The best situations for increased performance are:

1) When there are many tasks (users) whose accesses are distributed across several disk drives.

2) When a single task initiates concurrent I/O to two or more disk drives.

Each OSDD module is a direct replacement for the standard DIGITAL disk driver, and can assemble with all the options that the standard driver does. A single copy of the OSDD will handle multiple controllers for the same type of drive. OSDD can also be built as either resident (part of the executive program) or loadable (linked to the executive dynamically by VMR and MCR). If error logging is selected, OSDD will log both device and timeout errors. Write-checking is supported and interleaved with other data transfer operations. User mode diagnostic can also be performed concurrently with normal system operation. Diagnostic mode data transfers are performed directly, giving the diagnostic program control over the subsystem during both the seek and the transfer. Offset recovery is also supported.

In addition, OSDD will make use of the dual-porting logic on a dual ported drive. If a request is to be processed for a drive that is seized by the other controller, the request is deferred until the drive is released. This scheme allows two CPUs to have access to the same disk drive. It should be noted that RSX-11M file software does not support the simultaneous updating of the volume. Because these conflicts can be avoided or synchronized, the dual access feature can still be utilized.

The installation of OSDD is simplifed by the fact it is not necessary to perform a complete system generation in order to include it in an existing system. OSDD is distributed as source code to be assembled with the parameter file created by the SYSGEN. This is the same procedure that would be followed if a correction were to be made to the DIGITAL driver. If a new system is to be generated, there are no changes to the system generation procedures. OSDD creates the additional data structures it needs when the system is bootstrapped for the first time.

Any future changes to OSDD will be automatically distributed to the affected registered owners. Software Dispatches are monitored for changes that would affect the OSDD modules and, if required, OSDD will be modified to keep it compatible with future releases of RSX-11M.

IAS TIMESHARING CONTROL SERVICES AND PROGRAM DEVELOPMENT

Eric A. Johnson
The Jackson Laboratory
Bar Harbor, Maine

ABSTRACT

The IAS Timesharing Control Services provide a means for one Timesharing task to initiate another. Intertask communication is possible via the TCS send/receive facility. The Jackson Laboratory has successfully developed a CLI based statistical package utilizing the Timesharing Control Services to minimize resource requirements and maximize program flexability.

The Timesharing Control Executive is that segment of IAS that monitors and supervises all timesharing tasks. The resources of the Timesharing Control Executive are available to the programmer in the form of a set of macros called the Timesharing Control Services(1). TCS allows the programmer to create programs that can initiate other programs, communicate with those programs, and control their execution. By using the TCS subtasking or chaining facilities the programmer can effectively reduce core memory requirements and allow for simplified debugging procedures.

The actual procedure for initiating a subtask is very similar and in most respects easier than opening a FCS file. A Task Descripter Block (TDB) must first be defined by including the macro TDBDF$ in the prospective owner task. This creates a 60 byte data area that, once it has been defined to TCS, becomes a workspace to which TCS assumes access. The TDB is declared to TCS at some point in the invoking task by the run time macro TDBD$T. TCS requires certain information in the TDB to perform its functions. This includes the address and the length of the command line and the type of command line to be used (more about this later). This information is inserted into the TDB by using one or more of the assembly or run time macros provided for this purpose. One of these macros, RUN$T, is then executed to queue the subtask for the timesharing scheduler. The minimum requirements for a timesharing task to initiate a subtask is summerized in Figure #1.

TCS offers flexability in intertask coordination through the use of event flags. TCS macros are available to check for events (CKEV$T), read events (RDEV$T), as well as setting the local event flags of a subtask. TCS events including subtask successfully terminated, subtask aborted, subtask suspended, initialization failure, messege sent, ect... allow the programmer to synchronize the functions of two or more tasks in a straight foward manner. Other macros are available which allow the owner task the capability of suspending(SPND$T), resuming(RSUM$T), or aborting(ABRT$T) a subtask on command.

```
        .
        .
        .
TDB1:   TDBDF$                                  ; Define the TDB buffer
        .
        .
        .
CMDBUF: .ASCII  " DB0:[11,36]MYSUBTASK"         ; The command line definition
        CMDBUL=.-CMDBUF                         ; and length
        .
        .
        .
        TDBD$T  #TDB1                           ; Declare the TDB to TCS
        .
        .
        .
        RUN$T   #TDB1,#CMDBUF,#CMDBUL,#TS.USE   ; Initialize the TDB...
                                                ; and Queue the task
        .
        .
```

Figure #1 Minimum requirements for Initiating a Subtask

The command line used to invoke a task must be an ASCII string in one of four formats. The first of these is a blank followed by a user file specification. A task invoked in this fashion will be "auto-installed" in the same manner as the run command is performed in PDS. The second command line format consists of the three character name of any task installed as ...ABC. This may be followed by a blank and a command. The third format is like the second, but for tasks installed as $$$ABC. Lastly, any 1-6 character name may be specified that corresponds to an installed task.

Thus, any system utility can be invoked with a command line from a user written task. For example, if given the command line, "PIP LP0:ONEWAY.LST" TCS would invoke PIP and stuff the MCR Command Buffer with the command. PIP would execute and issue the GMCR$ directive to receive the command, the same way as if invoked from PDS with an immediate command.

Any subtask can use TCS if it is passed the necessary privileges. If a subtask initiates a task, it is the owner of that task and has control over it. Accordingly, a hierarchyl system of timesharing tasks can be run and controlled from one terminal.

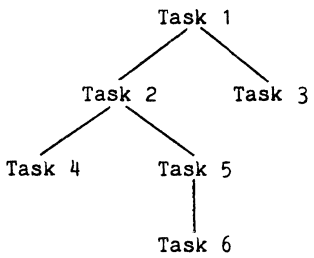


Figure #2 Task Hierarchy

Intertask communication is simplified for related timesharing tasks by the TCS send/receive facility. Whenever a chain task or a subtask requires information from its predecessor, or the owner task needs results from a decendant, messages of up to 255 bytes each, may be sent quickly and easily.

TCS also gives the programmer charge of program execution on the occurrence of a CTRL/C event. One task is designated as the CTRL/C task and is the only task notified of a CTRL/C event. On the occurence of a CTRL/C being typed at a terminal, an internal flag is set and any descendants of the CTRL/C task are suspended. The CTRL/C task can, at any time, test the flag to detect the event and take appropriate action.

Accounting is preserved by TCS. The subtask's statistics are returned to the issuing task's associate, TDB, and hence made available to it. The owner is automatically charged for the resources used by its decendants.

There are several system parameters that control a task's priviledges to access TCS(2). First, the user's profile must have the appropriate bits set in the two masks, TP1 and TP2; and the MTS parameter must be set for the maximum number of concurrently active tasks. The CLI under which the program is to be run (usually PDS) must be installed with adequate priviledges. Also, the terminals from which a system is utilizing TCS to perform some multitasking function must be allocated with the maximum task switch. In addition, when ever a subtask is to use TCS, the invoking task must give adequate priviledges via the TCS TDPR$A or TDPR$T macros. The priviledge structure of IAS proved to be the most difficult part of using TCS for this novice Macro-11 Programmer.

The Jackson Laboratory Computer Service provides applications programming in biomedical research as well as the business data processing associated with it's worldwide distribution of laboratory animals. The interactive statistical package used by the scientific staff originally was a set of Fortran routines held over from an IBM 1130 running a single user, card operating system. The individual programs were strung together in the conversion effort following the upgrade to a PDP11/45 in order to give the investigators direct access to the routines. It became obvious that this package was not fulfilling the needs of the staff. It was difficult to use, and there were many inefficiencies which made the program time consuming and expensive to use.

In restructuring the package, it was desired, in addition to an easy to use, standard syntax and more efficient operation, that additional analysis routines, regardless of source language, could be added without requiring extensive modification to existing routines. It was this language independence problem that turned us toward the CLI concept.

The resulting program (BOSS) was created as a driver for separate analysis programs. BOSS perfoms all syntax interpretation and preprocesses the data before subtasking to the specified analysis routine. Each analysis routine is responsible for the actual number crunching and output format only. They perform no interaction with the user themselves. Instead, the driver sends a message to the subtask and that task, through a subroutine call and two functions, receives any parameters and switch values that it may require. The analysis routine can be written in any language that supports a CALL statement. The advantages of separate programs for the analyses include simplifing debug proceedures and task overlay definitions. Also the user is now charged only for what he uses, not for the whole package.

The analysis routines often require the work file to be sorted and again, TCS was utilized. This time a subroutine call by the analysis routine causes TCS to provide the very efficient PDP-11 SORT utility with a command line and the work file is sorted.

After we found all of the priviledges that were required for TCS use, the development of the new package went quickly. The syntax was generated using Paul M. Cashman's MARGOT(3), a table-driven parsing algorithm. We found MARGOT very easy to use and have since found many more applications for it.

BOSS uses the IAS message output handler, MO, to handle Help and Error messages. This allows us to add messages to the package's help facility as we add analysis routines and keeps the driver down to a stingy 6K.

The flexibility that we realized with the statistical package driver suggests that any multi-function applications package requiring syntax interpretation could be very quickly and easily developed from a similar, though more sophisticated program. Developing a package from such an 'Omni-CLI' would involve simply defining the required syntax in a MARGOT-like language, and then writing the individual programs. Using the CLI concept with the support of the Timesharing Control Services, appears to be a very powerful programming tool.

I would like to thank Henry R. Tumblin for patiently answering an endless supply of questions. He taught me everything I know about MACRO-11. Also Randy Adams for his support during the project and for assistance in the preparation of this paper.

References

1. IAS Guide to Writing Command Language Interpreters, (Maynard,MA.: Digital Equipment Corporation, 1978) A complete description of all TCS macros may be found in this document.

2. IAS System Management Guide,(Maynard,MA.: Digital Equipment Corporation, 1977)

3. Cashman, Paul M. and Myszewski, Mathew I., MARGOT: A Macro-Based Generator of Command Language Interpreters; Proceedings of the Digital Equipment Computer Users Society, Vol. 3, No. 4, (Maynard,MA.: Digital Equipment Corporation, 1977)

AN IMPLEMENTATION OF BASIC USING MACRO-11

John Clemens
Labshare Division
Damon Corporation
Needham Heights, MA

A BASIC-language translator was developed at Damon Corporation including a macro implementation of an expression analyzer and code generator. The translator was developed on a PDP-11/60 running under RSX-11M, V3.0, and consists of a TECO pre-pass followed by a MACRO-11 assembly. The TECO pre-pass changes initial line numbers to legal labels and parentheses to angle brackets, and performs other substitutions. A macro-library contains a macro for each verb of the language and macros for parsing and generation.

Expressions are parsed from left to right with parenthetical precedence and can be of any degree of complexity if they fit on a line. The operators include logical, relational, and arithmetic; the atoms can be variables, constants, array elements, or function references. Variable types are single- and double-precision integer, single- and double-precision real, string, and byte. Multi-dimensional arrays are implemented for each variable type.

Three stacks are maintained during the macro-assembly process-- one for generating and recycling temporary locations, another for retaining a certain bit of information at each level of recursive evaluation, and a third for the correct processing of embedded FOR-- NEXT loops.

BASIC verbs using the expression analyzer are:

```
LET   atom = expression
FOR   atom = expression TO expression (STEP expression)
IF    expression THEN statement
ON    expression GOTO/GOSUB line1, line2,....lineN
```

This development was not too costly in time and effort and suggests that other languages could similarly be implemented.

INTRODUCTION

The Labshare group at Damon has been given the goal of developing a data-management system for clinical laboratories. (Damon maintains a number of such facilities from coast to coast). We selected the PDP-11/60 to develop and run the system on, with the option of running smaller labs with an 11/34. The application is one in which speed is part of the product and volume the name of the game; the system is largely disc-file intensive, with only minor computational requirements. In order to control our own file-accessing techniques and to maximize efficiency, we developed our own operating system, which was tailored for this need. We also had a large number of application programs to develop in a reasonably small period of time and felt that the job would be easier if they were coded in a high-order language.

We had used MACRO-11 running under RSX-11M for developing the operating-system programs, and this experience gave us the confidence to believe that we could implement a quick-and-dirty version of BASIC using macros, a version that would "compile" under RSX and produce programs that would run using our system. We did this, and it was fairly quick and not too dirty. There follows a description of the language itself and then a description of how some of its features were implemented.

LANGUAGE DESCRIPTION

General

The language we developed was not ANSI 1978 BASIC (whatever that is), nor was it Dartmouth BASIC, nor even DEC's BASIC +2. Almost all of the features common to all dialects were implemented; those that were

not usually had the property of being both difficult to implement and useless for our application. A number of additions were made to the language--some specific to our application and operating system, others of general data-handling usefulness.

The core BASIC verbs implemented were:

```
CHANGE    LET
GOSUB     RETURN
GOTO      REM
FOR expression TO expression STEP expression
IF  expression THEN statement
ON  expression GOTO/GOSUB
END
```

INPUT and PRINT were implemented in a non-standard (and more sophisticated) manner. The whole complex represented by the READ, DATA, and RESTORE statements was not implemented, although in this case they would not have been difficult to implement--just not useful. With separate data-defining statements (BYTE, INTEGER, DOUBLE, FLOAT1, FLOAT2, STRING), a programmer can specify variables of different types, and each of these variables may be dimensioned as a single- or multi-dimensional array or else undimensioned. These statements obviated the need for a separate DIM statement. Expressions of arbitrary complexity may be used but precedence is established only with the use of parentheses and all statements must fit on a line (which in our version of MACRO-11 seems to be about 120 characters). Most of the usual functions were implemented--especially the string-manipulation functions--as were a number of unusual ones. The MAT statement was not implemented, owing to lack of interest.

ENHANCEMENTS

The INPUT and PRINT statements were designed to be used with an IMAGE statement (fancier than the one in BASIC +2) and an exceptional return. The exceptional return in the "PRINT" statement indicates that the bottom of a page has been reached, so that the programmer can choose to have a heading printed at the top of the next page. The exceptional return in the "INPUT" statement allows an operator to change the normal course of an application by hitting the ESCAPE key at a point where he or she is being asked for some data.

The formats of the INPUT and PRINT statements are the same except that the PRINT statement has an additional argument (channel number) for printing on multiple devices. A sample INPUT statement would be:

```
1000 INPUT 2000,3000,var1,var2,var3
```

and would have associated with it an IMAGE statement, thus:

```
2000 IMAGE "NAME? ;I ORDER IS;V##OK?;I"
```

The ";I" denotes input to var1, with conversion performed depending on the type of the variable; the ";V##" denotes output var2 (performing necessary conversions) and truncate to 2 digits.

A number of verbs were "borrowed" from BASIC +2 and have the same meanings:

```
CALL      SLEEP
CHAIN     SUB
COMMON    SUBEND
ONERROR   WAIT
RESUME
```

Others were also borrowed from BASIC +2 and have the same general meanings but the syntax is different, because of the differences between our file system and RMS:

```
CLOSE     KILL
DELETE    OPEN
FIND      PUT
GET       REDEFINE
```

A number of additions to the language are probably not of general interest; they involve resource-allocation verbs (used to prevent deadlock, etc.) and other concepts specific to our operating system or else to our application. Three statements of possible interest are DATA, ARRAY, and STRUCTURE/ENDSTRUCTURE. For each of the data types there are statements (e.g., "DATAI1" for single-precision integers) that allow initialization of variables--this is more like the FORTRAN concept of DATA than the BASIC notion (interpreter based) of READing DATA into the variables. An ARRAY statement can point an "array-definition block" at preset data in order to have a preset array. The STRUCTURE construct allows the definition of a multi-dimensional entity in which each block may have subfields of different data types. The subfields are referred to by their names and by the subscript of the parent structure--rather like PL/1.

Deficiencies

A number of restrictions have already been referred to; the most general restriction is that we have limited the use of expressions to the LET, FOR/TO/STEP, IF/THEN, and ONGOTO/GOSUB statements. You will see from the way expression analysis is implemented that this restriction is arbitrary, and if public clamor at DAMON demanded the use of expressions in, say, PRINT statements, it could be done. We felt that if the number of places where expressions were usable were restricted, perhaps this would limit the number of expressions used, although it could be that the only implication is that the programmer is required to use more LET statements. Unnecessary expressions are discouraged, because they are evaluated by a large recursive macro with a substructure of macros that it invokes, which makes MACRO-11 grind rather slowly, causing tedium at the

terminal. Another restriction is that array and structure indices must be single-precision integers; this derives from the more general restriction that all type conversions must be made explicit (through the use of functions) and mixed-mode expressions are not allowed. Finally, all variables have to be declared (because of the different types).

Functions Implemented

The functions we implemented are mostly self-explanatory, and there should be no fear that I will explain the ones that are not. Below is a list of functions:

ABS	DATE	LOC	RIGHT$	TIME
ASCII	DATE	LOG	RND	VALI1
ATN	DATE	LOG10	SGN	VALI2
BITT	EXP	MID$	SIN	OCT
CHECK	FIX	MOD	SID	
CHR$	INT	NUM1O	SQR	
COS	JULIN	NUM8$	STRING$	
DATE	LEFT$	PI	TAN	
DATE	LEN	POS	TIME	

IMPLEMENTATION

TECO Pre-pass

The "compilation" process is in several stages:

1. A TECO Pre-pass converts the source program to text more intelligible to MACRO-11.
2. MACRO-11 assembles the program, pulling the relevant BASIC verbs as macros out of a macro library.
3. The object program is Task-Built, along with a main program called "IMAGE."
4. IMAGE is run, and it takes the core image of the BASIC program--and writes it out on a disc file intelligible to our operating system (but not RSX-11M).

```
1000    NAME    PN0000          ; PROGRAM NAME
1010    INTEGER I,J,K,(M(3,5))  ; SOME VARIABLES
1020    DOUBLE  (X(2,2))        ; DOUBLE PRECISION ARRAY
1030    STRING  (ST00,10)       ; 10 BYTE STRING
1050    INPUT   3000,1060,ST00,K ;INPUT 2 VARIABLES
1060    IMAGE   "NAME: ;I  AGE: ;I03\;E" ; IMAGE STATEMENT
2000    LET I= (J*(I+K))-((1/J)-K) ; EXPRESSION
2040    FOR I = J-K TO J-(M(#2,#4)) STEP K/J ; LOOP
2050    IF (I<=J) AND ((J-#1)>5) THEN LET J = K+J ; CONDITIONAL
2070    NEXT I                  ; END LOOP
2080    ON J+(I/#2) GOTO (2000,3000) ; COMPUTED GOTO
3000    CHAIN   ST00,#1         ; NEXT PROGRAM SEGMENT
4000    END
```

Figure 1: A Sample BASIC Program

The purpose of the TECO pre-pass is to perform a series of substitutions. Consider Figure 1--a BASIC program. There are line numbers, parentheses, and other items that MACRO-11 would not recognize gladly; there are also items "+" and "-", for example, which MACRO-11 might recognize too well. (We want our macro to analyze expressions, not MACRO-11.) The magic of TECO produces Figure 2.

Note that the line numbers are now of the form "L.NNNN:" and are therefore legal

```
        .MCALL  NAME
L.1000::        NAME    PN0000
L.1010::        INTEGER I,J,K, < M < 5,5 > >
L.1020::        DOUBLE  < X < 2,2 > >
L.1030::        STRING  < ST00,10 >
L.1050::        INPUT   3000,1060,ST00,K
L.1060::        IMAGE   <"NAME: ;I  AGE: ;I03\;E">
L.2000::        LET I EQ$  << J M$ < I A$ K > > S$ < < 1 D$ J > S
L.2040::        FOR I EQ$  <J S$ K > TO < J S$ < M < #2,#4 > > >
L.2050::        IF < < I LE$ J > ND$ < < J S$ #1 > GT$ 5 > > THEN
L.2070::        NEXT I
L.2080::        ON < J A$ < I D$ #2 > > GOTO < 2000,3000 >
L.3000::        CHAIN   ST00,#1
L.4000::        .END
```

Figure 2: Program text after TECO Pre-pass.

MACRO-11 labels. The "+" sign has been transformed to "A$". In general, symbols we generate here and in the macro expansions will have a "." or a "$" in them somewhere, so as not to conflict with user-defined symbols. Spaces have been inserted to separate text, so that the macros will consider the symbols separate arguments. Note that the "END" statement has been replaced with ".END". Guess why! All parentheses have been replaced with angle brackets, which MACRO-11 understands. Also ".MCALL NAME" has appeared at the beginning.

Macro Library

A NAME verb begins each program, which names the program (which is used by IMAGE) and ".MCALL's" all of the macros used to process the BASIC statements. There is a macro called "..EVAL", which evaluates expressions, and another, ".OPRAT", which generates code for expressions. All of these macros can and many do have sub-macros that they call. Many of these macros merely push some arguments and trap to the operating system. Considering the FOR and NEXT macros should be instructive.

We set some flags and evaluate the expression for the starting value A2 (note that TECO surrounded the expression with angle brackets). ..EVAL sets a flag when an expression is a simple variable; in that case we will set the index equal to that variable. Otherwise, we are assured that code has been generated for the expression and that the value now resides in temporary location ".0", in which case we will set the index equal to .0 and pop the temporary variable stack (more about stacks later). We do a similar thing for the end expression except we use a sub-macro (.FML) to store the final valve at an offset from a label we are going to generate based on the value of the symbol ..CTR. We do the same if there is a STEP, otherwise storing a default value of one. We then branch over the code that detects when the loop is finished, thus always executing it once.

Within this code we generate a label with .FNLAB, increment the index, compare, store, and do a RETURN (RTS) if we're finished. Otherwise we pop the run-time stack and do it again. To see how all this works we need to consider the MACRO-time stacks.

```
;FOR    ;FOR STATEMENT
        .MACRO  FOR A1 EQUL A2 TO A3 STEP A4
        ..TYPE=$'A1
        ..ATOM=0
        ..LCTR=0
        ..SPF.=0
        ..SPV.=0
        ..EVAL A2
        .IF EQ,..ATOM-1
        MOV A2,A1
        .IFF
        ..SPV.=..SPV.-1
        MOV .0,A1
        .ENDC
        ..ATOM=0
        ..LCTR=0
        ..EVAL A3               ;
        .IF EQ,..ATOM-1         ;NEXT   ;USED WITH FOR LOOPS
        .FML \..CTR,A3,4        ;
        .IFF                            .MACRO  NEXT,A1
        ..SPV.=..SPV.-1                 ..LEV=..LEV-1
        .FML \..CTR,.0,4                ..DSTK  \..LEV
        .ENDC                           .ENDM
        ..ATOM=0                ;..DSTK !
        ..LCTR=0                        .MACRO ..DSTK A
        .IF B STEP                      ..GOF. \.ST'A
        .FML \..CTR,#1,2                .ENDM
        .IFF                    ;..GOF. !
        ..EVAL A4                       .MACRO ..GOF. B
        .IF EQ,..ATOM-1                 JSR     PC,$'B
        .FML \..CTR,A4,2                .ENDM
        .IFF
        ..SPV.=..SPV.-1
        .FML \..CTR,.0,2
        .ENDC
        .ENDC
        BR .+30.
        .WORD 0                         ;FINAL VALUE
        .WORD 0                         ;STEP SIZE
        .FNLAB \..CTR,\..LEV,A1         ; ADD STEP
        ..CTR=..CTR+1
        ..LEV=..LEV+1
        ADD     A1,R2
        CMP R2,-12.
        BLE .+4                         ;DO AGAIN
        RETURN
        ADD #2,SP                       ;ADJUST STACK
        MOV     R2,A1
        .ENDM
;
;.FNLAB ! USED FOR GENERATING LABELS FOR LOOPS
;
        .MACRO  .FNLAB A,B,C
        .ST'B=..CTR             ; PUT NUMBER OF LABEL A1
$'A:    MOV .-2,R2      ; AT CURRENT STACK LEVEL
        .ENDM
;
;.FML - STORE FINALVALUE AND STEP SIZE
;
        .MACRO .FML A,S,OFF
        MOV S,$'A-OFF
        .ENDM
```

Figure 3: FOR/NEXT Statement MACROS

Stack Implementation

The implementation of stacks (and label generating) used the "\" feature of MACRO-11. This feature permits passing "\SYMBOL" as an argument to a macro, and at that macro's level the argument is a numeric string corresponding to the value of SYMBOL at the level that called the macro. That is--by the way--the reason for some of the seemingly unnecessary submacro-izing in the library.

Consider the problem above: we are generating unique labels for the NEXT statement to JSR to (that is how we can come back exhausted following the correct NEXT), and what we would like to do is push each label onto a stack for each FOR statement and pop it off a stack at each NEXT statement, in order to maintain proper FOR-NEXT nesting.

In the above example ..CTR is used for generating unique labels, whereas ..LEV counts the level of nesting of FOR-NEXT loops and serves as a "stack pointer" (both symbols are initialized to 0 in the NAME macro). In the .FNLAB macro a symbol is formed from the concatenation of ".ST" and the string formed from the value of ..LEV; this symbol constitutes the stack "storage element" and proceeds from ".ST0" to ".ST999" as the level of nesting increases. The symbol is assigned the value of the current label-generation counter. A label is affixed to the "MOV" instruction, which is the concatenation of "$" with the string corresponding to the value of the label-generation counter. In the FOR macro on return we increment ..CTR (the label-generation counter) forever and ..LEV (the stack pointer) for the time being.

When we encounter a NEXT statement, ..LEV is decremented by one (in effect popping the stack), so that it now corresponds to the latest stack element. We go down one macro level via the ..DSTK macro to form the string corresponding to the current level; then down one more macro level via ..GOF. to convert the value of ".STN" to a string to form the proper "$N" label to JSR to. Whshew! So that is how numbers are stored on a stack, and that is what makes FOR-NEXT work, what facilitates temporary label generation (these labels get recycled), and how still another stack used in the expression analyzer works.

Data Definition and PSECT Structure

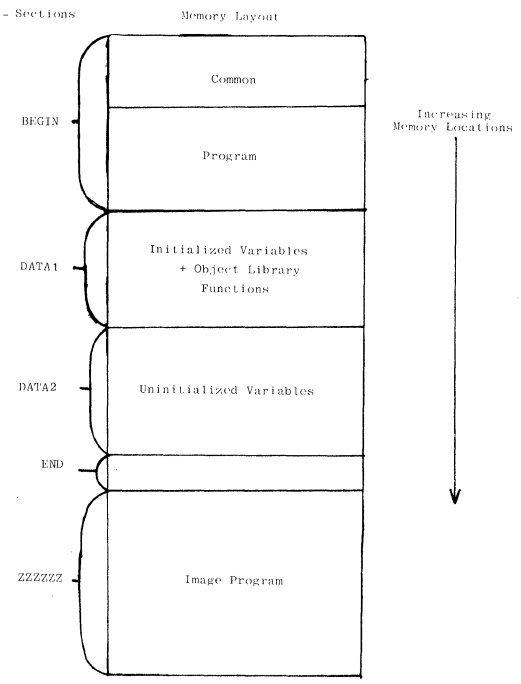


Figure 4: Memory layout of task

At Task-Build time there are five relevant P-sections and, as you can see from Figure 4, they are named "BEGIN", "DATA1", DATA2",

"END", and "ZZZZZZ". BEGIN contains the COMMON area and object code of the BASIC program. DATA1 contains the initialized variables from DATA-type statements and array-definition blocks; it also contains any object-library programs that may be required. DATA2 contains any uninitialized variables that don't live in COMMON, and END contains information for the IMAGE program, which lives in ZZZZZZ. The only part of the program that IMAGE saves on disc is the program part of BEGIN and all of DATA1.

The data-definition statements have several purposes.

1. To define the variable
2. To define it in the appropriate P-section
3. To make known the type of the variable
4. To make known the length of the variable in the case of strings
5. To set up an array-definition block in the case of arrays and structures.

The type of a variable is determined by defining a symbol, "$variablename", and assigning it a value. This effectively limits variable names to five characters; string lengths are assigned to symbols of the form ".variablename". Array-definition blocks have the form:

```
variablename...WORD    variablename
               .WORD   (definition)
               .WORD   ext1
               .WORD   ext2
                 ...
               .WORD   extN
```

Where (definition) is a word containing the number of dimensions in the low-order byte and the number of bytes per item in the high-order byte. The exti's give the extent in each dimension. A restricting implication is that all data must be defined before use.

Expression Analysis

In analyzing expressions, since everything happens in the first pass of the assembler, we did not have the compiler luxury of pushing everything onto stacks in Reverse Polish Notation and then popping the operands and variables off to generate code. We also felt that trying to implement implied precedence would be difficult and that forcing the use of parentheses would not hurt and would in fact even help program readability.

The macro that (with submacros) evaluates expressions is called ..EVAL and is diagrammed in Figure 5.

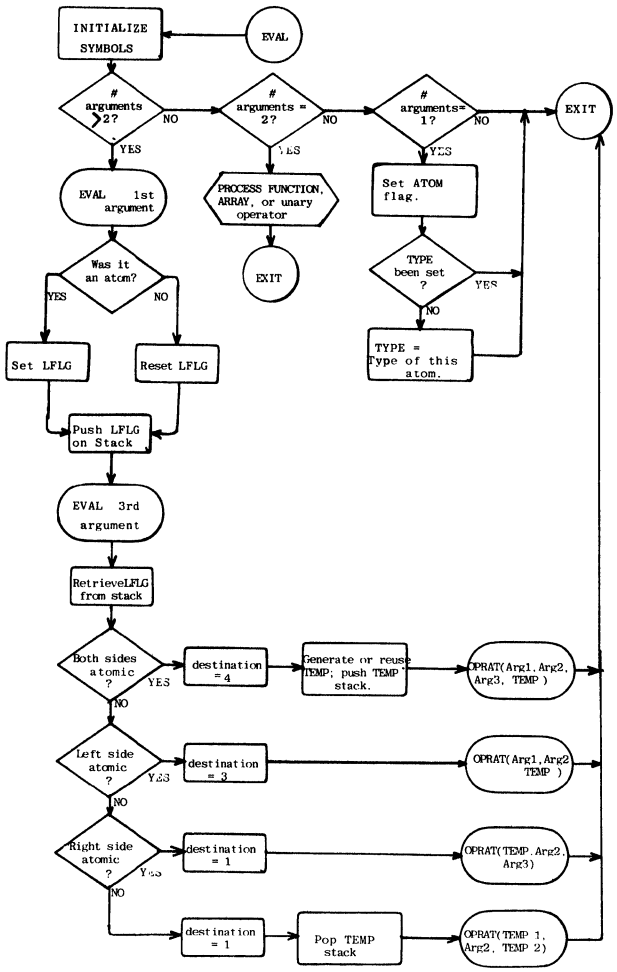


Figure 5: Flowchart of Expression Evaluator

..EVAL is recursive, and because each level of recursion removes a pair of angle brackets--a property of MACRO-11--..EVAL can be invoked until a level is reached where a single symbol is its argument. The macro takes three courses of action, depending on whether it has one, two, or many arguments (zero arguments implies a syntax error). If it has two arguments it decides it has a function reference, an array reference, or something of the form "unaryoperator (expression)". The two-argument case will be discussed later. If it has more than three arguments it decides it has something of the form:

"(expression) binary-operator (expression)...

Let us consider the case of three arguments, that is, "(exp1) op (exp2)". We first apply ..EVAL to (exp2); whenever we return to this macro-nesting level we are assured that all of the code necessary to compute exp1 has been generated or else none has because exp1 is atomic. Since the lower level of ..EVAL sets a flag indicating whether this is the case, we can set or reset a flag--..LFLG--,

accordingly (the ..ATOM flag is reset before each call of ..EVAL). The value of this flag is pushed onto a stack (this is the third one mentioned earlier). We now apply ..EVAL to (exp2), after which we pop ..LFLG from the stack. In each case that the expression were complex, the results of the computation were stored in a temporary location, whose number is on the temporary-location stack (these locations are generated as .0,.1,.2, etc. or for strings, ..0,..1,..2, etc.). We now have exactly four cases:

Case 1: exp1 and exp2 both atomic
In this case we generate (or recycle) a temporary location using .LBGEN and increment the stack pointer. .LBGEN calls .OPRAT to generate code to perform the indicated operation on exp1 and exp2, and we store the result in the temporary location (using the macro call ".OPRAT exp1, OP, exp2, temp", with the destination =4 indicating that the result should be stored in the fourth argument). The submacro .LBGEN works by looking at the temporary variable stack pointer, and assembling the appropriate label--".N". It then defines the location (if it is undefined) and makes the call to .OPRAT using the temporary label as an argument.

Case 2: exp1 atomic, exp2 not atomic
In this case we already have a temporary location, so we can store the result in it. We thus perform the operation on exp1 and temp and leave the result in temp (".OPRAT exp1, OP, temp", with the destination=3).

Case 3: exp1 not atomic, exp2 atomic
In this case we store the result in the temporary variable containing the result of the evaluation of exp1, (".OPRAT temp, OP, exp2", with destination =3).

Case 4: both exp1 and exp2 complex
In this case there are the distinct temporary variables temp1 and temp2 containing the results of computing exp1 and exp2. We decrement the temporary label stack pointer and store the result in the left-hand (lower stack count) temp, (".OPRAT temp1, OP, temp2", with destination =1).

It now can be seen that the case of many arguments of ..EVAL can be resolved pairwise, since after the first pair we always have a temporary result on the left and need only to ..EVAL the exp on the right to discover whether we have Case 2 or Case 4. The macro .NEXPR does this after the first pair (where "pair" means "exp OP exp") and continues until it gets blank arguments. It should be noticed that at return from the highest level of ..EVAL the final result is stored in the lowest temporary label of the stack--".0 or ..0".

..EVAL with two Arguments and Function Implementation

When ..EVAL has two arguments, it first determines whether the first argument is a variable (by virtue of its type being defined). All of the operators as well as all of the traps are defined in different numeric ranges by the NAME macro at the beginning. We can then detect unary operators and distinguish them from traps (which within expressions sometimes implement functions).

If we have a unary operator, we apply ..EVAL and determine whether it is an atom: If it is it is a special case of CASE 1, with only one atom; if it is not, it is a special case of Case 3. In either case .OPRAT can deal with it.

If the first argument is not a variable and is defined, it is a function implemented as a system trap; if it is not defined, it is an object-library function. For both types of functions a sub-macro (..ARPU) breaks out the separate arguments and generates code which will push them on the stack at run-time with specific symbols predefined for each function determining:

1. Whether for each argument to have its address or value pushed.
2. Whether first to push a return-argument address in the appropriate position.
3. Whether to push the length of a string before a string argument.

Then, depending on whether it is a trap function or in the object library, a trap function will then generate a trap, and an object-library function a JSR (to an undefined a label until task-build time). The decision as to whether a function is a trap or an object-library routine depends on the judgment of whether it is so commonly used by all tasks that it should remain core-resident, or whether it is less frequently used or takes little space anyway and can be added to each task.

At any rate, if it is none of the above it is an array reference--the same routine as above generates code to push the arguments but doesn't have any special symbols defined to make it do anything. It merely pushes the arguments, which are either integer variables or literals. Then a "JSR..ARDE" is generated, which calls the subroutine that computes from the indices and the array-definition definition block the address of the element. Note that this is in the object library not because it is infrequently used, but because it is short and we can avoid the trap-handling process to make it faster.

Code Generation

By this point code generation becomes reasonably trivial. The code generator (.OPRAT) is called upon to perform a binary (occasionally unary) operation. It is given (4 arguments - A1, OP(,A2(,A4)). and a "destination" value, and it is expected to store the result in A1, A2, or A4, depending on that value. What code is generated depends first on the operator and second on the data type (operations that generate logical results such as ">" result in integers =0 or not =0, and the AND, OR, NOT, etc., operators deal in these entities). The operators are logical, relational, and arithmetic. A flow chart of .OPRAT is presented in Figure 6, and an example of part of the macro (featuring the integer arithmetic) in Figure 7.

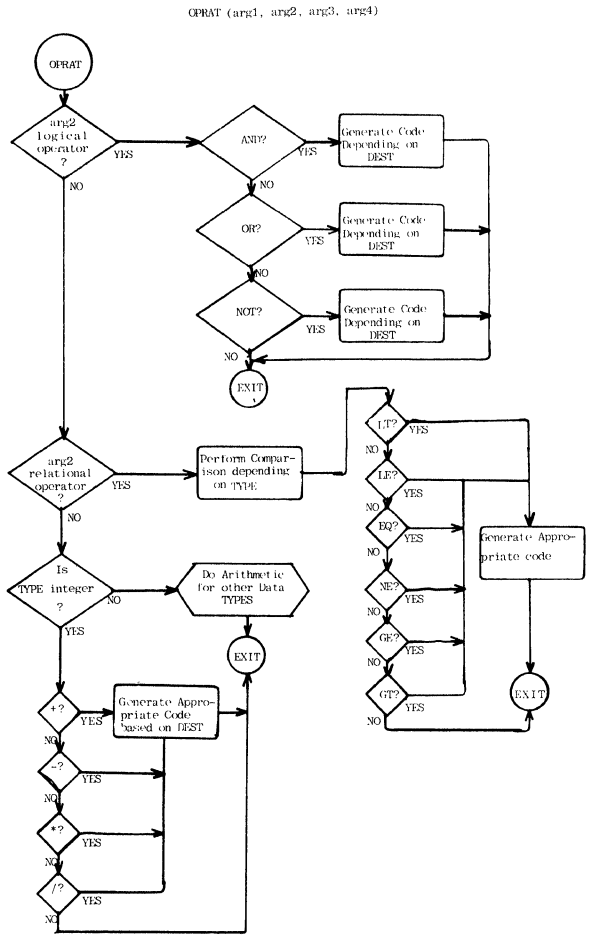


Figure 6: Flowchart of Code Generator

```
1797 ;
1798 ;       ARITHMETIC
1799 ;
1800         .IF     EQ ..TYPE-..INTY                ; INTEGER
1801                 .IF EQ B-A$             ; ADD
1802                 .IIF EQ ..DEST-1        ADD C,A
1803                 .IIF EQ ..DEST-3        ADD A,C
1804                 .IF EQ  ..DEST-4
1805                 MOV A,D
1806                 ADD C,D
1807                 .ENDC
1808                 .MEXIT
1809                 .ENDC
1810                 .IF EQ B-S$
1811                 .IIF EQ ..DEST-1        SUB C,A
1812                 .IF EQ ..DEST-3
1813                 MOV A,-(SP)
1814                 SUB C,(SP)
1815                 MOV (SP)+,C
1816                 .ENDC
1817                 .IF EQ  ..DEST-4
1818                 MOV A,D
1819                 SUB C,D
1820                 .ENDC
1821                 .MEXIT
1822                 .ENDC
1823                 .IF EQ B-M$
1824                 MOV C,R5
1825                 MUL A,R5
1826                 .IIF EQ ..DEST-1 MOV R5,A
1827                 .IIF EQ ..DEST-3 MOV R5,C
1828                 .IIF EQ ..DEST-4 MOV R5,D
1829                 .MEXIT
1830                 .ENDC
1831                 .IF EQ B-D$
1832                 MOV     A,R2
1833                 ASHC #-16.,R2
1834                 DIV C,R2
1835                 .IIF EQ ..DEST-1        MOV R2,A
1836                 .IIF EQ ..DEST-3        MOV R2,C
1837                 .IIF EQ ..DEST-4        MOV R2,D
1838                 .MEXIT
1839                 .ENDC
1840         .ENDC
```

Figure 7: Integer Arithmetic Code Generator

Finally, here is some of the code generated from our original example.

```
00  L.1010::        INTEGER I,J,K, < M < 3,5 > >
00  M.:     .WORD   M
02          .WORD   2*256.+..NDEM-2
04          .WORD   3
06          .WORD   5
00  L.1020::        DOUBLE  < X < 2,2 > >
10  X.:     .WORD   X
12          .WORD   4*256.+..NDEM-2
14          .WORD   2
16          .WORD   2
00  L.1030::        STRING  < ST00,10 >
00  L.1050::        INPUT   3000,1060,ST00,K
00          MOV #OUTCHAN,-(SP)
04          MOV #L.3000,-(SP)
10          MOV     #L.1060+2,-(SP)
14          MOV     #$0$,-(SP)
20          .WORD   ST00
22          .WORD $ST00*256.+.ST00
24          .WORD   K
26          .WORD $K*256.+.K
30          .WORD 0
20          TRAP    INPUT
22  L.1060::        IMAGE   <"NAME: ;1  AGE: ;I03\;E">
22          BR      64$
24          .ASCII "NAME: ;1  AGE: ;I03\;E"
52  L.2000::        LET I EQ$  << J M$ < I A$ K > > S$ < < I D$ J
52                  MOV I,.0
60                  ADD K,.0
66                  MOV .0,R3
72                  MUL J,R3
76                  .IIF EQ ..DEST-3 MOV R3,.0
02                  MOV     1,R2
06                  ASHC #-16.,R2
12                  DIV J,R2
16                  .IIF EQ ..DEST-4        MOV R2,.1
22                  .IIF EQ ..DEST-1        SUB K,.1
30                  .IIF EQ ..DEST-1        SUB .1,.0
36          .IIF EQ ..MATO MOV .0,I
44  L.2040::        FOR I EQ$  <J S$ K > TO < J S$ < M < #2,#4 > :
44                  MOV J,.0
52                  SUB K,.0
60          MOV .0,I
66          MOV #2,-(SP)
72          MOV #4,-(SP)
```

Figure 8: Translated version of a portion of sample.

CONCLUSION

The line-number labels and the varible names are defined as global symbols, and there are no other ones except for object-library entry points. This means that a programmer equipped with a BASIC listing and a TKB map can debug at the BASIC level using a program like ODT (which we have implemented). It has been a lot of fun to work on this project and really exercise the macro-assembler (since it is recursive, it should be able to solve all computable problems). Those who would cavil at its nonstandardness should remember that it was done for a specific purpose, and no one is trying to promulgate it. The effort required a couple of person-months to design and code; fewer to debug. The indication is that if you want to implement a language in a hurry (or perhaps want to translate programs to a PDP-11 that does not support the language in which they were written), you might consider the capabilities of MACRO-11.

ACKNOWLEDGMENTS

Dirk Brinkman and Charles Kern both provided a large number of the ideas used in this project. Kern did a lot of the work on the macro library and completely designed and implemented the TECO pre-pass. Hilary Horton lent more editing skills than such a paper deserves, and Rita Frazier much good typing and patient retyping. Pam Meyer of Damon's Corporate Communications department did a professional job on the illustrations. Finally, thanks to everybody at Labshare who attempted to write BASIC programs and thereby discovered the many bugs.

CONVERSION OF VERY LARGE PROGRAMS TO RSX-11 BASED SYSTEMS

Steven R. Deller
Computer Sciences Corporation
Falls Church, Virginia

ABSTRACT

Techniques and supporting tools for converting very large programs to RSX-11D or RSX-11M systems are described. The methodology is illustrated with examples of FORTRAN program conversions, but most of the techniques can be applied to programs written in any language.

Systematic solutions are provided which minimize code changes to maximize traceability to the original code. Potential pitfalls in each area are identified. Where feasible, existing DEC system software was used for solving the problems and verifying the conversion. Where needed, analytic tools were developed to support the conversion.

This methodology was applied to several real-world problems; a description of one of these conversions is provided. DEC system limitations that have prevented straightforward conversion are identified, and simple corrections are proposed.

INTRODUCTION

A systematic method of converting very large programs to RSX-11D or RSX-11M operating systems is described. The method consists of automated and manual techniques developed and used for the conversion of large FORTRAN programs from a CDC 6600 computer to a PDP-11/70. The programs consisted of many routines, including very large single routines. In most cases, the size of the programs and the sequence of calls exhausted the capabilities of the overlay system. Automated techniques were developed to support separation of program routines into multiple, cooperating tasks which operate as though the routines are in a single task. The separation is effected without alteration of the source code so that traceability to the original program is maximized and testing is simplified. While the tools presented are discussed in terms of the FORTRAN language and the RSX-11D system, the method applies to most languages and systems. Automated tools described are directly applicable to the RSX-11M system.

Terminology

The terminology used to describe software must be specific for clear understanding of certain concepts in this paper. We have selected the following terms and definitions as being the closest to most literature and therefore, hopefully the easiest to remember.

The term "routine" refers to a separately compilable unit of software. The term "task" refers to a collection of software routines which form the executable unit of a system. The term "program" refers to a collection of software routines which execute together to accomplish a single function.

The distinction between program and task is important to this paper. We describe later a method by which a single large program is divided into a set of cooperating tasks. Thus, a set of tasks comprises a program, contrary to usual programming practices in which a task and program are one and the same. A routine is further classified as a main routine (introduced in FORTRAN by the PROGRAM statement) or a subroutine (introduced by a SUBROUTINE or FUNCTION statement).

The abbreviation SGA stands for "sharable global area," a DEC term describing an area that exists within the address space of more than one task and contains code, data, or both. Thus, an SGA is a communication area for tasks in the same way that COMMON is a communication area for routines. The single difference is that an SGA may contain executable code while COMMON may not.

The abbreviation K stands for the decimal value 1024, the "binary Kilo" value. Thus 3K words refers to 3072 words.

Specific dialects of FORTRAN and versions of operating systems are discussed in this paper. Dialects accepted by three PDP-11 FORTRAN compilers are discussed:

- FTN refers to the old V08.05 FORTRAN compiler available under RSX-11D V6A.
- FOR refers to the V02.04 FORTRAN IV compiler available under RSX-11M V3.1 and IAS V2.
- F4P refers to the V02-51 FORTRAN IV-PLUS optimizing compiler available under RSX-11D, RSX-11M, and IAS.

Descriptions of FORTRAN which are operating system dependent refer to the F4P compiler running under RSX-11D V6.2 or RSX-11M V3.1. CDC FORTRAN refers to the Control Data Corporation FORTRAN V2.3 compiler running under the SCOPE V3 operating system. Versions of other utility programs are not specified since they are not germane to issues in this paper.

Conversion Considerations

The decision to convert large programs to the PDP-11 computer should be very carefully considered. Regardless of the methods used, most of the effort required to convert a very large program would not be required if the target machine had a larger address space. Use of either a DECsystem-10 or VAX-11/780 computer would simplify the conversion effort. Only translation of the source program into a FORTRAN dialect accepted by the target computer would be necessary. Nonetheless, existing equipment or software requirements may necessitate conversion to the PDP-11. The work in this paper is presented in the hope that such activity, when required, will be as easy as possible.

CONVERSION METHODOLOGY

The conversion effort is separated into five sequential activities:

1. Conversion to PDP-11 FORTRAN standard
2. Source modifications for very large data areas
3. Fragmentation of very large single routines
4. Assignment of routines to separate but co-operating tasks
5. Construction of task overlay structures to minimize memory requirements.

Detailed descriptions of each activity follow. The descriptions include procedure examples and discussion of automated tools developed to support the conversion.

Language Conversion

The conversion from one FORTRAN dialect to another involves preparation of a syntactically and semantically correct program, which may be compiled on the PDP-11. Usually the I4 compilation switch is used to default to 4-byte integers so that integer and real alignments are preserved and integer values have a sufficient range. If the source documentation permits, some consideration may be given at this point to the adequacy of data item sizes and floating point accuracy.

The conversion activity involves three separate steps:

1. Determination of language differences
2. Source conversion
3. Verification.

Each step is described in detail in the following paragraphs.

Determination of Language Differences. A majority of language differences can be identified by compiling on the PDP-11 and examining the source lines that produced compilation errors. Care must be taken with this approach, however, since a language construction may have identical syntax in both languages, yet have a different semantic interpretation.

For conversion from CDC 6600 FORTRAN, we encountered four frequently occurring syntactic constructs that had to be converted to PDP-11 FORTRAN syntax:

1. Two-way branching logical IF of the form:
   IF (logical) truelabel, falselabel
   converted to the form:
   IF (logical) GOTO truelabel
   GOTO falselabel

2. Multiple assignment destinations of the form:
   variable1 = . . . = variableN = expression
   converted to the form:
   variable1 = expression
   ⋮
   variableN = expression

3. Very large logical unit numbers converted to small, sequentially assigned numbers.

4. Seven-letter symbols converted to six-letter symbols.

In addition to these, many individual syntactic constructs required conversion. For example, the CDC FORTRAN allowed ENTRY statements to be expressed without repetition of the subroutine parameters, and compilation control expressions to be included in a PROGRAM statement. For F4P the subroutine parameters had to be repeated in the ENTRY statement, and the compilation control expressions removed from the PROGRAM statement.

Semantic differences are much harder to identify. Constructions unspecified by the ANSI standard, which require interpretation by compiler writers, are especially likely to be a source of semantic interpretation errors. For example, the definition of internal representations for .TRUE. and .FALSE. logical values frequently differ among dialects. Consequently, programs using integers instead of logicals in expressions (relying on type conversion by the compiler) may operate differently when compiled by different compilers.

Even dialects that assign the same values to logical constants may differ in the way logical values are tested. Both the CDC and PDP-11 FORTRAN languages use the usual definitions of all ones (-1, twos complement) for .TRUE. constants and zero for .FALSE. constants. However, both the CDC and DEC FTN compilers produce code that evaluates true for any nonzero value in an expression result, the DEC FOR compiler evaluates true for a nonzero lower byte, and the DEC F4P compiler evaluates true for a negative value (nonzero sign bit) in an expression result. In some of the code we converted, integers with values other than -1 or 0 were tested as logicals. For CDC execution they evaluated true, for F4P they evaluated false if positive, true if negative. Each occurrence was manually detected, usually during testing, and fixed.

Many other semantic conversion problems were encountered. The operation of a computed GOTO when the index is out of range was different between compilers. Code that directly modified real values through manipulation of equivalenced integer values had to be changed because internal representations differed between machines. Because these problems are semantic, not syntactic, simple editing programs could not aid these conversions. However, a program parser, currently under development for support of other automated tools, should allow generation of tools to at least detect these problems, if not automatically correct them.

Source Conversion. Most program units were converted using a combination of TECO editor macro commands and

a file conversion program written in FORTRAN. Text search subroutines previously written for a communication system were used to minimize the program development. The definition and implementation of both conversion aids were completed in a few hours by one person.

Verification. The converted source was verified in two ways. First, a FORTRAN compilation performed after each conversion effort determined whether the program was syntactically acceptable. Second, after the program had been successfully compiled, a file comparison was run (using the CMP utility) to produce a listing of differences between the converted source and the original source program. Another person then checked this file difference program manually to verify that each conversion was correct.

Although we were initially skeptical of the value of this last manual verification, it proved to be beneficial in all conversions performed. For each conversion, at least one conversion error was identified. In one of the earlier conversions, an error in the conversion definitions was identified. The conversion

```
A = B = 0        to      A = 0
                         B = 0
```

was performed by a TECO macro command. The particular implementation chosen produced the following incorrect conversion.

```
IF (L) A = B = 0    to    IF (L) A = 0
                          B = 0
```

Such an error in the conversion could prove to be extremely difficult to detect through debugging.

Large Data Areas

The conversion of large data areas in programs from large computers is one of the most difficult problems faced in a conversion task. The method used for conversion depends on whether the large data area is a single very large array or simply a very large conglomeration of very small data items.

Large Arrays. For single very large arrays, two different solutions are possible. First, the VIRTUAL array statement may be used with the RSX-11M FOR compiler. This solution is the most desirable, since it represents minimal source code modifications and adds very little overhead to array accessing. This facility, not yet available with F4P, has been promised with the next release. However, it will not be available for RSX-11D unless region definition and manipulation directives are added to the system, which seems unlikely.

An alternative solution to virtual arrays is to define a function subroutine for the array reference. The source program is modified by removing all declarations for the array and defining the array name as an external routine. The function subroutine allows access to a very large array stored either as a disk file or as a logical memory array using the buffer allocation and remapping functions of RSX-11M.

Because access to the backing store (either disk or logical memory area) is relatively slow, the function subroutine can provide array transfers in blocks so that most array references are handled directly from the buffer. If the routine is written in assembly language, average item access should not be much greater than that of a normal FORTRAN array. A compromise between readibility and efficiency could be provided by making primary access an assembly language program with a call to a FORTRAN program for any backing store references.

Large Data Groups. Handling large groups of small data items presents a different problem. Several of the programs converted in our effort involved unnamed COMMON areas of more than 20K words, leaving insufficient space for FORTRAN applications and support routines. To provide more space, it was necessary to separate this COMMON area into several named COMMON areas based on program usage. It was then possible to place programs using different COMMONs into separate tasks so that the data area requirement for any one task was less than 20K words. Use of the TECO editor allowed searching for symbol references so that program COMMON usage could be determined. Frequently, we found a relatively large portion of COMMON used by only a small set of related subroutines. When the entire program was later split into separate tasks, that COMMON needed definition only within the task containing those subroutines.

In a few cases, single very large routines consisting of more than 1,000 lines of noncommentary source statements contained local and common data references to very large data areas. Conversion of these routines required separation into smaller units, each of which referenced a much smaller data area. Such cases presented an extremely difficult task. These efforts—conversion of very large data areas included as parts of very large programs—presented the only problem that we felt might make conversion technically impossible. Some of the programs that we converted included some difficult problems in this area, but all were successfully converted—including one that contained more than 32K words of COMMON data area.

Large Routine Segmentation

Some of the programs converted consisted of more than 1,000 noncommentary source statements which, when compiled, required more than 6K words of memory. When combined with 20K words of data area, 3K words of I/O routines, 3K words of system support, and 2K words of arithmetic support, the allowable task size of 32K words was exceeded.

Even in programs with smaller data area requirements, programs with more than 500 source statements complicated later efforts in construction of separate tasks and overlays. Consequently, an attempt was made to segment all large routines into several smaller routines, each of which had less than 500 statements. This segmentation was accomplished through a systematic manual application of four separate steps. We are currently investigating the possibility of automating some or all of these steps. However all the steps in routine segmentation are relatively easy to apply manually, so automation will probably not prove to be cost-beneficial.

In order to illustrate the segmentation procedure, a small nonsense program, shown in Figure 1, was split into two segments. Production of two code segments from this single routine is described in the following paragraphs. It should be evident that the segmentation procedure is meant to be applied only to large routines, in spite of the small size of the example.

Source Code Segmentation. Segmentation begins with determination of program segmentation points. This determination attempts to define points that have minimum control and data transfer requirements; that is, points for which there are the fewest transfers into

```
0001                SUBROUTINE S(A,K)
0002                INTEGER*4 K,J
0003                I = K/2
0004                J = K/3
0005      10        IF (I.EQ.J) GO TO 20
0006                K = J
          COMMENT   **** SEGMENT POINT
0007                I = I + 1
0008                GO TO 10
0009      20        A = 0.3 + FLOAT(I)
0010                RETURN
0011                END
```

Figure 1. Sample Program for Segmentation

and out of any segment, and for which the fewest variables are shared between segments. This is the only step in the segmentation procedure which is reasonably difficult, and thus might benefit from development of an automated tool. A tool, based on a semantic parser, could be developed to list the number of control and data transfers resulting from segmentation at each source line.

Using an editor, the executable statements comprising each of the identified segments are separated from the original code. For each segment, a SUBROUTINE statement similar to the original statement, but with the segment name instead, is placed at the beginning. RETURN and END statements are appended to the end of the segment. Figure 2 shows the resulting code for subroutines S1 and S2 defined as segments of the original routine shown in Figure 1.

```
0001                SUBROUTINE S1(A,K)
0002                I = K/2
0003                J = K/3
0004      10        IF (I.EQ.J) GO TO 20
0005                K = J
0006                RETURN
0007                END

ERROR  50  SYMBOL:  20
F UNDEFINED STATEMENT LABEL
```

```
0001                SUBROUTINE S2(A,K)
0002                I = I + 1
0003                GO TO 10
0004      20        A = 0.3 + FLOAT(I)
0005                RETURN
0006                END

ERROR  50  SYMBOL:  10
F UNDEFINED STATEMENT LABEL
```

Figure 2. Sample Intermediate Segment Code

Intersegment Control Communication. Each segment defined in this manner is then compiled. Compilation errors for each segment identify any references to labels outside of that segment. A unique number is generated for each of these intersegment transfer labels. In the example shown, numbers starting with 100 are allocated as subroutine S1, and numbers starting with 200 are allocated as subroutine S2. The first entry point for a segment is defined as the beginning of that segment. Each label branched to within a segment from other segments is then allocated one of the numbers reserved for the segment in which it is defined. Thus, for subroutine S1 in the example, the beginning of the routine is given the unique number 101, and label 10 defined within the routine is given the unique number 102.

A uniquely named control variable (ICTL in the example) is defined for use in resolving internal transfers to external labels. Each external label referenced from a segment is redefined within the segment. Code at the label sets the control variable to the unique number previously assigned to that label, and then performs a RETURN.

To resolve external transfers to internally defined labels, a computed GOTO, indexed by the control variable, is added to the beginning of each subroutine. All labels referenced from other segments are assigned a relative position in the GOTO label list determined by the unique number previously assigned.

Finally the resulting code is compiled. Most errors in the intersegment coding produce compilation errors. For the simple example we have chosen, this compilation is not very informative so it is not shown in a figure.

Intersegment Data Communication. When all label references between segments have been resolved, each segment should compile without error. The resulting compilation listings may then be analyzed to determine all variables referenced in a given segment. At this point, lacking all declarations from the original routine, any array references will show up as functions. All variables referenced within more than a single segment are placed into a new named common (SCTL in the example) along with the control variable. Finally, each segment is edited to add all declaration statements from the original subroutine that are associated with variables used uniquely in that segment or variables defined within the common. The resulting code for subroutines S1 and S2 is shown in Figure 3. Note that the declaration for the variable "J" is not present in subroutine S2, since that variable is unused in that segment.

```
0001                SUBROUTINE S1(A,K)
0002                INTEGER*4 K,J
0003                COMMON /SCTL/I,ICTL
0004                GO TO (1,10), ICTL
0005                CALL ERROR
0006      1         CONTINUE
0007                I = K/2
0008                J = K/3
0009      10        IF (I.EQ.J) GO TO 20
0010                K = J
0011                ICTL = 201
0012                RETURN
0013      20        ICTL = 202
0014                RETURN
0015                END
```

```
0001                SUBROUTINE S2(A,K)
0002                INTEGER*4 K
0003                COMMON /SCTL/I,ICTL
0004                GO TO (1,20),ICTL
0005                CALL ERROR
0006      1         CONTINUE
0007                I = I + 1
0008                GO TO 10
0009      20        A = 0.3 + FLOAT(I)
0010                ICTL = 0
0011                RETURN
0012      10        ICTL = 102
0013                RETURN
0014                END
```

Figure 3. Sample Final Segment Code

Segmentation Control Routine. The final segmentation activity is definition of the control routine that will

mediate intersegment transfers. This routine consists of the original subroutine statements, the named common for intertask communication, and code for calling each of the segments when required. Figure 4 shows the control routine code used to mediate transfers between segments in the example. Only declarations for calling parameters and intersegment common variables are included.

```
0001                SUBROUTINE S(A,K)
0002                INTEGER*4 K
0003                COMMON /SCTL/I,ICTL
0004                ICTL = 101
          COMMENT   **** S1 ENTRY 1 FIRST
0005      999       IF (ICTL.EQ.0) RETURN
0006                JCTL = ICTL/100
0007                ICTL = MOD(ICTL,100)
0008                GO TO (1,2), JCTL
0009                CALL ERROR
0010      1         CALL S1(A,K)
0011                GO TO 999
0012      2         CALL S2(A,K)
0013                GO TO 999
0014                END
```

Figure 4. Sample Intersegment Control Routine

Upon initial entry, the control routine simply enters the first defined position of segment 1. Thereafter, the control routine uses the value in the control variable to determine which next segment to call and what control value to pass. A returned control value of zero signifies completion of the original routine; that is, the control routine should return to its caller.

Although the segmentation procedure appears somewhat complex, our experience indicates that it is relatively straightforward—less than one man-day is required to segment a large program. In some cases, segmentation may be completed in less than an hour.

Task Construction

Determining groups of routines to be developed into tasks and constructing those tasks using the task builder is one of the most difficult steps in a large program conversion. This difficulty results from the inherent complexity of the task building process and the problem of determining a particular organizational structure that simultaneously meets several complex restrictions.

A task build is inherently complex because of the many task structure options that must be determined. Although it is possible to ignore most of these options in normal program task building, it is not possible with task builds developed for large program conversions. The premium placed on task addressing space makes it imperative that all task build allocation and space control options be carefully determined. The potential for wasting large amounts of addressing space because of the large allocation size of PDP-11 page addressing registers makes the specification of SGA contents very important.

Four steps are required to generate individual tasks from the original program routines:

1. Allocation of routines to tasks
2. Preliminary task building
3. SGA task build specification
4. Intertask communication specification.

Application of any step may require several iterations if problems are encountered in some of the later steps. The first step, allocation of program routines into tasks, is driven by three sequentially applied criteria: isolation of all I/O to one task, minimization of SGA size, and minimization of intertask calls.

Because each step is somewhat complex and may be performed several times, it is imperative that a well organized set of compilation, listing, and task building command files be developed. It is especially helpful if the indirect command file processor has been installed for RSX-11D.

Figure 5 illustrates the main indirect command file for creating five cooperating tasks resulting from one of our program conversions. The figure also displays the details of the task build and overlay description files for both the SGA (RAYCOM) and one of the tasks (RAY1). Use of these command files during conversion provides automatic documentation of the conversion effort status at all times.

Allocation of Routines to Tasks by I/O. All routines that perform large amounts of I/O must be grouped into a single task. All I/O requests must come from a single task to avoid file or device usage conflicts and to synchronize reading of sequential file records. Routines containing only a few I/O statements need not be allocated to the I/O task. Instead, only the I/O statements themselves are placed into the task by creating a new subroutine consisting of only those statements.

To create the I/O subroutine, each I/O statement in the original routines is replaced by a call to the I/O subroutine, specifying a single calling argument. The argument contains a unique value for each call. The I/O statements removed in this fashion are each given a unique label and placed into the I/O subroutine, followed by a RETURN statement. The associated format statements are also moved from the original routines into the I/O subroutine. Then code—usually a computed GOTO—is added to the head of the subroutine to branch to the appropriate I/O statement depending on the value of the call parameter.

We developed a relatively simple automatic file-scanning program to perform this task. It allocates both the unique parameter values and the required labels within the I/O subroutine, producing all of the code except the entry and exit code of the I/O subroutine. Because the statements were collected from several other routines, there were occasional format statement labeling conflicts that had to be resolved manually.

Allocation of Routines to Tasks by SGA Usage. After all I/O has been grouped into a single routine, usage of common by routines is examined to determine whether SGA requirements can be reduced by putting all references to a particular common block into a single task. In order to simplify this effort, we wrote a small scanning program that examined all routines and common definitions to build a common routine usage map. This map listed for each common the routines that accessed that common. Frequently a named common was used only by a few routines, and it was a simple matter to assign these routines to a separate task. When this was possible, the associated common no longer needed to be part of the SGA, and the common requirements for addressing space in all other tasks were reduced.

There are two cases in which simple scanning of common declarations is inadequate. The first occurs when a copy of all common declarations has been put into all routines

```
;        .TITLE  RAYTRAC.CMD
;        .IDENT  /V02.03/
;        .DATE   21-OCT-78
.SETF LC ; DON'T LIST INTERNAL COMMENTS
.IFT LC ; COMMAND FILE TO COMPILE AND
.IFT LC ; BUILD TASKS FOR RAYTRAC,
.IFT LC ; PROGRAM 4 OF NOCOM SERIES.
.IFT LC ; ***** SYMBOL DEFINITIONS
.IFT LC ; COMR = COMPILE RAYTRAC
.IFT LC ; LISR = LIST RAYTRAC
.IFT LC ; LIBR = LIBRARY BUILD RAYTRAC
.IFT LC ; TKBR = TASK BUILD RAYTRAC
.IFT LC ; MAPR = MAP RAYTRAC
.IFT LC ; COMC = COMPILE RAYCOM COMMON
.IFT LC ; LISC = LIST RAYCOM
.IFT LC ; TKBC = TASK BUILD RAYCOM
.IFT LC ; MAPC = MAP RAYCOM
.IFT LC ; ***** OPERATOR INQUIRY
.ASK COMC  SGA COMMON COMPILATIONS
.SETF LISC
.IFT COMC .ASK LISC  SGA COMMON LISTING
.IFT LC ;--- BUILD SGA IF COMPILED
.IFT COMC .SETT TKBC
.IFT COMC ; SGA TASK WILL BE BUILT
.IFF COMC .ASK TKBC  SGA TASK BUILD
.ASK COMR  RAYTRAC COMPILATIONS
.SETF LISR
.IFT COMR .ASK  LISR RAYTRAC LISTINGS
.IFT LC ;--- BUILD LIBRARY IF COMPILED
.IFT COMR .SETT LIBR
.IFT COMR ; RAYTRAC LIBRARY WILL BE BUILT
.IFF COMR .ASK LIBR  RAYTRAC LIBRARY BUILD
.IFT LC ;--- BUILD RAYTRAC IF COMPILED
.IFT TKBC .SETT TKBR
.IFT TKBC ; RAYTRAC TASK WILL BE BUILT
.IFF TKBC .ASK TKBR  RAYTRAC TASK BUILD
.SETF MAPR
.IFT TKBR .ASK MAPR  RAYTRAC MAP LISTINGS
.IFT LC ; ***** FUNCTION EXECUTION
.IFF COMC .GOTO 10
F4P RAYCOM,RAYCOM/-SP=RAYCOM/I4/CO:50
.10: .IFT LISC QUE RAYCOM.LST
.IFF COMR .GOTO 20
F4P RAYTRAC,RAYTRAC/-SP=RAYTRAC/I4/CO:50
.20: .IFT LISR QUE RAYTRAC.LST
.IFT LIBR LBR RAYTRAC/CR,=RAYTRAC
.IFT TKBC @RAYCOM.CMD
.IFT TKBR TKB @RAY0.TKB
.IFT TKBR TKB @RAY1.TKB
.IFT TKBR TKB @RAY2.TKB
.IFT TKBR TKB @RAY3.TKB
.IFT TKBR TKB @RAY4.TKB
.IFT MAPR QUE RAY0.MAP
.IFT MAPR QUE RAY1.MAP
.IFT MAPR QUE RAY2.MAP
.IFT MAPR QUE RAY3.MAP
.IFT MAPR QUE RAY4.MAP
```

```
;        .TITLE  RAYCOM.CMD
;        .IDENT  /V01.03/
;        .DATE   22-AUG-78
;  COMMAND FILE TO TASK BUILD AND INSTALL
;        SYSTEM GLOBAL AREA, RAYCOM.
;
TKB @RAYCOM.TKB
HEL [1,1]
PIP RAYCOM.*;*/DF
PIP SY:[1,1]=[302,6]RAYCOM.TSK,RAYCOM.STB
REM RAYCOM/LI
INS [1,1]RAYCOM/LI/ACC=RW/UIC=[1,1]
HEL [302,6]
```

```
;        .TITLE  RAYCOM.TKB
;        .IDENT  /V01.03/
;        .DATE   22-AUG-78
; INDIRECT COMMAND FILE TO TASK BUILD
;        SYSTEM GLOBAL AREA RAYCOM.
;
RAYCOM/PI/-FP,RAYCOM/-SP/CRF,RAYCOM/-MD=
RAYCOM,XTDEF,XTCOMM
[1,1]SYSLIB/LB:XTRGSR:XTCVPB:XTALCO
[1,1]SYSLIB/LB:XTERRH:XTSEND:XTRECV
/
STACK=0
UNITS=0
ACTFIL=0
MAXBUF=0
FMTBUF=0
//
```

```
;        .TITLE  RAY2.TKB
RAY2/CP,RAY2/-SP/CRF=RAY2/MP
COMMON=RAYCOM:RW
;
UNITS=6
ACTFIL=1
ASG=LP:1:2:3:4:5
ASG=TI:6
//
```

```
;        .TITLE  RAY2.ODL
;        .IDENT  /V01.02/
;        .DATE   22-AUG-78
;        COMPUTER SCIENCES CORPORATION
;        RAYTRAC TASK 2 - RAYTR2 ROUTINES
         .NAME RAY2
         .ROOT RAY2-XTMAIN-XTD002-S
S:       .FCTR S1-S2-S3-S4-S5-S6-S7-ABNORM
S1:      .FCTR RAYTRAC/LB:RAYTR2:CALLA1
S2:      .FCTR RAYTRAC/LB:CALLA2:CALLA3
S3:      .FCTR RAYTRAC/LB:CALCD:CALCFO
S4:      .FCTR RAYTRAC/LB:CALVFO:DWREF
S5:      .FCTR RAYTRAC/LB:EENTER:NPSER
S6:      .FCTR RAYTRAC/LB:UPREF:FINDPT
S7:      .FCTR RAYTRAC/LB:BSINF:BCOSF
         .END
```

Figure 5. Indirect Command File Examples

without regard to usage. The second occurs when all common usage is restricted to unnamed common, so that only a single common exists. In these two cases, a routine analyzer that parses the complete FORTRAN program is needed to determine actual program usage of common data. We are currently involved in developing a generalized parser, for use with other automated tools, which could provide the basis for an analyzer of common data usage.

Allocation of Routines to Tasks by Call Hierarchy. The final grouping of routines into tasks makes use of a calling hierarchy to minimize intertask communication. In order to perform this activity, a calling hierarchy diagram must be made available, preferably with routine sizes indicated. Using such a diagram, it is possible to allocate the remaining routines across a number of tasks, such that the task sizes are approximately equal and routines that call each other are usually allocated to the same task. To simplify the manual creation of the calling hierarchy tree, a simple scanning routine was developed to extract all call statements from a routine's source code. Using this output plus compilation listings, very little effort was required to generate a calling hierarchy tree with associated routine sizes.

Occasionally, routines were encountered which were called from just about every other routine. In every case, these routines were small and well behaved: the routines did not retain any information from one call to the next, and did not have any side effects. Usually these routines performed some sort of well defined mathematical function, such as a vector rotation. In these cases, rather than have a large number of intertask calls, it was decided to simply put a copy of the routine into every task that called it. Any well-behaved routine could be handled in a similar fashion.

Preliminary Task Building. The second step in the task generation is development of preliminary task builds for each of the tasks identified in the first step. These task builds provide the information necessary to evaluate the groupings established by the first step. Because these task builds are used primarily for evaluation purposes, the command files should be kept simple, so that initial creation is relatively easy.

No SGA references are specified for these task builds; all common references are resolved within the individual tasks. For all tasks except the I/O tasks, the options UNITS = 0 and ACTFIL = 0 are specified to minimize the space reserved for I/O support code (which was unused). In addition, the object file F4PNIO is included in the task build commands to provide additional space savings by eliminating most I/O routines. For the I/O task, the appropriate UNITS and ACTFIL options are specified, so that an accurate estimation of space requirements may be developed. All other options are left unspecified, since they have no effect on space allocation.

Examination of the resulting task build maps frequently revealed potential problems in terms of task space. These problems were resolved either by establishing a new grouping of routines or by developing an overlay structure for some of the tasks. This step proceeded relatively smoothly, except for occasional difficulties with sizing I/O tasks. Because these tasks referenced most commons defined for a program, contained considerable amounts of code, and required the FORTRAN I/O support routines, it was almost always necessary to develop an overlay structure. This problem is discussed in greater detail in the paragraph on overlay generation.

SGA Task Builds. The third step is generation of SGA task builds. A BLOCK DATA subprogram is written for each SGA, containing the common declarations and data statements which define the SGA data contents. Then the appropriate task build command file for each SGA is developed. These command files include automatic transfer of the SGA task image and symbol table files into the system UIC directory, forcing subsequent task builds to use the proper common definition. Before use of such command files, the transfer step was frequently left out during the building process. The resulting tasks were invalid, and the entire building process had to be repeated once the error was discovered. After the SGA task builds have been specified, appropriate SGA reference specifications are added to the task build of each program task.

The allocation of common data areas to an SGA should be carefully considered so that the total SGA size is nearly, but not more than, some multiple of 4K words (one page of addressing space). At least one SGA must be associated with all program tasks to support the intertask communication routines that are added in the next step. This SGA must reserve sufficient space for storing intertask calling arguments.

For programs that we have converted, the space required for intertask communication has ranged from 500 to 1,000 words. The construction of the intertask communications subroutines allows use of up to 500 additional words of common space if it is available. Thus, instead of trying to make accurate estimates of intertask communication data area requirements, we simply required that a common SGA, referenced by all tasks, leave at least 1,000 words of space unused at this stage.

Because of requirements placed on SGA usage and complexities in task building allocations, it was almost always the case that only a single SGA was defined for a program conversion. Either a common area was assigned to a single task not part of the SGA for use by routines only within that task, or it was defined as part of the SGA, for use by all routines within all tasks in the program.

Once these task build command files have been defined, the entire set of tasks is built again so that any new sizing problems can be identified. Again, it may be necessary to perform some regrouping or build overlays into a particular task.

One particularly frustrating error may occur at this point. In one of our conversions, a task build failed because of an out-of-memory reference error. This caused the task build to abort, and no map was produced—the location of the error could not be determined. Although computations indicated that the task build should have proceeded without exceeding the addressing space, the error persisted even after many routines were removed.

Detailed investigation finally revealed that the error was associated with a particular routine—if that routine alone were task built, the error occurred. The routine defined a common area larger than any other routine in the program and, in particular, larger than that specified in the BLOCK DATA specification for the SGA. Since the SGA had been assigned to the highest addressing page, the additional locations specified by the common in the routine exceeded the PDP-11 addressing space. Once the common sizes were equalized, the task build succeeded with its original specifications.

Errors in sizing or referencing are more likely to cause task build aborts when an SGA is specified as part of the task build. This is the primary reason SGA task building was delayed until this step. If task build specifications are developed in small, incremental steps, failures at any step can usually be resolved by reference to a previously successful task build. A task build abort such as the one described, for which no map was generated, can be very frustrating if task build commands are greatly changed from a previous build.

Intertask Communication Specification. The final step in task building is to add intertask communication routines and any missing task build specification details. Our original efforts at intertask communication consisted of source code modifications using SEND and RECEIVE directives to pass the calling information. In order to keep coding modifications relatively simple, only one-way calling was allowed. That is, if a routine within one task initiated a call to a routine in another task, then no routine in the second task could initiate a call back to any routine in the first task. This restriction caused a considerable increase in the complexity to the allocation of routines to tasks. Furthermore, when program modifications were required, any changes in the allocation of routines to tasks necessitated considerable source code

changes. We naturally concluded that these restrictions and complexities were highly undesirable and should be eliminated.

In order to remove these difficulties, a set of reentrant, position-independent subroutines was developed to support intertask communication. The subroutines were specified so that intertask calling is transparent to the source program. Figure 6 lists the subroutines and files that provide intertask communication functions.

File	Subroutine	Description
XTRGSR	$XREGS	Register save in SGA (R0-R4, C, V).
	$XREGR	Register restore from SGA.
XTALCD	$XALCW	Allocate words from X$WORK common area.
	$XALCB	Allocate bytes from X$WORK.
	$XDALC	Deallocate space from X$WORK.
XTCVPB	$XCVPB	Convert parameter block from absolute to relative addresses and back.
XTERRH	$XERRH	Error announcement handler, TRAP instruction entry.
XTSEND	$XSJSR	Send subroutine call (JSR) to another task.
	$XSRTS	Send subroutine return (RTS) to another task.
XTRECV	$XRECV	Receive and process call from another task (JSR or RTS).
XTCOMM	$XCOMM	Defines intertask communication data area for inclusion in an SGA.
XTMAC	—	Macro definitions for intertask communication tables and code.

Figure 6. Intertask Communication Subroutines and Files

Several tables, listed in Figure 7, control the calling linkages established between tasks. Definition of these tables requires knowledge of which routines are the destination of intertask calls, which tasks call each routine, and the parameter list structure for each call.

Table	Contents
X$LINK	Task number, subroutine number, number of arguments, total argument size and relative address of argument sizes list in X$ARGS table.
X$TASK	Task name in radix-50.
X$ARGS	Subroutine argument size (bytes) for each subroutine and each argument in order.
X$ESUB	Task-dependent table of subroutine linkages for all calls to routines in other tasks (routines external to task).
X$ISUB	Task-dependent table of subroutine linkages to all routines called from other tasks (routines internal to task).

Figure 7. Intertask Communication Tables

For each task, it is necessary to specify which internally defined routines are accessible to other tasks and which routines in other tasks are called from that task. Both items of information may be determined by examining the task build maps produced in the previous step. For all tasks in which some routine calls a routine in another task, the external routine name will be listed as an undefined global symbol. All of these lists are joined to form a single list of all routines participating in intertask calling. Examination of the individual task build map listings allows identification of the task in which each of these subroutines is defined.

Finally, the calling sequence in terms of the size of each calling argument must be determined. The call statement extractor listing, generated by an automated scan program in the first step of task building, is used to construct this information.

Intertask Communication Definition Files. Once the intertask communication details are defined, an intertask communication definition file is generated, such as that shown in Figure 8. Each line in the file, except for .TITLE, .IDENT, and .END, ia a macro call. The TASK macro call defines the name of a task and the number of internal subroutines within that task accessible from other tasks. All SUB and ARG macro calls before the next TASK or DEND macro call are associated with that call. Each SUB macro call defines the sequential number of the subroutine in the list (starting with 0 and incrementing by 1), the name of the subroutine, and a variable number of macro parameters defining the size of each argument in bytes for that subroutine. If insufficient macro parameter positions are available, the ARG macro call is used to specify additional subroutine calling arguments. The DEND macro call terminates the definition of the last task.

```
.TITLE  XTDEF.MAC
.IDENT /V01.02/
TASK    RAY0,1
SUB     0,COOR,4,4,4,4,4,4
TASK    RAY1,6
SUB     0,RREC1,4,4,4,4,4,4,4,4
ARG       4,4,10.*4,4,65.*4,65.*4,4,4
ARG       4,4,4,4,4,2.*4,4,10.*4
ARG       4,4,10.*4,4
SUB     1,XTRW,2,4
SUB     2,PRINTU
SUB     3,HEDPRN
SUB     4,PRTPRN
SUB     5,GLOSS,4,4,4,4,4,4
TASK    RAY2,4
SUB     0,RAYTR2,4
SUB     1,CALLA1,4,4,4,4,4,4
SUB     2,CALLA2,4,4,4,4,4
SUB     3,CALVFO,4,111.*4,4
TASK    RAY3,4
SUB     0,HRAYCA,4,4,4,4
SUB     1,MINIRA
SUB     2,NTERPL,4,4,4,4
SUB     3,MODECH,4,4,4,4
TASK    RAY4,2
SUB     0,PATH
SUB     1,FENTER,4,111.*4,111.*4,4
DEND
.END
```

Figure 8. Sample Intertask Communication Definition File

In Figure 8, communication requirements for five tasks are defined. The first task, RAY0, contains a single subroutine, COOR, which may be called from other tasks.

Calls to COOR may specify up to six arguments of 4 bytes each. For these program tables, the macro computation could have been used to allocate subroutine numbers. However subsequent individual task specifications include only portions of the subroutine definitions, so that subroutine numbers are required for correct resolution of subroutine references. In order to keep the macro call forms identical, subroutine numbers are required in program table specifications as well.

The intertask communication definition procedure involves specification of a large number of tables which must be consistent. In order to aid in the specification, the macro definitions and intertask communication subroutines contain a considerable amount of error checking.

For the macro definitions, all redundancies in the table definitions are exploited to provide consistency checking. For example, a subroutine number in the SUB macro call cannot equal or exceed the number of subroutines defined in the previous TASK macro call. During execution, the intertask communication subroutines verify the numbers and sizes of arguments defined within the calling task and the receiving task. In addition, the allocation and deallocation of argument transfer space in the SGA is closely verified. If a routine writes beyond an allowed argument space in the SGA (because it expects a larger argument), corruption of verification words is highly probable and the error will be detected during deallocation of the argument space upon return from the subroutine.

Error announcement messages are defined in a message file for handling by the MO message output handler task. The primary announcement for each error is followed by lines displaying internal communication subroutine registers and hardware registers at the time of the error. The last displayed error line initiates the FORTRAN error traceback so that the calling sequence in effect at the time of the error is identified.

Intertask Communication Task-Dependent Data. For each task, the intertask communication routines require two internal tables: one for outward calls and one for inward calls. The outward table defines a global entry point for each routine in other tasks called from this task. These entry points define two instructions which invoke the intertask calling routines. These routines transfer all the subroutine calling arguments into the task common SGA and then request execution of the task containing the subroutine desired. The calling task then enters a wait state for either a return from the called task to the original routine, or a call for execution of some other subroutine.

A task-dependent portion of the intertask communication routines forms the main program of the receiving task. This main program calls an intertask communication subroutine to receive a call from any other task. When a call is received, the parameter block addresses in the SGA are adjusted for the receiving task address space, and the receiving task inward table is used to determine the address of the subroutine to be called. The return from that subroutine is to the intertask communication subroutines, which restore parameter block information in the SGA and return to the originating task. This return information awakens the originating task.

The intertask communication subroutines transfer the SGA arguments back to the original calling argument locations and finally return to the original call. The transfer of control in both directions provides for transfer of registers R0 through R4 and condition codes C and V. The content of R5 is adjusted in the receiving task to point to the parameter block definition in the SGA. Due to PDP-11 coding complexities, condition codes Z and N are not transferred. None of the contents of stacks are transferred. This register transfer is sufficient to support all FORTRAN subroutine and function subroutine calls, and most MACRO-11 subroutine calls.

All the calling and receiving information is stacked, allowing task calls to proceed in an arbitrary fashion. Thus, a routine in task 1 may call a routine in task 2, which may, in turn, call a routine in task 1 before returning. Such a sequence of calls is properly handled by the intertask communication subroutines.

For each task, the task-dependent inward and outward tables are defined using a subset of the total intertask communication routine table definition. Figure 9 shows these task-dependent definitions for two of the tasks specified in Figure 8. Each definition includes at its head a SETTSK macro call which triggers the definition of inward and outward transfer control tables and defines the task for which the tables are built.

```
.TITLE  XTD000.MAC
.IDENT /V01.02/
SETTSK  RAY0
TASK    RAY0,1
SUB     0,COOR,4,4,4,4,4,4
TASK    RAY1,6
SUB     0,RREC1,4,4,4,4,4,4,4,4
ARG       4,4,10.*4,4,65.*4,65.*4,4,4
ARG       4,4,4,4,4,2.*4,4,10.*4
ARG       4,4,10.*4,4
SUB     1,XTRW,2,4
SUB     3,HEDPRN
TASK    RAY2,4
SUB     0,RAYTR2,4
TASK    RAY3,4
TASK    RAY4,2
DEND
.END
```

```
.TITLE  XTD002.MAC
.IDENT /V01.01/
SETTSK  RAY2
TASK    RAY0,1
SUB     0,COOR,4,4,4,4,4,4
TASK    RAY1,6
SUB     2,PRINTU
SUB     5,GLOSS,4,4,4,4,4
TASK    RAY2,4
SUB     0,RAYTR2,4
SUB     1,CALLA1,4,4,4,4,4,4
SUB     2,CALLA2,4,4,4,4,4
SUB     3,CALVFO,4,111.*4,4
TASK    RAY3,4
SUB     0,HRAYCA,4,4,4,4
SUB     1,MINIRA
SUB     2,NTERPL,4,4,4,4
SUB     3,MODECH,4,4,4,4
TASK    RAY4,2
SUB     0,PATH
SUB     1,FENTER,4,111.*4,111.*4,4
DEND
.END
```

Figure 9. Sample Task-Dependent Communication Definition Files

When the TASK macro call containing that task name is encountered, each SUB macro call generates another

entry in the inward table. All other TASK and associated SUB macro calls define the entries for the outward table. Thus, in Figure 9, task RAY0 contains a single inward table entry for subroutine COOR and defines four outward table entries. The last of these outward table entries allows routines within RAY0 to call subroutine RAYTR2 in task RAY2. Examination of the task table definitions for task RAY2 reveals that it provides an outward entry for subroutine COOR in task RAY0. This is an example of calling back to an originating task, which was described earlier. Task RAY0 contains the main routine for the multitask program. That main routine calls subroutine RAYTR2 in task RAY2 and that subroutine calls subroutine COOR defined in the originating task RAY0.

Intertask Communication Task-Dependent Code. The intertask communication table definitions also specify the primary task—that task containing the main routine from the original program that was converted and, therefore, the first task executed. The first task macro call specified in the task definition list is the primary task. Thus, in Figures 8 and 9 task RAY0 is the primary task.

The intertask communication task-dependent code for the primary task differs slightly from that for the secondary tasks. For the primary task, a single statement—a call to the subroutine XTINIT—is inserted in front of all other main routine executable statements. Secondary tasks are built with a main routine consisting only of a CALL XTINIT statement followed by an END statement.

For the primary task, the initialization code returns to the call. For secondary tasks, the initialization code immediately enters a wait for a call from an external task. The FORTRAN main routine for secondary tasks is used so that the FORTRAN error processing trace mechanism is properly initialized.

Task Overlaying

Overlays are specified for individual tasks when task addressing space is exceeded and reallocation of routines to other tasks (or a new task) is infeasible. The overlay construction methods differ little from those used for usual task builds. We have identified three problems requiring greater concern during task builds:

- Adherence to program hierarchy restrictions
- Routines dependent on calling history
- Overlays involving I/O tasks.

Hierarchy Restrictions. Task overlay hierarchies for tasks that participate as members of a multitask program differ from normal tasks in two important, related aspects. First, each overlay must include intertask communication routines in its root. Second, all routines accessible to external tasks participate in both the internal task calling structure and the external, overall program calling structure.

Thus, a task overlay structure could be consistent with its internal calling structure, yet violate the overall program calling structure. Figure 10 shows the simplest example of this problem. The calling sequence is: MAIN calls A, which calls B, which calls C. The two task overlay structures shown in Figure 10b task build correctly because there are no structure violations local to either task. However, when routine B in task 2 calls routine C in task 1, routine A is overlayed. Because the contents of routine A are overlayed before it has completed execution, the return to A will produce an error. This problem is particularly difficult to detect before execution.

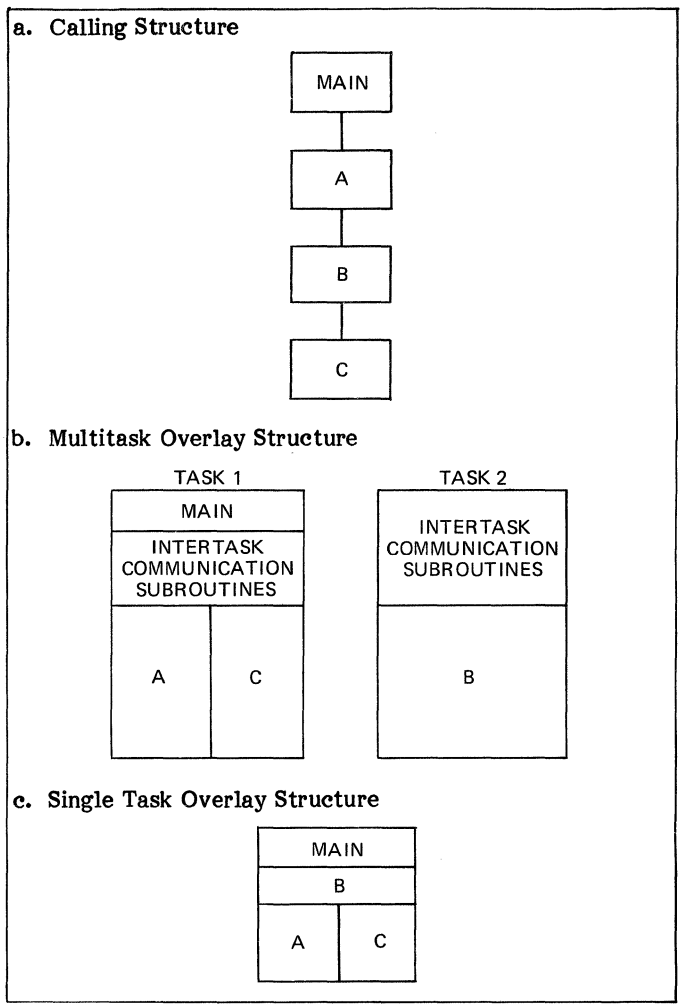


Figure 10. Calling Hierarchy Violation Example

The complete set of task builds and the program calling hierarchy must be referenced to identify the problem. This difficulty is different from that experienced with usual programming involving single tasks. For single tasks, a necessary condition for a calling error is that the overlay description contain a routine inversion. That is, some routine is farther from the root than another in the calling hierarchy, yet closer in the overlay hierarchy. Figure 10c illustrates such an inversion for a single task—routine B is farther from the calling hierarchy root than A, yet it is closer to the overlay hierarchy root. That single task, if executed, would produce the same type of calling error as described previously for multiple tasks.

For most programs, task overlay structure is based on the calling hierarchy so calling errors do not occur. For multitask programs, however, adherence to calling structures local to the task is insufficient. The overlay structure must also adhere to calling hierarchies for the program, across multiple tasks.

History-Dependent Routines. Routines that use information from a previous call are termed history-dependent. Data areas for history-dependent routines cannot be included in overlays because loading an alternative overlay would destroy the data, causing all information from previous calls to be lost.

It should be noted that history-dependent effects are excluded implicitly by paragraph 10.2.6 of the ANSI 1966 FORTRAN Standard. In spite of this restriction,

history-dependent routines are commonplace in large computer programs because overlays are infrequent and routine data values remain intact. Consequently, the problem appears more frequently in large program conversions than in usual minicomputer programming.

Solution of the problem first requires identification, but recognition of history dependency for large programs is a very complex problem. Consequently, all large programs that were converted were considered history-dependent unless manual inspection proved otherwise. For small programs, manual inspection is rapid and has a high probability of correct evaluation. Even if identified, removal of history dependencies may prove to be difficult. Again, automation appears to be of little use.

Handling of history dependencies, real or assumed, on an overall, routine basis seems more desirable if conversions are to be rapid. The solution is straightforward: move all local data definitions into a program section (PSECT) independent of the executable code. That PSECT can be placed in the overlay root (not overlaid) while the code is placed into an overlay branch, which is overlaid. For assembly programs, use of a separate PSECT defined with the attribute GBL (instead of LCL) will allow assignment of the PSECT to the overlay root. Unfortunately, FORTRAN defines the $VARS PSECT (containing local variables) with the attribute LCL, and no system mechanism allows modification of that attribute.

Specification of the read-only switch (/RO) does generate the required separation of data from code, but also produces an undesired side effect—separation of the task image into read-write and read-only areas. At task build time, these areas cannot share an addressing page, causing an average loss of 4K words of addressing space. This loss cannot be tolerated in these conversions. A compilation switch to control the GBL/LCL attribute or a task builder command to override PSECT attributes is needed.

Without access to PSECT attributes, conversion of history-dependent FORTRAN programs requires source code modification. All local variables involved (or assumed involved) in history-dependent computations must be placed into a named common defined for use only by the routine being converted. Then, during task building, that common area can be forced into the program root. Such modifications to source code are undesirable, and should be used only when no reasonable alternative exists.

I/O Task Overlays. The task assigned to contain all program I/O statements is likely to require overlaying. The I/O task usually contains large data areas in addition to common SGA references. Several entities, not required for non-I/O tasks, must be included with the I/O task:

1. FORTRAN I/O support routines
2. I/O buffers for each simultaneously active file
3. File Control Services (FCS) routines.

In addition, compilation of FORTRAN I/O statements usually produces large code segments.

The complex relationships among I/O and FCS support routines complicates construction of overlays involving these routines. Figure 11 shows the task build (RAY1.TKB) and overlay description (RAY1.ODL) command files of the I/O task defined for one of our conversions.

Several points should be made regarding these command files. Within the TKB file, the RAYCOM SGA is specified, as required for intertask communication. The ODL file specifies a relatively complex overlay structure consisting of two task-dependent intertask communication routines (XTMAIN and XTD001), common area definitions, non-overlaid applications routines (RNOIS and NWOMAP), shared object time routines, three overlay factors (F1, F2, and F3), and a co-tree. Object time is commonly, but incorrectly, used to describe routines that support execution. The co-tree specifies two overlay factors consisting of object time support routines.

Specification of I/O support routines in a co-tree allows reference by all routines in the root tree, regardless of their position in the tree. A primary disadvantage of co-trees is the possible creation of calling hierarchy errors in the same way as is possible for tasks in multitask programs. Consequently, co-tree construction requires careful attention to the routine calling hierarchy.

The overlay structure shown was developed through detailed study of DEC documentation and considerable testing. Removal of the references to BSINF and BCOSF in OTSUP1 produces a co-tree that can be used for the development of other I/O task overlay structures.

Two other points of interest. First, the XTRW routine in factor F3 is simply a collection of miscellaneous I/O statements taken from routines in other tasks as described in the paragraph titled Allocation of Routines to Tasks by I/O. Second, the definition of COM forces four commons—XRRECI, XPRNTU, XNWOMP, and XGLOSS—into the root. These commons were created from the local variables in four routines (RRECI, PRINTU, NWOMAP, and GLOSS) to remove history dependencies from those routines. The XNWOMP definition remains from an earlier overlay description in which NWOMAP was overlaid. The other commons are required since the associated routines are overlaid.

RELATED TOPICS

The methodology described only provides for conversion to an initial operational version of the program to be converted. Testing of an operational multitask program has not been described. In general, the testing methods are similar to those used to test multiple program implementations. Each task may be provided with debugging aids, trace printouts, and snapshot dumps as necessary.

During development of this methodology, deficiencies were identified in existing automated tools and in existing DEC software and documentation. Some of our observations about these deficiencies, and possible corrections, are discussed below.

Automated Conversion Tools

Several automated tools were developed for use when applying the conversion methodology. These tools are based on either TECO macros or on scanning routines written in FORTRAN. The FORTRAN subroutines were considerably simplified through use of a format scanning routine which had previously been written for a communications processor. Figure 12 illustrates one of the simpler scanning routines developed, and shows use of two different format processing subroutines at lines 22 through 24 and line 36.

```
;       .TITLE  RAY1.TKB
;       .IDENT  /V01.01/
;       .DATE   22-AUG-78
;  INDIRECT COMMAND FILE TO TASK BUILD RAYTRAC
;
RAY1,RAY1/-SP/CRF=RAY1/MP
COMMON=RAYCOM:RW
ACTFIL=5
ASG=SY:5,TI:6
//
```

```
;       .TITLE  RAY1.ODL
;       .IDENT  /V01.04/        AUGUST 9, 1978
;       COMPUTER SCIENCES CORPORATION
;       RAYTRAC TASK 1 OVERLAY DESCRIPTION
;
        .NAME RAY1
        .ROOT RAY1-XTMAIN-XTD001-COM-NONOVL-OTCOM-*(F1,F2,F3),COTREE
;*****************************************************************
;   FORTRAN COMMON AREA DEFINITIONS
COM:    .FCTR XRREC1-XPRNTU-XNWOMP-XGLOSS
        .PSECT XRREC1,RW,D,OVR,GBL
        .PSECT XPRNTU,RW,D,OVR,GBL
        .PSECT XNWOMP,RW,D,OVR,GBL
        .PSECT XGLOSS,RW,D,OVR,GBL
;*****************************************************************
;   NON-OVERLAID FORTRAN ROUTINES
NONOVL: .FCTR RAYTRAC/LB:RNOIS:NWOMAP
;*****************************************************************
;   OBJECT TIME SHARED ROUTINES (PART OF MAIN ROOT)
OTCOM:  .FCTR OTCOM1-OTCOM2
OTCOM1: .FCTR [1,1]SYSLIB/LB:$INITI:$OPEN:$CLOSE:$SAVRG
OTCOM2: .FCTR [1,1]SYSLIB/LB:$IOELE:$IOARY:$GETS:$PUTS:$ERRPT
;*****************************************************************
;   MAIN TREE OVERLAYS.
        .NAME READ      ;PRIMARY READ ROUTINE
F1:     .FCTR READ-RAYTRAC/LB:RREC1
        .NAME WRITE     ;PRIMARY WRITE ROUTINE
F2:     .FCTR WRITE-RAYTRAC/LB:PRINTU:GLOSS
        .NAME RW        ;COLLECTED READ/WRITE STATEMENTS
F3:     .FCTR RW-XTRW
;*****************************************************************
;   OBJECT TIME SUPPORT (INCL APPLICATIONS) OVERLAYS
COTREE: .FCTR *OTOVL-*(OTIOP,OTSUP)
;   ROOT OF CO-TREE, OBJECT TIME SHARED I/O SUPPORT
        .NAME OTSOVL
OTOVL:  .FCTR OTSOVL-[1,1]SYSLIB/LB:$NAM:$BACKS:$ENDF
;   OBJECT TIME I/O SUPPORT
        .NAME OTSIOP
OTIOP:  .FCTR OTSIOP-OTIOP1-OTIOP2
OTIOP1: .FCTR [1,1]SYSLIB/LB:$FIO:$ENCDE:$CONV1:$CONVR
OTIOP2: .FCTR [1,1]SYSLIB/LB:$ISF:$OSF:$ISU
;   OBJECT TIME ARITHMETIC, CODE, FILE SUPPORT
        .NAME OTSSUP
OTSUP:  .FCTR OTSSUP-OTSUP1-OTSUP2-OTSUP3
OTSUP1: .FCTR RAYTRAC/LB:BSINF:BCOSF
OTSUP2: .FCTR [1,1]SYSLIB/LB:$ALOG:$ATAN:$EXP:$JMOD:$SIN:$SQRT
OTSUP3: .FCTR [1,1]SYSLIB/LB:$MLJ:$PWRR
        .END
```

Figure 11. Sample I/O Task Build and Overlay Description Command Files

These format processing subroutines are being utilized in the development of a generalized parser for FORTRAN. Completion of this parser will provide for relatively easy development of a number of automated tools. A calling hierarchy generator and common data usage map generator are two such tools planned.

Two improvements to the intertask communication subroutines have been considered. The first would entail a modification of the argument processing routines so that any arguments already located within the SGA address space would not be transferred into the intertask communication area, thus reducing intertask communication data space requirements.

The second improvement would be to modify the argument handling routines to accept variable length arguments. In the current routines, an argument is optional in a call, but

```
0001              PROGRAM SCAN
                  C       .IDENT  /V01.03/
                  C       COMPUTER SCIENCES CORPORATION
                  C       SOURCE PROGRAM ANALYSIS TOOL
                  C       CREATED: 26-JUL-78
                  C
                  C THIS PROGRAM PRINTS OUT LINES WITH 'PROGRAM', 'SUBROUTINE', OR
                  C A SPECIFIED INPUT STRING.  COMMENT LINES ARE IGNORED.
                  C CONTINUATION LINES AFTER A PRINTED LINE ARE ALSO PRINTED.
                  C
0002              BYTE FILE(40),INPUT(40) !FILENAME AND STRING TO SEARCH FOR
0003              BYTE LINE(80),OUTCNT    !LINE FROM FILE, PRINT LINE COUNT
0004              BYTE CC,INDENT          !CARRIAGE CONTROL, INDENTATION COLUMN
0005              INTEGER LNG             !LENGTH OF INPUT STRINGS
0006              INTEGER MP,ML           !STRING MATCH POSITION, MATCH LENGTH
                  C******************** START OF PROGRAM
0007     1        TYPE 850
0008     850      FORMAT ('$ENTER FILE NAME: ')
0009              ACCEPT 800, LNG,FILE
0010     800      FORMAT (Q,40A1)
0011              IF (LNG .EQ. 0)  STOP
0012              FILE (MIN(LNG+1,40))=0  !END FILE NAME WITH 0
0013              TYPE 855
0014     855      FORMAT ('$ENTER STRING FOR SEARCH: ')
0015              ACCEPT 800, LNG,INPUT
0016              INPUT(MIN(LNG+1,40))=0  !END STRING WITH 0
0017              OPEN (UNIT=1, NAME=FILE,TYPE='OLD',BUFFERCOUNT=2,READONLY)
0018              IOUT=0                  !NO LINES PRINTED YET
                  C******************** MAIN SEARCH LOOP
0019     100      READ (1,805,END=900)  LNG,LINE
0020     805      FORMAT (Q,80A1)
0021     110      IF (LINE(1) .EQ. 'C')  GO TO 100        ! IGNORE COMMENT LINES
0022              IF (FMTFND(MP,ML,LINE,LINE(LNG),'PROGRAM'))  GO TO 120
0023              IF (FMTFND(MP,ML,LINE,LINE(LNG),'SUBROUTINE')) GO TO 120
0024              IF (FMTFND(MP,ML,LINE,LINE(LNG),INPUT))  GO TO 130
0025              GO TO 100
                  C******************** SUCCESSFUL SEARCH PROCESSING
0026     120      INDENT=2        ! INDENT 2 SPACES FOR PROGRAM OR SUBROUTINE
0027              GO TO 200
0028     130      INDENT=20       ! INDENT 20 SPACES FOR SEARCH STRING
0029     200      IOUT=MOD(IOUT+1,56)     !NEXT LINE, 56 LINES PER PAGE
0030              CC=' '                  !CARRIAGE CONTROL = SINGLE SPACE
0031              IF (IOUT .EQ. 1)  CC='1' !...OR NEW PAGE IF FIRST LINE
0032              IOUT=MOD(IOUT+1,56)
0033              WRITE (6,860)  CC,LINE
0034     860      FORMAT (A1,T<INDENT>,80A1)
0035              READ (1,805,END=900)  LNG,LINE
                  C ** CHECK FOR ALPHANUMERIC OR "+" IN COL 6, OR NUMERIC AFTER A TAB.
                  C ** IF FOUND, IT'S A CONTINUATION LINE AND IS PRINTED AS WELL
0036              IF (FMTMCH (MP,ML,LINE,'     &','     +','      #')) GO TO 200
                  C ** OTHERWISE, RESUME SEARCH BY CHECKING THIS LINE FOR SPECIAL STRINGS
0037              GO TO 110
0038     900      CLOSE (UNIT=1)
0039              GO TO 1
0040              END
```

Figure 12. Sample Source Code Scanning Tool

can only be a single size. For the programs we have converted so far, we have experienced only one variable length argument. In that case, it was possible to put all calls to that subroutine in the same task as the subroutine, so that it did not participate in the intertask communication. To support variable length arguments, interpretation of other argument values would be required to allow computation of the argument size during execution.

The limited application of intertask communication routines limits any additional work on them. Improvements such as those described above will be made only if absolutely required by some future large program conversion. The FORTRAN parsing programming efforts, which have applications outside the area of program conversion, will be pursued regardless of program conversion requirements.

DEC Software

In general, the DEC system software and documentation provide a sufficient basis to support the conversion process. The use of the indirect command file processor and TECO editing utilities provided considerable power for very little expenditure of effort. Although both these

utilities are unsupported by DEC for RSX-11D, we found them easy to add to the system and without problems during use.

There are, however, a number of minor deficiencies in other DEC software that prevent straightforward conversion. These deficiencies have been separated into three areas: SGA generation deficiencies, FORTRAN deficiencies, and RSX-11 compatibility deficiencies.

SGA Generation. In spite of some relatively good descriptions and presentation of examples in the task builder manual, many concepts are incompletely or incorrectly described. The RSX-11D documentation is particularly obscure when describing the interaction of task builder and installation switches for construction of, and reference to, an SGA. While an adequate tutorial of the problems is impossible in this paper, we would like to present some clarifications we obtained by running a number of test programs.

Figure 13 illustrates the five binary options associated with building and installing an SGA task and building a task to reference that SGA. The first option, the PI switch, is independent of the rest of the options. The switch specifies that the SGA be built position-independently, such that reference to the SGA may use any addressing space available in the referencing task, independently of other referencing tasks. Negation of the switch specifies that the SGA be associated with a specific addressing area which is the same for all referencing tasks. The only logical reason for specifying position dependency is that the SGA includes some code that was written in a position-dependent fashion. Since data is inherently position-independent, it should be possible to associate data with either a position-independent or position-dependent SGA without any problems.

SGA Task Build Options

1.	PI	-or-	-PI

Referencing Program Task Build Options

2.	LIBR	-or-	COMMON
3.	:RW	-or-	:RO

SGA Installation Options

4.	/RW	-or-	/RO
5.	/LI	-or-	/CM

Figure 13. SGA Construction Options

While this is true for most data areas that a programmer might define through assembly code, it is not the case for FORTRAN COMMON areas. In order to eliminate naming conflicts with subroutines, resolution of FORTRAN COMMON is performed using program section (PSECT) names. For some reason—which we were unable to determine—the task builder is capable of resolving PSECT names between a task and an SGA only when the SGA is defined as position-independent. Thus, FORTRAN COMMON must be included only in position-independent SGAs. On the other hand, since almost all DEC-provided support programs are position-dependent, definition of an SGA containing those programs must be position-dependent. The result is that combining support programs with FORTRAN COMMON into a single SGA to reduce addressing space waste is not allowed.

The second option displayed in Figure 13 is provided in the task build options only for documentation purposes. Selection of either the COMMON or LIBR keyword identifier for specification of an SGA reference has no effect on the task build operation or usage of the SGA.

The third option, specification of read-only access or read-write access to an SGA by a reference task, interacts with the fourth option—specification of read-only or read-write limitations on the SGA during installation. The two options combine to create four possible outcomes:

- If the reference task specifies read-only access and the SGA installation specifies read-only limitations, the memory management control registers will restrict all reference tasks to read-only access. Any attempt by that task to write into the SGA will cause a memory fault.

- If the reference task specifies read-only access and the SGA installation specifies read-write limitations, that particular reference task will be limited to read-only access, although other reference tasks may read and write into the SGA. As before, any attempt by the reference task to write into the SGA will cause a memory management fault.

- If the reference task specifies read-write access and the SGA installation specifies read-only limitations, the reference task will fail to install (and, therefore, run) because of an SGA access mode incompatibility.

- If the reference task specifies read-write and the SGA specifies read-write limitations, the reference task will be able to read and write freely into the SGA space.

Finally, the two SGA installation options interact to determine SGA resolution following usage. When no active tasks reference a particular SGA, that SGA may be considered to be in the dormant state. When an SGA is dormant, the only SGA image maintained by the system is the task file. When one or more referencing tasks become active, the SGA is also considered active.

When an SGA becomes active, the file image is copied into main memory and all modifications to the SGA are maintained in the main memory image. When all referencing tasks again become inactive and the SGA becomes dormant, the main memory image may either be discarded or written back into the file image, so that it is retained until the SGA becomes active again. If the access limitation specifies read-only, the main memory image will be discarded because the access limitation precludes any modification relative to the disk file image. If the access limitation specifies read-write, the common (CM) or library (LI) switch determines the resolution at the end of usage. The main memory image is copied to the file image if CM was specified at installation, and it is discarded if LI was specified. The implication is that any modification to an SGA library area only has meaning during a particular usage of the SGA, whereas modifications to a common area have meaning across SGA usages.

Many of the programs that we converted included BLOCK DATA COMMON specifications, which provided for initialization of common areas. Often, these initialized data areas were included as parts of an SGA. The traditional method for providing SGA initialization is to maintain a second file image of the SGA and to copy that second image into the first image prior to program execution. Because the particular multitask implementation that we chose guarantees that all referencing tasks

will remain active until execution is complete, it is possible to obtain the same initialization effect simply by specifying the LI switch during SGA installation.

FORTRAN Deficiencies. During the conversion efforts, three deficiencies in the F4P compiler were noted:

1. Lack of VIRTUAL array support
2. Inaccessibility of error processing
3. Inaccessibility of PSECT attributes.

These deficiencies are discussed in the following paragraphs.

The F4P compiler does not include support for VIRTUAL specification of arrays to support data areas greater than 32K words. While this deficiency has not prevented any conversions to this date, its lack has caused some difficulties in what would have otherwise been rather straightforward conversions. Since virtual arrays are provided by the FOR compiler under RSX-11M, this feature should be added to F4P as soon as posssible. The F4P is described by DEC as a superset of FOR; this is not true for virtual array support at the current time.

The F4P support routines for error processing do not allow user definition of new error calls. This is a deficiency because it prevents a user from utilizing an otherwise very well thought out error announcement mechanism. The internal control tables for error processing are not directly available to the user, so development of user error announcements involves a considerable amount of code in order to use the MO message output handler. Access to the error definition tables would allow a user to provide error controls and announcements just by defining a few byte values and using the TRAP instruction mechanism already implemented. While not necessary, access to such a facility would make the programming task easier.

The F4P compiler does not allow user modification of PSECT attributes. As described in the overlay generation section, specification of the GBL instead of LCL attribute in the $VARS program section would allow overlaying of FORTRAN subroutines without regard to their use of local variables. Although the ANSI FORTRAN standard specifies that subroutine local data values will be undefined upon each invocation of a subroutine, many programs violate the specification by using variable values from previous subroutine calls. Overlaying of such subroutines produces execution errors.

If the $VARS program section could be given the attribute GBL, variables would not be involved in program overlays, and overlaying would not affect execution. Although correction of this deficiency is not necessary to fulfill any formal FORTRAN specifications, its addition would not violate specifications, and would remove some of the problems faced during conversion efforts.

Operating System Incompatibilities. Several papers have been written describing incompatibility problems between RSX-11D and RSX-11M. This section is only intended to identify those incompabilities which created significant problems in developing the intertask communication subroutines. The primary area of difficulty involved the different implementation of send and receive directives in the two systems.

For RSX-11D, intertask communication was provided by straightforward use of the "variable send and request or resume" directive in the originating task and the "variable receive or suspend" directive in the receiving task. For RSX-11M, the separate use of a send directive followed by a request or resume directive produces a race condition; execution could cause a deadlock in which all tasks are waiting for other tasks to call them. In order to eliminate this race condition, it was necessary to use receive asynchronous system traps and internal event flags. The solution involved more code and used up a resource (event flag) that might be needed by applications programs. Thus, it is concluded that RSX-11M directives for intertask communication are deficient with respect to those provided for RSX-11D.

One other difference in the operating systems may become significant in a conversion effort. Currently, RSX-11D contains no virtual map space handling directives. Virtual array support requires these directives. In this respect, the RSX-11M operating system is superior to RSX-11D.

CONCLUSION

A methodology for converting very large programs to the PDP-11 has been presented. The conversion is based on five separate activities, supported by automatic tools. Application of this methodology to several conversions has proven its worth. Although a large conversion still remains difficult, the methodology provides systematic generation of a solution more rapidly than with other approaches. Further, fewer errors remain after creation of the first operational version of a program.

During development of the methodology, DEC software and documentation was adequate in most cases. Specific, minor deficiencies were described. Automated tools supporting all phases of a conversion were successfully and easily developed. Plans for enhancements and additions to conversion tools were described. Automation of most of a conversion effort is considered to be feasible.

ACKNOWLEDGMENTS

The author is indebted to Janet Comfort, Computer Sciences Corporation, for her programming help. Her willingness to try alternate conversion methods and perform numerous tests provided much of the information necessary to the definition of the conversion methodology. The author is also indebted to Janene Deller, Computer Sciences Corporation, for her technical editing and generous assistance with preparation of this manuscript.

RUNNING "REAL-TIME" WITH IAS

Bolson E., Frimer M. Cardiovascular Research and Training Center, University of Washington, Seattle, Washington.

ABSTRACT

Several methods have been studied to implement real-time features in the interactive time-sharing system, IAS. This report discusses the development of communication routines between real-time tasks and time-shared tasks, and the implementation of a compact, "pseudo-handler" task which overcomes several problems in the former.

Introduction.

The Cardiovascular Research and Training Center's (CVRTC) Data Acquisition Laboratory is used for a wide range of heart research requiring analog/digital conversion, x-ray image X/Y digitizing, and graphic display. The computer facility presently consists of a PDP 11/45 with 124K memory, RK06 disk, 3 RK05 drives, 32 A/D lines, and about 20 terminal lines, of which several are dial-up at 1200 baud. Also available are several locally interfaced data acquisition units, primarily an Autotrol large table X/Y digitizer, and a 9600 baud communication line to a Decsystem-10. Until recently, (December, 1977), the operating system was RSX-11D, which worked relatively faithfully for the computer-wise staff. As usage by unsophisticated users such as medical personnel and technical staff increased, it was determined that IAS V2.0 would provide compatibility with RSX-11D while improving resource utilization (especially response time) and "user-ability" in an interactive environment.

Crucial Application.

One major application, which was added near the time of this conversion to IAS, is the remote usage of Tektronix (TM) 4953 digitizer tablets in conjunction with Tektronix 4012 graphics terminals at 1200 baud. These tablets respond to ASCII commands and send a series of manually digitized coordinates to the computer while optionally plotting on the terminal.

Since the coordinates are sent continuously as long as the user depresses the cursor button, prompt acceptance of the digitized data over the terminal lines is necessary. This basically real-time operation caused quite a problem with the time-sharing system.

Straight Time-sharing

The first attempt to acquire data using a time-sharing task seemed successful until system activity increased. (It should be mentioned here that at that time, only 112K was attached to the computer, causing swapping to occur with 3 and sometimes 2 users!) The problem was found to be caused by swapping; no matter how large the QIO input buffer was made (within reason), it could be filled up at 120 characters/second when enough activity occurred. Unfortunately there is no means within IAS to give a time-sharing task a "high-priority", and IAS and the TT handler are not structured to respond to massive terminal input. Double buffering wouldn't help, since the task must be active to switch buffers. The possibility of locking the task in core was initially disgarded due to the loss of multi-task swapping and small amount of core available. A separate partition could not be used either (due to the need to provide space for applications not using digitizers), for the same reasons.

Real-time/Time-sharing

IAS has a very flexible timesharing control structure, allowing spawning of subtasks to arbitrary depths, while maintaining control and communication between these subtasks. Unfortunately, the real-time executive portion of IAS, based as it is upon RSX-11, has no such control. Furthermore, there is no provision for communication between real-time and time-shared tasks. This installation successfully developed several communication routines based upon the SEND and RECEIVE DATA executive requests and the RECEIVE AST facility. Briefly, the AST (Asynchronous System Trap) allows a task to be informed when a SEND DATA has been queued for it, whether it is real-time or time-sharing. The reason for not using the RECEIVE DATA or SUSPEND request became obvious when PDS became active, informing the user that the job was suspended. PDS is the "monitor" or command language interpreter which controls all time-sharing users under IAS.

The communication protocol decided upon was to SEND DATA and REQUEST the real-time "sub-task" which would do the actual data acquisition, storing the data on a disk file. It of course merely did a RECEIVE DATA upon start up. The time-sharing "control-task", however, could not suspend, as

indicated before. Instead, it enabled a RECEIVE AST and waited for an event flag to be set. These are all non-priveleged executive requests. The "sub-task", when finished, executed a SEND DATA with a status word, and exited. The AST routine set the event flag in the "control-task", thus waking it up. The control-task then did a RECEIVE DATA to get the status.

The above sequence of events worked fine as stated. However, several inadequacies became apparent.

1. First, the real-time sub-task executed under the UIC it was installed with, usually the UIC under which it was built.

2. Second, because of other bugs, some of them DEC'S, the sub-task would lock up, and the user did not have the privilege to abort it.

3. Third, if the user typed "control C" and aborted the control-task, the real-time task would remain running.

4. Fourth, unless the real-time task had a priority greater than the time-sharing system tasks, it would not be allowed to run. When it was given a higher priority, there was a danger of thrashing or even complete system lockup, since the real-time task could not be checkpointed to make room for time-shared tasks.

Diligent research and brainstorming resulted in some solutions, though not all were good. It was discovered that building a task with a default UIC of [0,0] would allow it to execute in the UIC of the REQUESTing task. In order to allow the user to abort the real-time task, it was made "self abortable"; it would exit through a RECEIVE AST if sent a message once executing. This was possible because the information sent to the AST routine includes the event flags the task might be waiting for and the program counter (PC) address. The AST routine set each flag required and inserted the address of the FORTRAN EXIT subroutine in place of the previous PC in the stack, and performed an AST EXIT. This kludge successfully aborted the task.

To take advantage of this capability, the control-task had to be informed of a request for abortion. To implement this, the "control C" recognition time-sharing facility was used. It was found that the wait for terminal event macro had to have a sub-task data block, even a dummy one. the procedure was as follows: (see Figure 1)

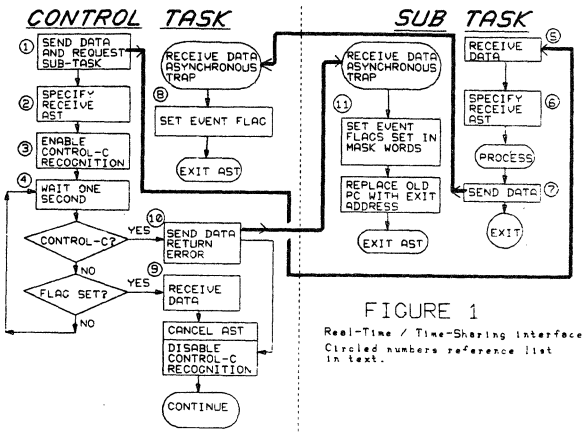


FIGURE 1

Real-Time / Time-Sharing Interface.
Circled numbers reference list in text.

1. SEND DATA and REQUEST real-time sub-task

2. SPECIFY RECEIVE AST.

3. ENABLE conrol C recognition.

4. Every second, test for control C or event flag set by AST.

5. In sub-task, RECEIVE DATA.

6. SPECIFY RECEIVE AST.

7. On normal finish, SEND DATA to control task and exit.

8. Control task, on RECEIVE AST, sets event flag.

9. Main loop detects event flag, RECIEVEs DATA, and continues.

10. Main loop, on detecting control C, SENDs DATA to sub-task, and tells control-task to abort. Actually the control-task just went to initialization portion of interactive program.

11. Sub-task, ON RECEIVE AST, sets all flags requested, and sets PC in stack to an EXIT routine, and exits. Task exits.

The above procedure worked. Unfortunately, the aforementioned bugs became important. It seemed that real-time and time-sharing tasks did not co-exist as peacefully as documented. Even with several DEC patches, one problem remained insoluble: if there was little system activity and no hole was available for the real-time task, whatever its priority, the task would never be loaded. The frustrated user would control C, the control task would respond, but the real-time task would not even start until the control task exited, causing mass confusion. If that wasn't enough, it became apparent that more than two users would

cause havoc in the partition, because the time-sharing tasks had to come up every second, due to the test for control C, and the real-time tasks, even though half sharable, were still too big, since they didn't seem to move around much.

Pseudo-handler

Finally, it was determined to scrap the previous development, if a suitable small, sharable, highly communicative, and high priority task could be written. A pseudo-device handler fit the bill perfectly. It could run at high priority, was small by definition of system requirements, could communicate through QIO'S and event flags, and could be written to handle an arbitrary number of terminals. It is improbable such a solution would have been attempted, however, had not the previous attempts failed.

After specification and design, three weeks of intensive effort provided a tablet handler (TB) with the following characteristics:

1. Since no re-entrancy is required, TB polls through a list of nodes waiting for tablet input. A polling rate of 6 per second was found to be sufficient.

2. Nearly all storage is in the QIO node and a 40 word node picked from system common. There is no built in limit to the number of terminals TB can handle in this way.

3. TB emulates the hardware functions of the Autotrol digitizer, allowing users familiar with one unit to use the other without re-training. In addition, it saves the calling program work by interpolating and sieving points digitized.

4. TB frees the calling program for swapping until the user has entered the first control key from the terminal. The terminal handler read-ahead buffer is of sufficient size to allow time for the task to be requested and swapped in. The task is then locked in core until the number of points requested have been digitized. If in the future this technique is insufficient for the number of users, nodes might be picked to store data temporarily. However, so many users would tax all system resources so badly that allowing such usage would be inadvisable.

Conclusion

The slow, painful development of the TB handler has been rewarded by better understanding of the operating system, and a smooth, user-oriented package for manually digitizing picture data. Although reports were that IAS could not conveniently handler more than 3 users on a PDP 11/45, this installation has managed to handle 6 users, including program developers. Despite the previously described problems, the real-time/time-sharing control structure has been successfully used to control a task for A/D conversion which must run at very high priority. This task is small enough that it usually has a space for loading.

In summary, IAS is capable of handling users with interactive and real-time requirements on a relatively small computer system.

REPLACING MCR IN AN OEM ENVIRONMENT

David M. Kristol
Massachusetts Computer Associates
26 Princess Street
Wakefield, Massachusetts 01880

ABSTRACT

This paper describes a user-written command interpreter which replaces MCR for an OEM laboratory system. The replacement, UMCR, intercepts unsolicited input that normally goes to MCR, and it can spawn and control tasks. In addition UMCR has a command language which is well-suited to the laboratory user and includes several unusual features.

BACKGROUND

In early 1977 Massachusetts Computer Associates (COMPASS) was hired to build a data acquisition and analysis system for a laboratory instrument manufacturer. The system, which we will call UMCR, was to support multiple users acquiring and analyzing data from multiple instruments simultaneously. It had to permit users to analyze data as it was being acquired. A sophisticated command language would offer the experienced user great flexibility, yet, by making many default assumptions, help the novice user to get started quickly. The command language should be verbose to help the novice, yet allow abbreviations to reduce keystrokes. Also, it should be a "soft" system, tolerant of user errors. Finally, the host system should support program development activities concurrently with the laboratory-oriented processing. A PDP-11, running RSX11M, was the obvious candidate for supporting the UMCR system.

RSX11M presented its share of obstacles, of course. Most of them arose in the command interpretation area. The RSX11M command interpreter, MCR, was an unsatisfactory user interface because it gave a novice user too many ways to get into trouble and because it could not possibly support the planned command language without extensive modification. Clearly, then, some other task would serve as the user's interface to the system.

Making some other task the command interpreter (CI) had its share of problems, too. The user would still be free to type ↑C to reach MCR and get into trouble. Worse still, by typing input when the CI had no read queued for the terminal, the user could inadvertently send characters to MCR. The result could be confusing if MCR rejected the input with an error message, or disastrous if the input accidentally proved to be a valid MCR command.

A straightforward, modular way to implement the various acquisition and analysis functions is as separate tasks. However, an independent CI has only limited task control abilities. Since the RSX11M Executive permits only one copy of a task to run at a time, clearly the CI had to mimic MCR's ability to run multiple copies of a task by running them under different names (aliases). In addition, UMCR was required to run multiple copies of the same task from one terminal, whereas MCR could only run one.

To round out the CI's capabilities, some task control facilities were considered essential. The CI had to be able to abort a task at the user's request and had to be able to tell when an acquisition or analysis task had exited.

To recapitulate, the major problems with using RSX11M for the UMCR system were in the areas of terminal control and task control. As it turned out, the necessary building blocks were readily available within RSX11M to solve these problems. The following sections describe our approach, the resulting implementation, and the command language itself.

APPROACH

Task control was the easier problem to surmount. The system would consist of a set of "UMCR tasks" communicating with the CI by a simple protocol. These UMCR tasks would perform the data acquisition and analysis. To run one, the CI would do the equivalent of the MCR RUN command, using an alias for the task name. The UMCR tasks would be required to announce their imminent exit by sending a message to the CI.

The only message from the CI to a UMCR task would request the task to abort itself. The ABRT$ Executive directive was not satisfactory for this purpose because (1) the aborted task would have no chance to clean up, (2) the message appearing on the

terminal would be confusing to the UMCR user, and (3) the issuing task (CI) would have to be privileged in a system with multi-user protection.

Terminal control appeared to complicate our design considerably until an elegantly simple approach was found. We wanted the CI to be "live" all the time, in the sense that the user could always enter an escape character (↑C) and wake it up. This facility would allow the CI to abort a UMCR task even when output was coming to the user's terminal (TI).

The first approach we considered would have the CI issue an "attach" to the terminal and field unsolicited character ASTs. Of course, this approach was unsatisfactory because, with the CI attached, no other UMCR tasks could output to the terminal.

A second approach would have the CI "pass" its attachment, somehow, to another task, which would then be responsible for attending to ↑C from the terminal. However, if we used standard RSX11M facilities there would be a brief "time window" during which neither the CI nor the other UMCR task was actually attached to the terminal. Also, this approach required too much logic to be present in the analysis task, even if the CI did not designate that task to take control of TI. Having recognized a ↑C, this task would have to pass the attachment back to the CI so the CI could prompt for input.

The final unsatisfactory approach would create a "terminal control task" which would do all reads and writes on the user's terminal and handle ↑C's. We feared that this approach would generate too much Executive activity by passing input and output around, thus slowing the system and using dynamic memory. Also, this task would probably take a fair amount of space, and there might have to be one copy per terminal.

Our final approach, by comparison, was quite simple. Upon reflection we recognized that we wanted the CI to behave just like MCR, but not be MCR. MCR is always "live" because the terminal driver passes to MCR all input for which there is no other outstanding input request. This type of input is called "unsolicited input". Even after the '>' prompt, MCR does not make an input request, which is why output may occur on the terminal after a prompt has appeared.

Clearly if we wanted our CI to treat the terminal the way MCR does, we should modify the terminal driver to do exactly that. We would create a special "UMCR mode" in the terminal driver. Once a terminal was in UMCR mode, all unsolicited input would go to the task that put it in that state, rather than MCR. Furthermore, if ↑C was typed, a 'UMCR>' prompt could be produced.

The idea of modifying RSX11M did not immediately appeal to us because such changes could lead to other problems, DEC would not support them, and they would complicate the system generation procedure. Nevertheless the alternatives presented at least as many headaches, and the resulting system would not run as smoothly.

IMPLEMENTATION

The required capabilities for UMCR seemed to point to the CI being a privileged task, since it would have to delve into RSX11M structures. However, the size limitation on privileged tasks was surely going to be a problem, and debugging such tasks raised the specter of frequent system crashes during program development. We therefore chose to make the CI non-privileged.

The decision to make the CI non-privileged had other virtues. Clearly the manipulation of RSX11M structures still needed to be done. However, our decision forced us to define those operations very carefully. The result was the specification of one very small task (UINS), with which the CI communicates, and of modest changes to the RSX11M Executive which are completely transparent to the rest of the system. UINS supports task spawning, and the Executive changes provide terminal support.

To run tasks, the CI passes the file name and alias of the desired task to UINS and waits for a local event flag to be set. UINS does not, itself, contain code to install and run tasks. Since the INStall part of MCR is adept at these functions, we decided to let it do the hard work. The undocumented '/RUN=REM' option in INStall is used by MCR to implement the RUN command (install, run, remove), so we used it as well. UINS simple queues an appropriate INStall command to MCR and waits for the requested task to begin. Then it sets the CI's local event flag.

The normal terminal driver places unsolicited input lines on MCR's receive-data queue. Since MCR is privileged, it can retrieve these lines, which are not genuine receive-data packets, and process them. When MCR processes a command line for an installed task like PIP, it places the result in the system's "GMCR queue". When PIP runs, it retrieves the line with a GMCR$ directive (get MCR command line).

Because the CI is non-privileged, we needed non-privileged ways to get the terminal into and out ot UMCR mode and to fetch command lines. Since we wanted to minimize changes to RSX11M, we used two existing mechanisms: QIO$ and GMCR$ directives. Special terminal QIO sub-functions set and clear UMCR mode. The GMCR$ mechanism is expedient for associating the unsolicited command lines with the CI. When TI is in UMCR mode, the terminal driver places

unsolicited input lines on the GMCR queue. The CI can then easily obtain them with the GMCR$ directive. (The CI does not get lines that are cleaned up the way MCR normally leaves them. Thus our use of GMCR$ is non-standard.)

The terminal driver modifications to support "UMCR mode" are conceptually simple. Facilities are added to allow a task (the CI) to put the terminal into, or take it out of, UMCR mode. Code is added to type 'UMCR>', instead of 'MCR>', when the terminal is in UMCR mode and ↑C is typed. Finally, at the point where unsolicited input is completed, code is added to note whether UMCR mode is in effect. If it is, the input line is placed on the GMCR queue, directed to the task which placed that terminal in UMCR mode.

Two other parts of the RSX11M Executive require modification to insure system integrity. The code which handles exiting tasks (DREIF) turns off UMCR mode for a terminal if the exiting task had put it in UMCR mode. Turning off UMCR mode makes the terminal revert to a normal RSX11M terminal, communicating with MCR. Also, the GMCR queue must be emptied of all entries addressed to the exiting task and not just the first one.

To support these changes for system generation, we added all of our modifications with conditional assembly symbols and modified the SYSGEN procedure. Since the RSX11M Version 3.1 SYSGEN is comprised of indirect command files, changes were easy to make. The modified SYSGEN asks "DO YOU WANT UMCR SYSTEM SUPPORT?". An affirmative answer defines the necessary symbols to produce that support. Thus our client can easily generate the systems they need without special digressions that could lead to errors.

COMMAND LANGUAGE

We wanted the UMCR command language to be simple to use, allowing an advanced user to invoke commands quickly while supporting the struggling new user. It was to offer string substitution and language extension via procedure files. It had to give users sufficient control structures to support running lengthly sequences of data acquisition and analysis while the system was unattended. The language facilities to achieve these goals are described below.

The commands that UMCR recognizes fall into three classes: external, control, and internal. External commands perform data acquisition or analysis and are realized as separate tasks. However, external commands are completely parsed before the associated task is run, the task receiving the result of parsing. Control commands implement control structures in procedure files. Internal commands are those commands (other than control commands) which the CI executes within itself. As with most command languages, the user enters the command name first, followed by any optional parts.

External commands often have many parameters and switches that select the exact processing desired. Typical designs for similar languages require users to specify all parameters in the exact order they are expected (positional notation), or to specify only certain parameters by name (keyword notation). The user who does not know the positional order, in the first case, or the keywords for parameters, in the second, is lost.

Consider some examples. Assume we are dealing with command CMD which has parameters with keyword names P1, P2, P3, and P4. Their positional order corresponds to the digit in their names. (Real parameters for a real command would have keywords like TEMPERATURE, MASS, VOLTAGE, etc. We use these simpler names here so the reader can remember their positional order.) To execute CMD with positional parameters the user says

```
CMD  [1],[2],[3],[4]
```

where [i] represents a value for parameter Pi. (Command lines end with Carriage Return to begin execution.) A purely keyword version of the same invocation might be:

```
CMD P1=[1],P2=[2],P3=[3],P4=[4]
```

or

```
CMD P3=[3],P1=[1],P4=[4],P2=[2]
```

or some other variant.

The UMCR command language merges positional and keyword notation in an unusual way. The result is a straightforward synthesis of those notations with prompting. In UMCR, a keyword changes the "current parameter number". Thus a command like:

```
CMD P4=[4],P1=[1],[2],[3]
```

is meaningful, and is equivalent to the previous examples. The use of the keyword P1= has changed the current parameter number to 1. Then the values [1], [2], [3] are treated positionally. Of course, the preceding P4= had changed the current parameter number to 4, as well. (Remember that real parameters are not named Pn, so the command interpreter does not use a digit in the keyword to determine the proper position. In a sense the keywords are symbolic position numbers.)

Prompting falls out of this scheme naturally. The user's prompt request character is ESCape (ALT MODE), represented by $ below. For example, if the user types:

```
CMD$
```

UMCR responds by prompting with:

```
P1=
```

and waits for the user's response. When the user enters:

```
[1]$
```

UMCR responds by prompting again, this time with:

```
P2=
```

The user can "mix and match" positional and keyword notation and prompting as shown in the equivalent examples below (underlined portions produced by UMCR):

```
(1)  CMD [1],[2],[3]$
     P4= [4]

(2)  CMD$
     P1= [1],[2]$
     P3= [3],P4=[4]

(3)  CMD [1],P4=$
     P4= [4],P2=$
     P2= [2]$
     P3= [3]
```

One glaring omission in the discussions above is the question of what happens if the user types:

```
CMD [1]
```

Does a (groan) "syntax error" occur? The answer is a definite "No!". UMCR contains a multi-level defaulting structure that attempts to supply a value for the omitted parameters. In fact the defaulting mechanism is one of the key ingredients toward making the system easy to use.

Human engineering considerations of the defaulting mechanism dictate that parameters of the same name (keyword) in different commands in fact be the same parameter. The user never has to decide "Is this the CMD1 version of parameter XYZ, or the CMD2 version?". Although UMCR has many parameters that are unique (or peculiar) to only one command, some are used by two or more commands.

The defaulting mechanism, as we said, applies to parameter values. There are three levels of default: base-level, local, and global. The base-level default is built into the parameter code and cannot be changed. The local default supplies the value of a parameter in the context of a specific command. The global default applies to any instance of the parameter in any command. The local and global defaults may be set by the user, and they retain their values until they are explicitly changed by the user. Their initial value is null (none).

To assign a value to a parameter, UMCR first uses any value supplied by the user on the command line(s). Next the local default, or, if there is none, the global default is used. In their absence the base-level default is used.

The previous discussion about specifying parameter values omitted the fact that default values are displayed with each prompt. The value displayed is the default that would be used (local, global, base-level) if the user specified no value. If we use the notation '<i>' to represent the default for parameter Pi, example (3) above would really look like:

```
(3)  CMD [1],P4=$
     P4= <4> [4],P2=$
     P2= <2> [2]$
     P3= <3> [3]
```

When the user finally types Carriage Return to begin command execution, there may be parameters for which UMCR requires a value and the user has specified neither a default nor a regular value. Also, some parameters the user supplied may have caused errors. In these cases UMCR prompts for a correct value. Once satisfied, UMCR will begin command execution.

Now we can see how various classes of users might make use of UMCR. The novice user will generally step through all parameters, supplying values as he goes. The experienced user will generally set defaults for parameters. She will use positional notation to supply a few parameters, since the most variable ones are usually first. Occasionally she will use keywords to change a specific item, or prompting, to verify defaults.

Using parameter defaults is one way the user can minimize input keystrokes. We should mention that UMCR recognizes commands and keywords by the minimum number of characters in a given context. Two further mechanisms for decreasing the number of keystrokes are string variables and the special symbol @. Arbitrarily named string variables may be assigned values and then used in any input to UMCR. For example, if [1], the value for parameter 1 in the discussions above, had a particularly long value, the user could assign the value to string variable A and type:

```
CMD 'A',[2],[3],[4]
```

to invoke the command.

The symbol @ is really just a special string variable name that does not require quotes ('). However, its content is constrained to be a file specifier string. Since files in the UMCR system often differ only by extension, this tool is very handy for selecting files rapidly in different parameters. For example, the user could

type:

```
ATSET  DK1:[100,75]F00
```

to set the value of @. In a command he might say:

```
CMD  @.DAT,@.LST
```

The string 'DK1:[100,75]F00' is substituted for each occurence of @.

The UMCR system includes a facility for running long sequences of commands automatically, like RSX11M's indirect command files. Two features of this procedure file mechanism are distinctive, however. First, a procedure file is invoked without a preceding special character (such as @ in RSX11M). Second, and more important, procedures may have arguments. The two aspects combine to make procedure files an effective command language extension mechanism.

The argument passing mechanism resembles parameter handling in external commands while using the string variable mechanism. Procedure files are called by specifying the filename in place of a command name. The balance of the command line is assumed to contain procedure arguments which are remembered.

Nothing is done with the arguments until the ARGS command is encountered. ARGS states the names and positional order of the arguments for the procedure. The names, in fact, are string variable names which are assigned values by matching up the arguments with the names. These string variables are then used like any others.

ARGS is slightly more complicated than just stated, however, because the user may also use keywords to specify procedure arguments in the procedure call. The keywords must match the argument names in ARGS. As with external commands, keywords in procedure calls change the current argument position number.

One other helpful detail about the procedure file mechanism is that arguments for which no value has been specified retain their previous value. Combining this with the keyword mechanism, a procedure file can set "defaults" for procedure arguments. If the procedure is called with some missing arguments, the default (string) values will allow the procedure to proceed.

An example will illustrate these points. Assume that we are writing a procedure PROC which has four arguments: A1, A2, A3, A4. (Here again, the names chosen are for convenience. In fact they may be arbitrary length alphanumeric symbols.) The files would probably start with the five lines:

```
SSET  A1=<1>
SSET  A2=<2>
SSET  A3=<3>
SSET  A4=<4>
ARGS  A1, A2, A3, A4
```

where <i> is a default value for Ai and SSET sets a value of a string variable

The rest of the procedure file depends on the function to be performed. References to (and alterations of) the argument string variables may be made freely.

The procedure call for PROC can resemble the examples given previously for external commands, except that there is no prompting. Again, letting [i] stand for a user-supplied value for argument Ai, the lines below show user input and the string values used by UMCR:

input	values for A1, A2, A3, A4
PROC	<1>,<2>,<3>,<4>
PROC A3=[3]	<1>,<2>,[3],<4>
PROC [1],[3]	[1],<2>,[3],<4>
PROC A3=[3],[4]	<1>,<2>,[3],[4]

The ability to run long sequences of UMCR commands unattended is assisted by UMCR control commands: IF-THEN-ELSE, WHILE-DO, FOR. The numbers on which these commands operate may come from the procedure file itself, from user input (an ASK command), or from the task error code returned by an exiting task. This last item makes it possible to alter the course of a procedure file if something unusual happens during data acquisition or analysis.

All of the syntactic processing by the CI has been implemented using a modified version of the MARGOT command language interpreter generator (developed by M.J. Myszewski and P.M. Cashman of COMPASS; see Proc. 1977 Spring DECUS, pp. 1131-1144). MARGOT was originally developed to support another project at COMPASS, and the UMCR system represents its third usage by us. The MARGOT package provides a convenient conceptual framework for thinking of syntactic structures at a BNF-like level. Our experiences reinforce the earlier favorable comments: the MARGOT package greatly simplified syntactic processing for us. (The original MARGOT is available as DECUS 11-322.)

CONCLUSIONS

Once we overcame the technical hurdles, we found RSX11M to be a very satisfactory base on which to build the UMCR system. Despite our modifications to it, RSX11M has been reliable and trouble free. We use the same modified system for program development and UMCR task check-out, so we feel certain that the modifications are indeed transparent to standard RSX11M software.

The technique of using a small privileged task neatly encapsulates the function of spawning other tasks and has worked reliably. The task was simple enough to be debugged in less than a day. The overhead incurred by having UINS run MCR and INStall has been unobjectionable.

The modifications to the RSX11M Executive proved to be quite small: 42 bytes. Changes to the terminal driver required 132 bytes, or about 4% of the size of the driver we use. We consider these costs very small when compared with the cost of obtaining equivalent functionality any other way.

The command interpreter has been used extensively during program development of the analysis tasks, but so far few "real" users have had access to it. Initial experience indicates that there will probably be a "learning curve" during which users will gradually acquaint themselves with its full power.

ACKNOWLEDGEMENTS

The author wishes to thank Robert M. Supnik and Nancy Kronenberg for their help in the early part of this project. Their investigations exposed the problem areas in RSX11M, and they proposed general solutions to them.

THE DEC FORTRAN ENVIRONMENT FOR BUSINESS APPLICATIONS

David J. Hirschfeld
Business Controls Corporation
Elmwood Park, New Jersey

ABSTRACT

Many DEC users of PDP11 systems are in the FORTRAN environment running applications other than those usually classified as Business Applications. These users often turn elsewhere to accomplish the processing of their business information never realizing the tremendous potential inherent in their PDP11 hardware and their FORTRAN software for the fulfillment of business data processing needs. With awareness of this potential many DEC users could implement effective Business Systems running alongside their other applications and do it far more economically than with any other alternative.

Many DEC users of PDP11 systems are in the FORTRAN environment and are running applications other than those usually classified as Business Applications (Order Entry, Billing, Sales Analysis, Inventory Control, Materials Requirements Planning, Job Costing, Accounts Receivable and Payable, Payroll, List Maintenance, General Ledger and other similar applications).

Many of these users turn to other sources such as service bureaus, time-sharing, other manufacturers and even non-compatible DEC systems to meet their needs for Business information processing. A conscious awareness of the excellent potential their existing computer environment has for the fulfillment of Business data processing needs opens up a new and very attractive alternative.

The purpose of this paper is to alert the DEC user to the tremendous potential within his DEC FORTRAN environment for the effective and economical fulfillment of his Business data processing needs.

In an article published in the April '78 issue of Datamation called "Comparing Computer Language Performance", Mr. Jerome W. Blaylock reports on the results of a study conducted at the University of Houston Computing Center comparing the performance of COBOL and FORTRAN in a Business Environment. The tests were done on a large Byte oriented machine which favors the Byte oriented COBOL language over the Word oriented FORTRAN language which would fare even better on a Word machine like the PDP11. Mr. Blaylock is quoted as saying "Our experiments indicate that FORTRAN is - or at least can be - effective compared to COBOL".

When considering DEC FORTRAN under RSX11M on a PDP11 computer, one might well expect an even more enthusiastic endorsement of the FORTRAN environment for Business Applications. Here are some of the significant reasons why this conclusion can be expected.

Reliability

One of the major considerations of anybody thinking of putting the key operating data of their business on a computer is the reliability of the software tools they use. Nothing can set you back harder than a bug in the manufacturer's software that affects your application and is completely out of your control to correct. A good rule of thumb is to allow manufacturer's software a three year period of field experience and debugging to achieve an acceptable level of reliability. Both DEC's RSX11M operating system and their FORTRAN IV compiler pass this test and have proven their reliability in thousands of installations. The number and type of corrections coming down from Maynard at this time are relatively small and deal with problems remote enough from the main stream of business data processing functions as to allow for a very high level of confidence by the Business System Planner.

Multi-Programming Capability

The flexibility required by a user oriented, interactive Business Information System and by a Company wishing to mix Business and non-Business applications requires a high degree of multi-programming capability. The RSX11M operating system offers the user outstanding flexibility for multi-programming, the only practical limits being the amount of core available and the amount of core required by individual application programs.

Score another point for the DEC FORTRAN.

Our experience with this environment in implementing Business data processing systems for many Companies has shown the following average core requirements:

- .Operating System.....32K Bytes
- .Re-entrant FORTRAN Res. Libr..24K Bytes
- .Each Application Program.....12K Bytes

Using our experience, it can be shown that even with a PDP1134 having a capacity of 256K bytes core memory, the user can have up to 16 Business Application programs running simultaneously.

Combine this capability with printer spooling available with RSX11M and you have one powerful, interactive, user oriented Business data processing system.

Efficiency

The efficient use of system resources by FORTRAN is demonstrated dramatically above. There is more, however, to be said about the efficiency of FORTRAN operating in a Word machine as most minicomputers are. For example, our experience has shown that the average compilation time for a FORTRAN Business Application program was under 30 seconds exclusive of printing source listings. With the spooling capability of RSX11M, each programmer can effectively start his next compilation after 30 seconds. Execution of application code is also highly efficient since the compiler's object code conforms to the hardware's word oriented architecture.

Compatibility with Other Applications

Present applications in FORTRAN, BASIC and MACRO 11 can run side by side with your FORTRAN Business Application programs. Each user is protected with his own UIC. Even Word Processing applications can run concurrently with your FORTRAN programs. Some users that have PDP11's as a real-time part of their production operation such as Typesetters for example, need to have back-up hardware on-site that is almost always idle. Why not use it for Business data processing, or conversely, have a Business system that can back-up your present system and vice-versa.

Availability of FORTRAN Programmers

Aside from the fact that a DEC user in the FORTRAN environment probably has some FORTRAN expertise in-house, there are more FORTRAN programmers in the world, than any other language. It is also alot easier to learn FORTRAN than it is to learn COBOL for example.

Availability of Re-entrant Code

With DEC's FORTRAN IV plus, the user can write application code that is completely re-entrant. Essentially, this means that more than one terminal or user can share one core copy of a program. In certain business applications this is an extremely important capability. Consider, for example, a large wholesale distribution operation where any number of terminals may be needed to inquire into the status of inventory and enter a customer's order. If each terminal had to have its own program, the maximum number of terminals in a 256K byte machine would be about 16. With re-entrant code, however, one 12K byte program could service any number of terminals, each with its own destructable data buffer.

Availability of Proven Application Packages

One of the most attractive advantages of the FORTRAN environment for Business data processing is the availability of proven application packages.

Taking advantage of the existing high quality application packages available for PDP11's in the FORTRAN environment brings big and immediate benefits to the PDP11 user.

1. Avoiding the cost of development and the thinning of usually already scarce programming resources.
2. Proven packages are clean, debugged and standardized.
3. Proven packages are accurately and professionally documented.
4. Professionally developed packages usually represent latest state of the art status. More money is spent on developing these packages than any one user could justify.
5. The user can be up and running quickly and inexpensively, thereby gaining the confidence of top management as well as their support for further excursions into Business Data Processing.
6. Packages developed for RSX11M and FORTRAN represent little difficulty for the existing staff to pick-up, operate and even modify.

These types of professionally developed packages are available and adaptable for most Companies in the most standardized areas of Business data processing.

- .Accounts Receivable
- .Accounts Payable
- .Payroll
- .General Ledger
- .Bill of Materials
- .Materials Requirements Planning
- .List Maintenance

Some areas of Business Data Processing almost always require customizing to the unique characteristics of a particular business firm. Applications like Order Entry, Invoicing, Sales Analysis and Commissions usually reflect the unique personality of a firm and to try and change the successful traits of a Company as it interfaces with its customers to fit into a package is flirting with disaster.

Most firms that offer Business application packages also offer customized design and turnkey software services. It is often wise to use their experience with these applications rather than to dilute your non-business oriented data processing staff with these applications. The most important thing, however, is for the PDP11 user to be consciously aware of the inherent potential existing in his FORTRAN system for a good solid Business Information System.

SIZING AND PLANNING

A DECNET NETWORK

Richard Pigman and William Lahtinen
Digital Equipment Corporation
Maynard, Massachusetts

ABSTRACT

This paper outlines a set of planning requirements that must be addressed in implementing a network. These requirements center on the following tasks:

1. Configuring a network that will accommodate projected growth. (The term "Network Sizing" is frequently used to describe this activity.)
2. Planning for network installation and operation.

SIZING THE NETWORK

In order to size a network, application and system requirements must be defined -- application requirements first, and then system requirements. Once application requirements relative to functional distribution, input/output characteristics, and growth forecasts are established, development and training can begin, system needs can be identified, and the network sizing task addressed.

Planning for network installation and operation can begin before the order with Digital is signed -- certainly no later.

NETWORK SIZING -- An Example

Because application and system needs are unique to specific situations, this paper will use a sample installation in order to illustrate how the planning concepts for network sizing are applied. We will work with a sample order entry system that has the following application and system requirements:

Application Requirements -- Example

- A central location will handle network management, data base control, and program control.
- There are six remote locations with the following characteristics: the users are non-data processing type personnel; the transactions are update and query; the system will be in operation eight hours a day (9-5).
- Business activity (and thus, transaction volume) will increase over the next three years.

System Requirements -- Example

- The topology is defined as consisting of a central node and six remote nodes. (See Figure 1.)
- The Operating System used at each node is known.
- CPU requirements for each node are known. (See Figure 1.)
- All system transaction types are defined.
- Forecasts of transaction volumes are available for the next three years. (See Figure 2.)

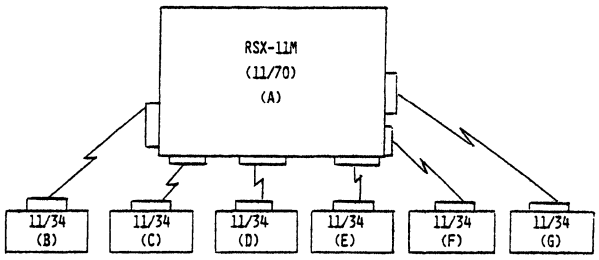


Figure 1 -- Network Configuration -- Example

SYSTEM NEEDS

DAILY VOLUMES (TRANSACTIONS/DAYS BETWEEN 11/70 AND 11/34)

	YEAR 1	YEAR 2	YEAR 3
B	10.000	17.000	25.000
C	8.000	14.000	20.000
D	4.000	11.000	15.000
E	-	4.000	10.000
F	-	4.000	10.000
G	-	3.000	10.000
TOTAL	22.000	53.000	90.000

Figure 2 -- Transaction Volume Forecast -- Example

Factors still to be determined are: the communications interfaces required for each link, and the speed of the communications lines. Also, we want to estimate the CPU utilization required for

communications.

DEFINING COMMUNICATIONS NEEDS: Interfaces, Line Speeds, Processing

In order to select the proper communications interface and correct line speed, you must have accurate data on message traffic for the network links, and applications throughput requirements. Once the message traffic for the application is projected, CPU usage can be estimated, and minimum communications line speed requirements can be determined.

Typically, you would have a forecast of the number of daily transactions each node is expected to process, rather than message traffic for the network. Figure 2 shows the transaction forecast for each of the 11/34 nodes (B through G) in our order entry example. We must examine each link in the network, looking at transaction volume, CPU utilization, and line speed. We came up with our expected transaction figures when we defined our application needs. Now we must estimate CPU utilization.

There are several factors which impact CPU utilization. Some of these are defined by the application, others are user-determined.

The CPU utilization factors defined by the application are:

- Number of Messages/Second.
- Size of Message (characters).

User-determined factors are:

- Communications Interface (character interrupt vs. DMC11).
- CPU (11/70 or 11/34).

Number of messages per second is derived from transaction volume, which will be discussed later. The size of the messages is determined by transaction type, which is already known from our application definition. In our example, the CPUs are already given, but if they were not, processing requirements could be established through evaluation of the other three factors. CPU selection is determined by the amount of processing power the user wants to dedicate to the network.

It is important to note that the communications interface selected affects CPU utilization. This again, as we will see, reflects a choice by the user of the CPU utilization he will tolerate.

The factors that must be considered in establishing required line speed are also defined by the applications and by user requirements.

The factors defined by the application are:

- Number of Messages/Second.
- Size of Messages (characters).

User-determined factors are:

- Full duplex (FDX) vs. half duplex (HDX).
- Error rate of the communications line.
- Synchronous or asynchronous transmission.

As with CPU utilization, the message rate and size defined by the applications is a determining factor in specifying the required line speed.

Now let us look at some tools that will help us determine how much CPU we need for the network, and what lines can best satisfy our requirements. These tools consist of several graphs and tables (Figures 3 through 9) illustrating some of the information gathered on the DECnet Phase II product performance. We will be using these graphs to aid in selecting the line speed and communications interface for our order entry system. The graphs will also allow us to estimate CPU utilization. All of our branch offices have this information. Your local salesman and Software Services representative can assist you in sizing your network.

The data we will be using is for DECnet-11M V2.0.

The first graph (Figure 3) represents the net line utilization, versus message size, by line speed. The graph depicts what the user application would see as net utilization, depending on message size and line speed. For example: the graph shows that a 256-character message, running on a line rate at 9.6K results in a net line utilization of 70%. These figures are for the DMC11/FDX communications device.

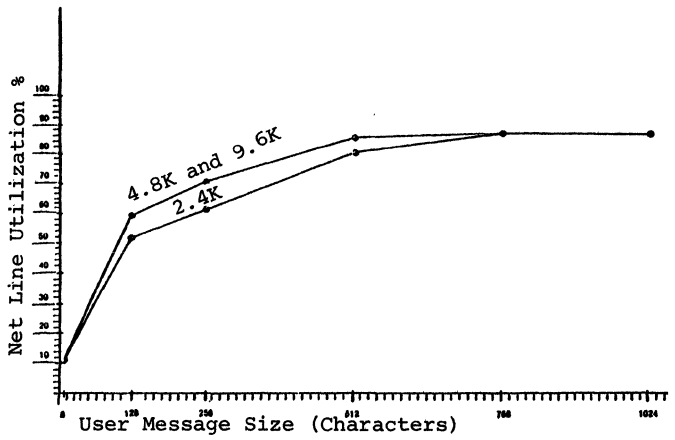


Figure 3 -- Net Line Utilization vs. Message Size

The next graph (Figure 4) represents the CPU utilization by processor versus user message rate. For example: this graph shows that 10 messages/second uses less than 20% of an 11/34. For the DMC11, it has been found that message size for message lengths between 8 and 1024 characters has a negligible effect on CPU usage. Therefore, this one graph is applicable for message lengths between 8 and 1024 characters.

For the DUP11, a character interrupt device, the message length does have a significant impact on CPU usage. Figure 5 shows the CPU usage for DECnet-11M V2.0, on a 11/34 with a DUP11 interface, for various length messages at various message rates. Figure 6 shows the same data for a 11/70.

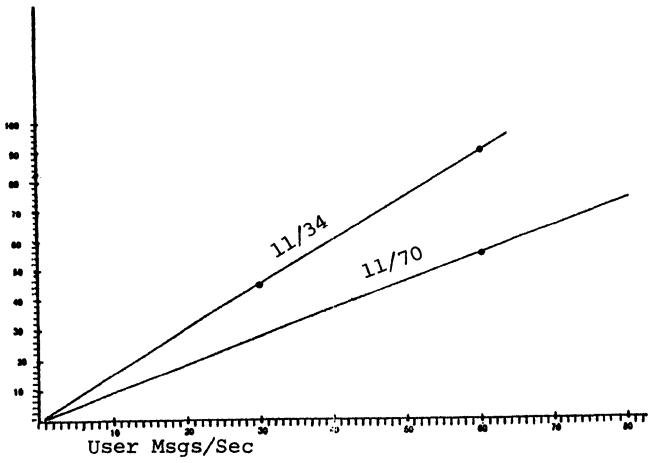


Figure 4 -- CPU Usage vs. Message Rate

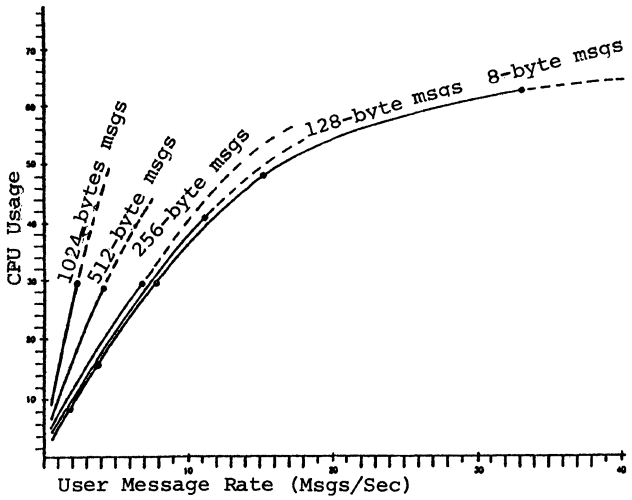


Figure 5 -- CPU Usage vs. Message Rate for 11/34

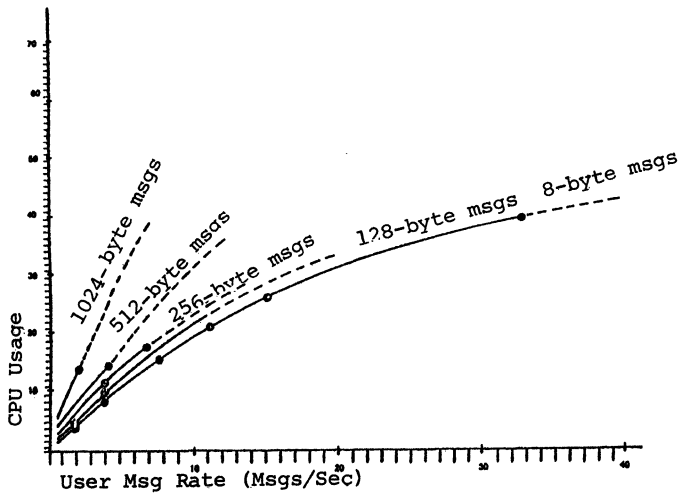


Figure 6 -- CPU Usage vs. Message Rate for 11/70

In order to determine CPU utilization and communications interface, we need to know the minimum user line speed, the net line speed the user application sees, the message rate, and message size required by our application. The minimum user line speed is a percentage of the minimum real communications line speed. The concept can be expressed as follows: Net Line Utilization is user line speed over real line speed (expressed as a percentage). The same factors which influence real line speed affect line utilization.

We are now ready to analyze our data and determine the line speed and communications interface for our order entry system.

First, we need to translate our transaction rate per day into peak hour requirements. We want to configure the network to handle the peak hour network traffic.

The peak hour load can be determined either by knowing the application or by estimating the peak load using empirical data.

For our order entry example, the peak hour load is assumed to be 17% of the daily volume (See Figure 7). As an example, take Node B in our network. In year 3 the node will process 25000 transactions a day. The peak hour load then is:

Peak load = (.17) (25000) = 4250 transaction/hour.

Averaging the load over the hour, we get transactions per second.

$$\text{Transactions/second} = \frac{4250}{3600} = 1.2$$

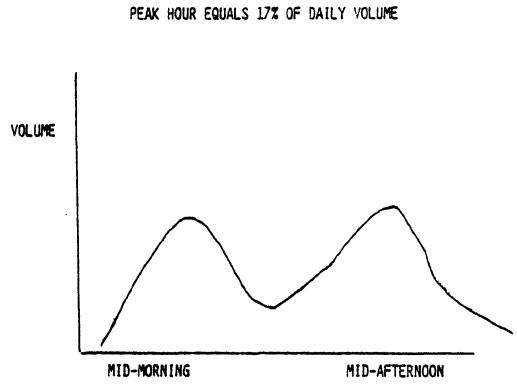


Figure 7 - Peak Load Pattern

We now must translate transactions per second into network traffic (messages/second). This translation is application specific. For each application transaction, how many network user messages are generated? For our order entry example, we will assume the following:

- One message to the 11/70 -- average size = 128 characters.
- Average of four messages back to the 11/34 for each message to the 11/70 -- average size of each = 128 characters.

Therefore, for each transaction, there are an average of 5 messages to and from the 11/34 (Node B). The peak load averaged 1.2 transactions a second; therefore, the peak message rate for Node B is:

Messages/second = (1.2 transactions/second) (5 messages/transactions) = 6

In year 3, Node B will have a peak message rate of 6 messages/second.

Similarly, we can calculate the peak message rates for each node in the network. The peak message rate for the 11/70 is the sum of all the message rates for the 11/34 nodes.

Now that the message rates are known, we can estimate CPU utilization from the graphs. For example, our message rate for Node B is 6 messages/second, 128 characters a message. Using a full duplex DMC11, the CPU utilization (Figure 4) would be 9%. If we use a full duplex DUP11, the CPU utilization, (Figure 5) would be 24%. Figure 8 shows the message rates and CPU utilization using a full duplex DMC11 interface for all the nodes in the network.

	MESSAGES PER SECOND (YEAR 3)	CPU UTILIZATION (FROM DMC11 CHART)
11/34(B)	6.0	9%
11/34(C)	4.8	6%
11/34(D)	3.6	5%
11/34(E)	2.4	4%
11/34(F)	2.4	4%
11/34(G)	2.4	4%
11/70(TOTAL)	21.6	20%

NOTE: CPU UTILIZATION DOES NOT INCLUDE APPLICATION PROCESSING OR OTHER BACKGROUND PROCESSING.

Figure 8 -- Message Rate/CPU Utilization Table

It is important to note that CPU utilization does not include application processing, or other background processing. Also, since the DMC11 performs all DDCMP functions, CPU utilization is not affected by the communications error rate, as it would be by character-interrupt interfaces. Network throughput, however, is affected by the error rate, even with DMC11 interfaces.

Assuming we want to minimize the amount of CPU devoted to the network, we select the DMC11 as our communications interface.

Now we want to select the line speed. Again, taking the link between Node A, the 11/70 and Node B, 11/34, we will determine the minimum line speed required. If we use a full duplex DMC11 link between the nodes, we will need a line speed capable of handling the greatest message rate in a given direction. It can be determined that the message rate to Node A from Node B (to the 11/70) is 1.2 per second at 128 characters per message. This is derived as follows: 1.2 (transactions/second x 1 (messages per transaction). The message rate to Node B from Node A (to the 11/34) is 4.8 messages/second at 128 characters per message. This is derived as follows: 1.2 (transactions/second x 4 (messages per transaction). Therefore, the user line must be rated to handle the higher message rate into the Node B (to the 11/34).

- Net bits per second - (message/rate) (characters/message) (bits/character).
- Net bits per second = (4.8) (128) (8) = 4916 BPS.

As previously pointed out, the user line speed (Net BPS) is equal to the net line utilization times the real line speed.

- Net BPS = (Net line utilizations) (Line speed).
- Line speed - Net BPS/Net line utilization.

We can determine from Figure 3, that with a message size of 128 characters, net line utilization of 60% is realized. Therefore, the minimum required line speed of the link between Node A and Node B is:

- Line speed = 4916/.60 = 8193 BPS.

Therefore, a 9600 baud line which can handle 8193 bps, is required for the link. This line speed does not include provisions for:

- Error rate of the communications line.
- Queuing delays for line traffic (response time sensitive).

Similar calculations will determine the line speeds required for the other links. Assuming all links are full duplex DMC11 links, the CPU utilization and necessary line speed for all the remote nodes is shown in Figure 9.

	CPU UTILIZATION	LINE SPEED
B	9%	9600
C	6%	7200
D	5%	7200
E	4%	4800
F	4%	4800
G	4%	4800

Figure 9 -- CPU Utilization/Line Speed Table

A clear statement of the application needs, particularly message rate and message size, are essential to size a network. Again, the validity of your plan depends on the accuracy of your data.

Now that you have sized your network, you are ready to sign the order with DEC, and go on to the next planning activities - installation and operation.

PLANNING FOR INSTALLATION AND OPERATION

A word of caution! After placing the order, there is a tendency to sit back and wait for the equipment to arrive. This is dangerous! It is important that planning for installation and operation begin as soon as possible. In some cases, planning should have begun while application and system needs were being evaluated.

Several factors contribute to the necessity for planning.

1. It is quite likely that the network will consist of components from several vendors.

2. There are frequently different operating systems involved in a network.

3. There is usually geographic dispersion to consider. All nodes are not located in the same room. Node-to-node communication can span continents.

Any one of these factors can cause special consideration to arise. When you take all of them together, the problems to be handled are formidable. In addition, from a project control point of view, the sheer magnitude of the installation task can be awesome.

KEY ELEMENTS IN PLANNING

First, and most important, is the appointment of a network manager. The network manager function can be exercised by one person, a part of a person, or an entire department, depending on network size. The appointment of a network manager should be made as early as practical in the planning process. In fact, it can, in many cases, be made when identifying application system needs or while the network is being sized. The network manager has several functions. He or she serves as an interface with DEC and with other vendors, develops the schedules, procedures, and documentation, and identifies training requirements. These are discussed in turn below.

Developing Schedules

The network manager should first address the problem of scheduling. Input to this task consists of corporate requirements (by when must the network be installed, for example), equipment and personnel availability for installation and operation. The development of procedures, the preparation of documentation for training should also be addressed.

In general, scheduling should specify tasks, individuals or groups responsible for each, and indicate by when each task must be completed. It is desirable to schedule the ordering and delivery of all components. (Don't forget lines and modems. For some reason, these are often the last thing anybody thinks of.) Scheduling should also cover installation sequence, training, and deadlines for the preparation of procedures and documentation.

Developing Procedures

Don't wait until after the network is installed to develop procedures. Procedures should be prepared ahead of time. People who are going to be involved in installing and operating the network should be trained in their particular responsibilities in advance, should understand what these responsibilities are, how they are carried out, and know where the documentation is located. Installation procedures are useful both in installing the original network and in adding nodes later. Other procedures (backup/fallback, testing-test equipment, preventative maintenance, fault isolation, and escalation) are used mainly in operating the network. It is good practice to check the soundness of backup/fallback procedures at installation -- before a crisis situation develops.

Installation

Installation procedures are among the first to be developed. When installing a network, remember two concepts: phasing and staging. Phasing means installing the nodes in sequence or in phases -- don't try to install the entire network at once. Staging is recommended if networks are new to you. Staging means shipping communicating nodes to one location and developing the applications before separating the nodes. This approach can save a lot of telephone time and possibly much travel time.

Backup

Any critical communications link in a network is a weak link -- unless backup procedures are available to assure continued function in the event of primary link failure. Availability of a Chicago-New York line, for example, could be so critical that it would be wise to have a dial backup connection or a second line available. Availability of the backup, however, is not enough: operating personnel must know how to use it. Testing procedures for testing the network and for using test equipment are also necessary. Routine testing in off-hours can spot gradual communications line degradation and remedy the condition before failures occur. Many users who operate large networks have found it wise to configure testing and fallback equipment and procedures into a "Technical Control Center" or a "Network Management Center" and testing is performed on a routine basis. Since these facilities are often operated by relatively unskilled personnel (on the third shift) well defined procedures are essential.

Maintenance

Preventive maintenance is not too mundane a concern in a network environment. Remember that PM's are necessary for network system availability -- and they must be scheduled so as not to interfere with network operations.

Fault Isolation

The time to develop fault isolation procedures is before something has failed. In a failure situation, operations personnel are often under intense pressure and do not have time to learn how to operate test equipment or test the system. Procedures must be developed, and personnel trained to use them, before crisis situations arise. Routine testing, as discussed above, provides a good environment for training personnel in the use of fault isolation procedures.

Problem Escalation

Because of the complexity of networks and the likelihood that multiple vendors will be involved, there is a high probability that situations will arise where it will be necessary to escalate a problem, both in the user and vendor organizations. Procedures for doing this should be developed so as to bring a problem to the attention of those most competent to deal with it.

Training

In addition to developing schedules and procedures,

the network manager coordinates vendor and internal training, based on procedures developed for operating the network.

Documentation

The documentation function records the schedules and all the procedures discussed above. A procedure for updating the documentation should also be developed to make sure that the entire staff is working with current information. Other necessary documentation includes the packaging of internal training in a useful format, and the provision of any reference material that might be useful to the people operating the network.

A node map should be developed. A node map is a document which shows where all the nodes in the network are located, identifies the hardware and software configuration of each, and names the individual or group responsible for its operation.

It is also advisable to prepare a call list, based on the escalation procedures noted earlier, so that if problems do occur, operating personnel at each node know what procedure to follow. All documentation should be available at every node in the network.

How DIGITAL Can Help

There are several ways that DIGITAL can help a customer in planning installation and operation of his network:

- The Customer Support Plan
- Training Courses and Seminars
- Software Testing Aids
- Site Preparation Guide
- Site Management Guide
- Consulting Service

Customer Support Plan

The Customer Support Plan is a requirement for all network sales proposals. It is prepared by DIGITAL as part of the pre-sales process and provides information such as: (1) account background, (2) major applications, (3) customer staffing and skills, (4) service recommended to the customer, (5) the delivery plan of equipment, (6) an alternation mechanism -- that is, how does DIGITAL interact with the customer to change the plan, and (7) who is the DIGITAL representative involved in the process.

Training

Training courses and seminars -- DIGITAL provides a number of lecture-type and audio visual courses designed for customers getting ready to operate their own networks. Our catalog, available through Educational Services, lists and describes all of the courses available. DIGITAL's consulting services, which is discussed later, will develop customized seminars to meet the special needs of individual customers.

Testing Aids

Software testing aids include line and node counters, software loopbacks, and various operator controls. These are standard in most operating systems. Procedures for using them and the information they provide are the same for every operating system. There are, however, some individual differences.

The list of testing aids is fairly lengthy but it is important to know that there are counters which record line and node events, including retransmissions, extraneous messages, blocks received, blocks sent, connects initiated, connects received. A full list is covered in the System Manager's Guide for each operating system. Commands that turn lines on and off and which list counts of line events are also discussed in detail.

Site Preparation

The Site Preparation Guide, available from DIGITAL, covers considerations that should be exercised in the preparation of the site for installing a node. It covers site planning, acoustics, vibrations, lighting, cleanliness, electrical considerations, grounding, electromagnetic interference, temperature, humidity, etc. The Guide also includes a handy site preparedness checklist which summarizes all the considerations on a single sheet of paper.

Site Management

The Site Management Guide contains general and specific information relative to each line site. It is furnished on a per-system basis to users who buy field service contracts. It contains worksheets such as the customer activity log, PM log, configuration worksheets, problem reporting, trouble log, etc. It is also a handy place to put the site documentation discussed above.

Consulting Services

Consulting Services are available from our Software Services Organization. They are referred to as Level I services and Level II services. Level I services cover the installation of software products in a network and a demonstration of network functionality. Level II services -- and these are only examples -- cover anything the customer wants that we can provide. This can include, for example, planning and scheduling a staged installation, conducting seminars, reviewing network utilization and performance, and development of application software. These services are provided for an additional fee. For more details, consult your nearest DIGITAL office.

Summary

Most of the information contained in this paper is based on common sense. Experience has taught us (and others) that neglect of any of the factors discussed, will make network installation and operation more difficult, lengthier, and more expensive.

A SOFTWARE DEVELOPMENT SYSTEM FOR SMALL DEDICATED AND FRONT-END MICROCOMPUTER (LSI-11) APPLICATIONS*

J. W. Tippie and P. E. Rynes
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439

ABSTRACT

The development of software for small dedicated microcomputer applications is frequently impeded or complicated by lack of adequate peripherals, utilities, and operating systems. In this paper we shall describe a software development system for the LSI-11 microcomputer, some of its anticipated applications, and some of the software problems. The term "small dedicated application" is used here to refer to one-, or few-of-a-kind systems based on the LSI-11 that lack a device suitable to support an operating system, and that are not subject to frequent software changes.

INTRODUCTION

In an environment heavily committed to PDP-11 family minicomputer applications, the LSI-11 is very attractive for small dedicated applications because of its readily available software support, familiar hardware architecture, and ease of transferring files (software and data) between larger minicomputers and the microcomputer. The problem that remains to be solved is providing an adequate software development facility for small dedicated applications. Software development is frequently hindered by lack of adequate support facilities such as peripherals, software utilities, and operating system environment. Typical microcomputer development systems costing $20,000 or more are available, but suffer from being single user, and dedicated to software development for some particular microprocessor family.

Software development (be it for minicomputer or microcomputer) involves considerable time at low resource utilization (text editing), with short bursts of high-resource utilization (compile, assemble, task build). It further requires access to such resources as large direct access storage (for development source code files, libraries, and data), terminals (local and remote), line printers (for listings), and file transfer media such as floppy disk and magnetic tape. The peripheral investment is high for both minicomputer and microcomputer software development, but not providing the support hardware is even more costly in terms of reduced programmer efficiency. Thus a multiuser environment based on a larger minicomputer running RSX-11/M with adequate peripherals is ideal for development of software both for minicomputers (PDP-11 family), and microcomputers (LSI-11).

*Work performed under the auspices of the U.S. Department of Energy.

The system shown in Figure 1 has evolved at Argonne to support minicomputer software development. The remainder of this paper describes some of the software tools, problems and applications specific to the support of the LSI-11 microcomputer in this environment.

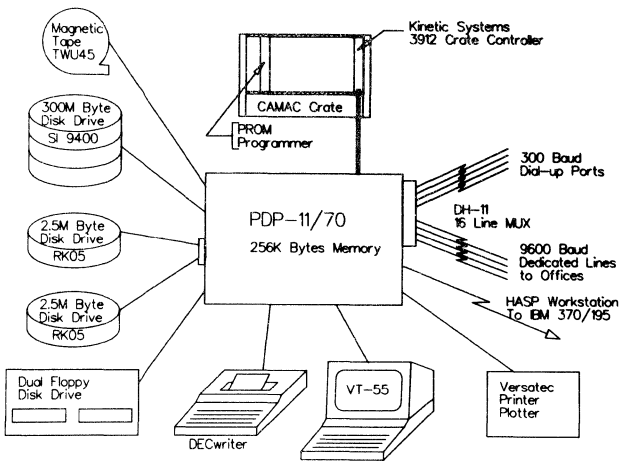


Figure 1. PDP-11/70 Software Development System.

Software Considerations

For reasons discussed earlier the multiuser environment as provided by RSX-11/M (or another multiuser operating system) is desirable because it permits maximum use of available resources. The RSX-11 environment provides the necessary file structure, editors, assembler, task builder, and other utilities. In an ideal situation the software developer wishes to write his application (preferably in a higher level language), edit the source code, compile it, link or task build it with library support, and test it (see Figure 2).

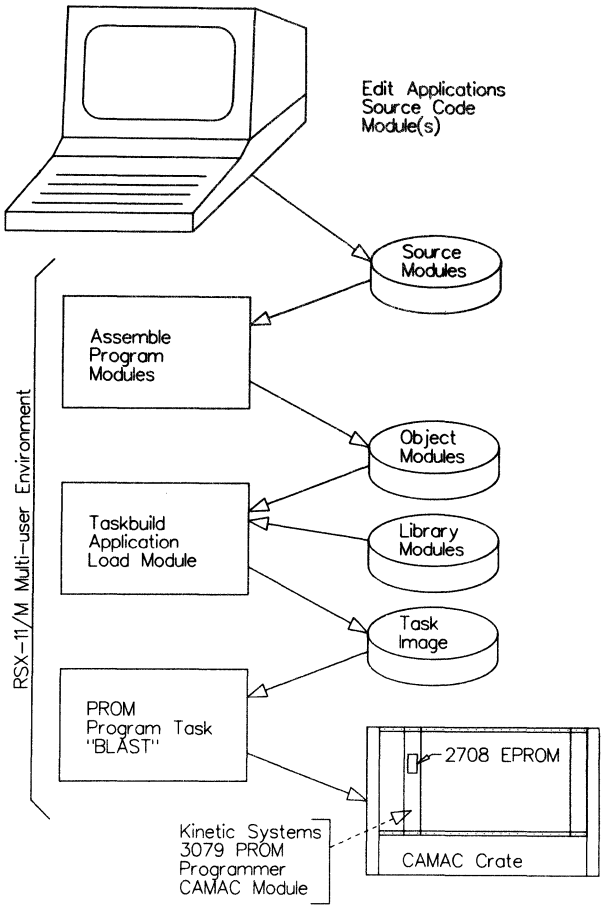


Figure 2. Software Development Sequence.

The standard RSX-11/M environment supports the functions of edit, assemble, and task build of a core image file, and provides a rich collection of utilities. It does not support (as standard) a higher level language that generates code suitable for execution in ROM (Read Only Memory) without operating system support. Neither does it support (as standard) utilities to program PROMs or down-line load task images of developed software.

These utilities (to be discussed below), coupled with some of the MACRO "Languages" such as SUPERMAC[1] or BIOMAC[2] would enable the user to develop dedicated LSI-11 software in a relatively efficient manner. The major utility that remains to be developed for use with RSX-11/M is a higher level language that runs under RSX, but generates code and has an object time library suitable for execution in an environment with no operating system, and that separates read-write segments from read only segments.

Application Task Organization

Applications requiring the program to reside in read only memory must be divided into pure (read only) and impure (read/write) segments. Only the pure segment can be programmed into PROM. The impure segment is treated as a virtual program section and is restricted to contain only global references, and memory allocation directives (.BLKW, .BLKB, etc.). The task builder permits each segment (pure and impure) to be made up of several program sections that can be concatenated or overlayed as desired. For example:

```
.PSECT PURE,RO,LCL,REL,CON
(pure program module)
.PSECT IMPURE,RW,LCL,REL,CON
(impure program module)
```

Note that impure .PSECTs (VSECTs) must either carry the same name or be allocated individually at task build. The following example illustrates the task build options to generate a file suitable for programming into PROM.

```
>TKB
TKB>PTASK/-MM/-HD/SQ,LP:=OBJ1,OBJ2...
TKB>/
ENTER OPTIONS:
TKB>STACK=0
TKB>PAR=GEN:base:window
TKB>VSECT=IMPURE:base:window
TKB>//
```

To produce a clean memory image file the stack=0, no memory management, and no header options are required. The task image file and corresponding memory organization of the target machine are illustrated in Figure 3.

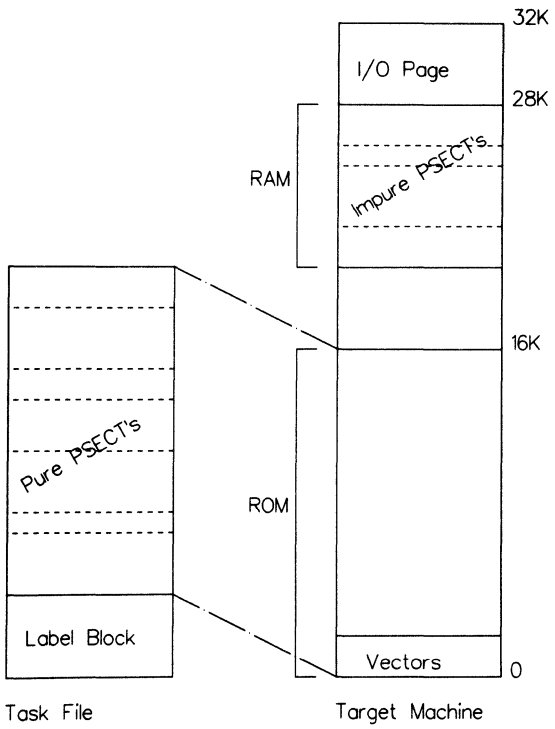


Figure 3. Task Image File and Memory Organization.

PROM Programming

The hardware to support programming of 2708 and 2716 EPROMs is illustrated in Figures 1 and 2. The Kinetic Systems 3079 CAMAC PROM programmer was chosen because of the existing CAMAC hardware and RSX-11/M CAMAC driver.[3] With this hardware it is very easy to program PROMs and verify the information in a programmed PROM in a multiuser environment.

A utility program, BLAST, was developed to program PROMs from either task image files or data files under RSX-11/M. PROMs can be programmed either sequentially or alternating high byte-low byte in separate PROMs (as for a task image).

Downline Loading

To complement the PROM programming utility, a down line loading utility is needed to load tasks into dedicated processors and to test software to be programmed into PROM. The most flexible method of implementing a down line load facility is to use the asynchronous dedicated or dial up parts of the host system. If binary data is encoded into the printable character set, the standard host computer terminal driver can be used.

The down line load bootstrap is implemented in PROM. A "type-through" mode is provided to enable the user to initiate the RSX-11/M down line load task. A special escape character (CTRL/A) is used to initiate the transfer in the host and the down line load bootstrap code in the LSI-11.

In the initial implementation, binary data is converted to HEX-ASCII (0-9, A-F) with two characters per byte, and a six bit representation is being developed. Messages of the following format are used for processor to processor communication:

protocol ID
message type
sequence number
length
•
•
•
information
•
•
•
check sum

The information field for down line loading includes a load address and length plus data.

Applications

Several applications using these techniques are under development. The first is a dedicated processor to control a large scattering chamber. This application will control several detector-bearing arms that can be moved both radially and azimuthally. The processor must display the current angles and detector radii and accept commands from the physicist/operator to position the detectors without crossing arms, running the detectors into the beam, or violating assorted other physical limitations. The controller must also accept control information from a larger computer system when appropriate (see Figure 4).

Another application under development is to use LSI-11s as "intelligent crate controllers" in a large distributed serial CAMAC system. The "intelligent crate controllers" will be down line loaded with tasks and

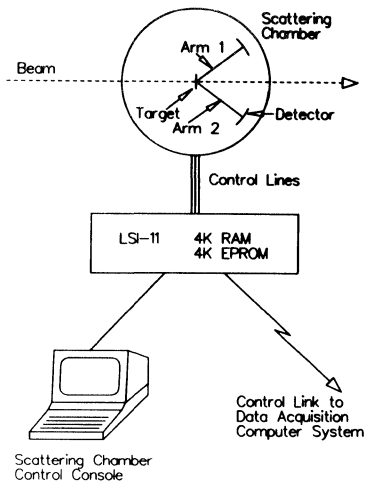


Figure 4. Scattering Chamber Control System.

control profiles to cycle electrical storage batteries through loads simulating those encountered in a battery powered vehicle (see Figure 5). Summary data collected from the batteries will be sent back to the main computer and recorded.

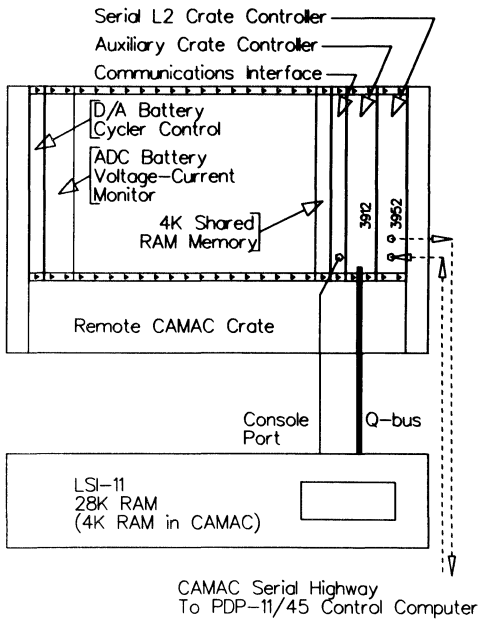


Figure 5. NBTL Remote Battery Controller.

Conclusions

Development of software for dedicated LSI-11 based applications in the RSX-11/M multiuser environment is very effective. It represents a better use of resources than RT-11 or some of the other microcomputer development systems. Problems with higher level languages and separation of read-write segments from read only segments remain to be overcome. Clearly, DEC

should be urged to address these problems, as the resulting development system could easily become one of the most powerful in the industry.

REFERENCES

1. SUPERMAC available through DECUS Library.

2. G. S. Herman Giddens et al., *BIOMAC: Block Structural Programming Using PDP-11 Assembler Language*, Software: Practice and Experience, Vol. 5 (1975).

3. J. W. Tippie, P. H. Cannon, *CAMAC Driver for the RSX-11M V3 Operating System*, Proceedings of the Digital Equipment Computer Users Society, Vol 4, No. 2, P563 (1977).

ACKNOWLEDGEMENT

The authors wish to acknowledge the efforts of S. Alford, a participant in the Argonne Summer 1978 Undergraduate Research Program, who implemented much of the PDP-11/70 code.

ACCELERATOR CONTROL USING RSX-11M AND CAMAC

Joseph E. Kulaga
Physics Division, Argonne National Laboratory
Argonne, Illinois 60439*

ABSTRACT

This paper describes a computer-control system for a superconducting linear accelerator currently under development at Argonne National Laboratory. RSX-11M V3.1 running on a PDP 11/34 is used with CAMAC hardware to fully control 22 active beam-line elements and monitor critical accelerator conditions such as temperature, vacuum, and beam characteristics. This paper contrasts the use of an RSX compatible CAMAC driver for most CAMAC I/O operations and the use of the Connect-to-Interrupt Vector directive for fast ADC operation. The usage of table-driven software to achieve hardware configuration independence is discussed, along with the design considerations of the software interface between a human operator and a computer-control system featuring multi-function computer-readable control knobs and computer-writable displays which make up the operator's control console.

INTRODUCTION

Accelerator Components

A joint effort by personnel of the Physics and Chemistry Divisions of Argonne National Laboratory was begun in mid-1975 to design and construct a superconducting linear accelerator[1] to provide precision beams of heavy ions for nuclear physics research. The accelerating elements consist of a linear array of independently-phased split-ring resonators[2] operating at an rf frequency of 97 MHz and a temperature of 4.6 °K. The drift-tube assembly of the resonator is fabricated entirely of niobium and cooled internally with flowing liquid helium. The cylindrical housing is constructed of an explosively-bonded composite of niobium on copper, as are the end plates which attach by means of a demountable rf seal. These innovations have produced a rugged and highly successful superconducting rf resonator. The beam is focussed radially by superconducting solenoid lenses which achieve higher field strengths and greater focussing power than a quadrupole lens of the same size; the solenoids are placed one after each second resonator in the beam line. The cryostats, which currently house up to six resonators and three solenoids, are interchangeable, modular vacuum tanks twelve feet in length and three feet in diameter. They contain liquid nitrogen and liquid helium distribution systems, with all control and instrumentation signals coming in and out of the cryostat by way of a nitrogen filled instrumentation line. The superconducting linear accelerator, the first application of rf superconductivity to heavy-ion acceleration, has been tested successfully with a two-cryostat configuration of six resonators and five solenoids, and is scheduled to expand to a 16-resonator, 8-solenoid system by mid-1979.

*This work was performed under the auspices of the U. S. Department of Energy.

System Design

In choosing a control system[3] for this new accelerator, a computer-based system was seen as the best approach for a flexible, powerful control mechanism with expansion capabilities. The goal was to focus all monitoring and control functions into a simple, compact console, featuring computer-driven components, that could supply all monitoring information needed by an operator on demand or in real time, provide alarm messages should the need arise, and allow full control of all active elements.

CAMAC hardware was chosen as our standard instrumentation and interface system because of its inherent modularity and flexibility. It was decided that the console should consist of a limited number of display and control devices whose functions were assigned as needed and that the main device for displaying information to the operator would be a color TV monitor. (The new dimension that color provides not only allows separation of fields of data for improved readability but is extremely useful for drawing the operator's attention to warning or alarm messages.)

The potential for major hardware re-configuration in such an experimental project necessitated a modular software system driven by tables defining the instrumentation needs and the accelerator configuration. To achieve this goal a multi-tasking operating system with inter-task communication was indicated.

Computer Hardware

The computer system, designed to completely control and monitor the superconducting linac, is based on a PDP 11/34 processor with 96K words of memory and floating point hardware. Peripherals include a dual disk drive, a Versatec electrostatic line printer/plotter, a Tektronix 4006 storage tube graphics terminal, a CRT terminal, and a CAMAC serial highway system. An operator's console is located next to the computer and is interfaced to it by means of several CAMAC modules and standard DL-11 interfaces (Fig. 1).

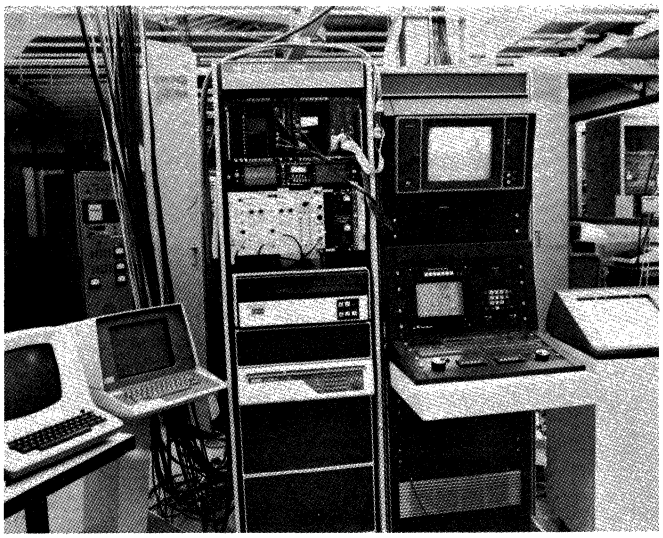


Figure 1. Superconducting linac control system.

CAMAC Hardware

A master CAMAC crate resides with the computer in the accelerator control room and is interfaced to the 11/34 by means of a UNIBUS crate controller. In addition to the CAMAC serial highway driver, modules within the master crate include a color TV monitor driver to provide displays of current accelerator conditions, a touch panel display driver by which most control is initiated, gate modules to interface two high-quality ADC's (analog to digital converter) for beam diagnostics, and a control panel interface module. The single serial crate is located approximately 150 feet from the computer, next to the accelerator beam line, and contains all the modules necessary to control and monitor the linac. A 32-channel scanning ADC is used to measure solenoid currents, as well as read out thermocouples that monitor the temperature of the nitrogen-cooled heat shield. Thermometry at liquid-helium temperatures is accomplished by measuring the resistance of germanium resistors. The computer routes a particular resistor to a lock-in amplifier by means of a specially designed digital I/O subsystem controlled by CAMAC; the amplifier output is connected to an ADC input for computer read out. The resonators each require two voltages to set a phase and one to set the field strength, for a total of three DAC (digital to analog converter) channels per resonator. The current of each solenoid is adjusted by setting the DAC's of the independent solenoid power supplies over the same digital I/O bus used for the liquid-helium temperature thermometry. Beam current is measured, digitized, and read out through a CAMAC scaler module, while cryostat vacuum is interfaced through a CAMAC input gate module. It is estimated that the capacity of a single serial crate is sufficient to handle all control and monitoring needs of up to four cryostats.

SOFTWARE SYSTEM

Overview

It seemed clear from the outset that the software system would require a real-time multi-tasking system with a reasonable response time, inter-task communication, and support for time-based task initiation. RSX-11M V3.1 was chosen as the most suitable DEC operating system for this application. All tasks but two are written in Fortran IV and Fortran resident library support was included. (The two exceptions were heavily involved with CAMAC hardware and it was essential that they execute as quickly as possible.) All interactions with CAMAC hardware were handled with Fortran-callable Macro assembly language subroutines. Most tasks can be activated from the operator's console, with provisions for scheduling several of them for repetitive time-based execution (e.g., execute now and every 10 minutes thereafter). Several specialized tasks that perform rarely needed table initialization or input a great deal of information are executable from a terminal only. Details of typical operation will be discussed later.

CAMAC I/O

The computer interfaces to CAMAC via a Kinetic Systems 3912 crate controller which maps each module subaddress to a unique UNIBUS address so that functionally each module appears to be on the UNIBUS (a module readout may simply consist of a movement of data out of a specific location, e.g., MOV SUBADR, RØ). It can also generate up to 24 unique interrupt vector addresses for module LAM's (look-at-me signals) or it can OR any number of them to a given interrupt vector. Since most of the CAMAC I/O operations involve relatively simple processes such as reading an ADC, setting a DAC, or writing information to the TV monitor, one might consider mapping to the I/O page to perform them. However, even if one could tolerate the risk of giving every task privileged access to the I/O page (!), one would have to serialize access to modules such as the TV monitor driver that may have as many as four tasks trying to write information "simultaneously" (e.g., normal information and high priority alarm messages).

The CAMAC I/O problem was solved by using an RSX-11M compatible driver[4] that treated each UNIBUS crate controller and each serial highway driver as a distinct logical unit. Each unit could further be given up to 256 independent I/O subchannels for the queuing of requests made with the standard RSX QIO mechanism. In typical operation, subchannel ϕ of a given unit is used for all non-interrupt I/O to any module addressed by that unit, and subchannels numbered 1 or greater are assigned to specific modules that produce LAM's. As a consequence, I/O to any given module is serialized, and, since each logical unit is always free and only individual subchannels can become busy, a "wait-for-LAM" request can be issued for as many modules as there are subchannels. This CAMAC driver supports I/O requests consisting of single

or multiple CAMAC (FNA) commands (with optional looping capability), suspension of FNA list execution pending a LAM, and makes serial highway operations transparent at the QIO level. This driver has proven very effective in this multi-tasking environment, and is used for all control and monitoring applications.

The potentially high data rate associated with the beam diagnostic software prevented the use of the CAMAC driver for this application. When tuning one of the superconducting resonators, an operator scans a range of phase angle settings to find the phase angle corresponding to the desired energy gain for the beam particle being accelerated. Also of interest during this tuning process is the temporal distribution of the beam pulse. Two 4K channel energy and time spectra are accumulated through high-quality ADC's interfaced to CAMAC by means of input gate modules. The typical 2-parameter event data rate is about 1 KHz, with the rate potentially reaching 4 KHz. For this application, an ADC gate module was assigned a unique interrupt vector address for its LAM, and the V3.1 Connect to Interrupt Vector directive (CINT$) was employed. Since this application is representative of full usage of CAMAC hardware and multi-tasking with a great deal of inter-task communication, it will be discussed in some detail here— the details of operator initiation of tasks will be presented in a subsequent section.

When the operator indicates to the resonator setting task RESON that he is about to set the field strength and phase angle of a particular resonator, a task called CENTROID is initiated. This task creates an 8K dynamic region for storage of the energy and time spectra, maps to it, clears the region, and initiates task ACCUM. The data acquisition task ACCUM, written in Macro, maps to the 8K region, issues the Connect to Interrupt Vector directive CINT$, and enters a wait for global event flag state (when set, the flag indicates ACCUM is to disconnect from the interrupt vector and exit). When ADC interrupts occur, the interrupt service routine is entered in Kernel mode, and its virtual address space includes all of the Executive, the pool, and the I/O page. The two ADC's are read out directly (since the crate controller maps their CAMAC subaddresses into the I/O page) and the information placed in one of two buffers. When a buffer is full, a fork level routine is called to allow an AST (asynchronous system trap) to be queued for the pulse-height analysis routine. The analysis is carried out from the buffer at task level priority. Using the criteria of 50% ADC deadtime for a mid-scale peak at 4096-channel digitization, an 8 KHz rate can be handled with no noticeable deterioration in system performance. Using the Mark Time directive, CENTROID is activated every two seconds, calculates the centroid of the single peak appearing in the energy spectrum and writes its location (i.e., energy) on the TV monitor for the operator using a QIO to the CAMAC driver. Viewing the beam energy value on the monitor, the operator may reset a resonator phase angle, which results in a QIO the CAMAC driver to set the necessary DAC's. The operator typically activates task DISPLAY (from task RESON) to display either the energy or time spectrum on the Tektronix storage tube terminal (using a subroutine package which issues appropriate QIO's to the standard RSX terminal driver). Task RESON communicates with tasks DISPLAY and CENTROID by means of the SEND DATA and RECEIVE DATA directives for various option selections. Figure 2 indicates the data flow and inter-task communication at this point.

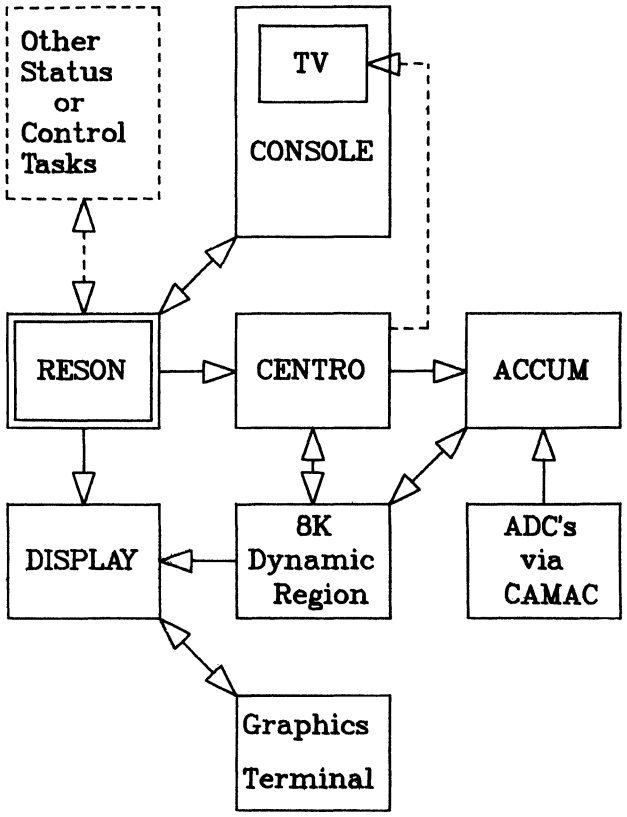


Figure 2. Control and information flow.

When the operator signals completion of a resonator setting, tasks CENTROID and ACCUM automatically exit and the 8K dynamic region is deleted.

Configuration-Independent Software

Realizing that the experimental nature of the accelerator could potentially result in frequent changes in the instrumentation and the hardware configuration, all software addressing any accelerator monitoring or controlling hardware is table driven. An initialization task is used to load a disk resident accelerator configuration table and define the order and placement of the resonators and solenoids and define the CAMAC addresses of the DAC's and ADC's needed to control them. Any task needing to control or monitor some aspect of an active accelerator element (a solenoid or resonator) reads the table entry for that element and retrieves the necessary CAMAC access information, typically the FNA code in a compact, one-word form used by the CAMAC driver. A subroutine that issues the QIO to the driver for a control or monitor function would use the FNA code as its input argument. This mechanism allows dynamic hardware reassignment should the need arise. For example, if one channel of a scanning ADC becomes defective, it is a simple matter to reassign another channel or even another module to replace its function. Similarly, all thermometry parameters including CAMAC access information and temperature conversion coefficients are stored in easily-modified tables residing on disk as random

access files. Since some temperature sensors are read frequently, it would be undesirable to retrieve CAMAC access information each time they are read out, so an in-core table is built from the disk file for those sensors. Should any access information change, it is a simple matter to signal the thermometry software to rebuild its in-core table from a newly constructed disk file.

This approach is not used on modules such as the TV monitor and touch panel display drivers since they are not subject to possible reassignment or reconfiguration. All software addressing these "system" modules assumes a fixed CAMAC address.

OPERATOR'S CONSOLE

One of the more interesting aspects of the linac control system is its very simple control console which is the focal point of all monitoring and controlling functions (Fig. 3). All control activity

Figure 3. Control console.

is initiated at the touch panel display, a black and white TV monitor over which is positioned a touch-sensitive transparent panel containing a 4 × 4 matrix of capacitance-sensitive areas. The display driver for the touch panel is a CAMAC module, and all I/O is accomplished with the CAMAC driver. A given control task draws and labels buttons on the screen corresponding to the touch-sensitive areas and assigns a particular function to each (Fig. 4). Above the touch panel is a CAMAC driven color TV monitor on which can be displayed all information gathered by monitoring tasks (e.g., temperature, vacuum readings, solenoid currents, etc.) either in real time for constant monitoring of an item or in the form of a summary of the most recently recorded data from a disk file. Of the 26 lines visible on

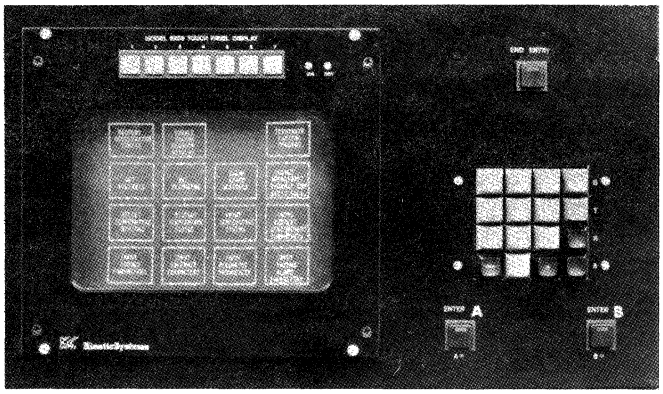


Figure 4. Sample touch panel menu.

the screen at one time, 20 are reserved for such information, and the last 6 for warning or alarm messages and special task usage. A color monitor was chosen for its ability to add emphasis to warning and alarm messages to attract the operator's attention and give some indication of the severity of the condition. For example, a flashing red message is used when a power supply failure is suspected and immediate operator intervention is required. Similarly, yellow messages warn of out-of-range temperature readings, a problem less severe than equipment failure, but requiring attention. Green text is likewise used for an acceptable real-time condition, while white is used for general information. It was not felt that this added dimension of relative importance could be effectively conveyed on a black and white monitor. Below the touch panel are three 32-character alphanumeric displays, two 5-digit numeric displays, and two control knobs (Fig. 5); beside the touch

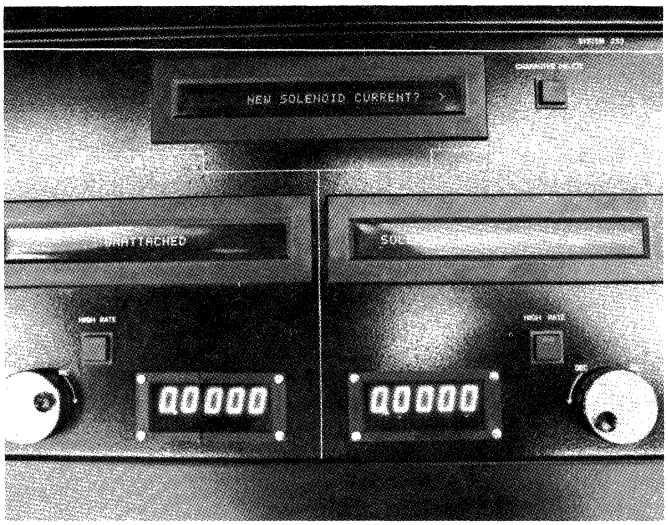


Figure 5. Control knobs and displays.

panel is a numeric key pad (Fig. 4). All five displays and the keypad are interfaced by means of DL-11 type interfaces so that all I/O to them is done with QIO's to the RSX terminal driver. The control knobs, interfaced via a CAMAC scaler module, and the displays are only "logically connected" to a system parameter when a particular task requires it, so during the course of

controlling the accelerator, a given knob or display assumes a variety of functions. A pair of continuously rotatable knobs was chosen as the basic control mechanism for operator-adjustable device control values (currents and voltages) because of the natural association one makes between control knob rotation and analog signal levels (e.g., a clockwise rotation increases a value and a counter-clockwise rotation decreases it, as in a radio volume control). Other devices, such as position-sensitive or pressure-sensitive "joystick" controls, were considered, but seemed to lack the "natural feel" of a control knob. Several factors dictated the use of two control knobs for the console. One knob would be insufficient to satisfy the need to adjust two interrelated parameters simultaneously (e.g., resonator phase and amplitude). Having more than two knobs seemed unjustified for several reasons: 1) the inability of a single operator to simultaneously adjust more than two knobs (only two hands!); 2) the strong desire to maintain a simple uncluttered console, as opposed to the "separate-knob-for-every-variable" approach; 3) no foreseeable need to simultaneously adjust three or more variables; and 4) no need for a knob dedicated permanently to a specific function—all parameters can be adjusted by means of a computer assignable knob.

The operation of the system is best understood through use of an example session. If the operator boots in the system, the STARTUP command file installs all tasks in the control system, runs an initialization task to place the CAMAC hardware in a proper initial state, and, as its final step, executes task MAIN which displays a "main menu" of options for the operator on the touch panel display (Fig. 4). A response to a button interrupt may take one of several forms. A function such as "ENABLE VACUUM READOUT" results in a flag being set in a global common area. A function such as "CLEAR ERROR MESSAGES" is executed within the MAIN task and results in the error message field of the TV monitor being cleared. A function such as "DISPLAY TEMPERATURE STATUS" results in the execution of an independent task which displays the current temperature information from a disk file on the TV monitor. Functions such as "SET RESONATOR" or "SELECT TEMPERATURE OPTIONS" would request an appropriate independent task to run and then cause MAIN to exit. The selected control task then "takes over" the touch panel to display its menu of options for the operator. There are presently five such control tasks, two of which have two "pages" of options (one page for execution options, one page for resonator selection).

Under normal circumstances, the upper alphanumeric display of the control panel is blank, the two parameter description displays have the legend "UNATTACHED", and the numeric displays indicate a zero value. When setting a solenoid, for example, control knob "A" remains "UNATTACHED", but control knob "B" can be used to set the solenoid current. The parameter "B" display indicates its "logical connection" as the control for the solenoid current while the numeric display indicates the value to which the current is to be set. Rather than rotate the control knob, the operator could alternately input a new value using the keypad, whose "connection" is indicated by a prompt in the uppermost alphanumeric display (Fig. 5). Similarly, when adjusting a resonator, displays "A" reflect the current resonator field strength value and knob "A" allows the operator to alter its value, while the "B" displays and knob deal with the resonator phase angle. This ability to dynamically assign both a logical and a physical connection in this manner has enabled us to replace a conventional control console having numerous displays, knobs, switches, and lights, with a simple, uncluttered, yet functionally expandable console.

CONCLUSIONS

Our experience with the superconducting linac control system to date has led to the following conclusions:

1) The hardware configuration independence achieved with the table-driven software approach has proven that technique to be indispensable in an environment associated with an experimental project. On the occasions when the hardware or instrumentation needed to be reconfigured, the changes were reflected in the software system within a few minutes— much less than if a piece of software had to be modified.

2) A touch panel for task and sub-task initiation, computer assignable control devices, and a color TV monitor, have been shown to form the basis of an effective, simple, and expandable control console. Very little operator education is required because the number of controls is so limited and their labelling obvious. The touch-panel menu, at any given time, presents a list of the only reasonable options available, so inappropriate option selection is virtually impossible.

3) The RSX compatible CAMAC driver has been proven to be a safe and effective I/O mechanism for CAMAC modules. The V3.1 Connect to Interrupt Vector directive works very well with our CAMAC hardware and the data acquisition rates we encounter.

REFERENCES

1. L. M. Bollinger et al., "The Argonne Superconducting Heavy-Ion Linac," Proceedings of the 1976 Proton Linear Accelerator Conference, (AECL-5677 Chalk River, 1976) p. 95.

2. K. W. Shepard et al., "Development and Production of Superconducting Resonators for the Argonne Heavy-Ion Linac," to be published in the Proceedings of the 1978 Applied Superconductivity Conference.

3. All of the instrumentation for monitoring and controlling the accelerator, as well as the control panel itself, was designed by Mr. Robert Daly of the Electronics Division of Argonne.

4. J. W. Tippie, and P. H. Cannon, "CAMAC Driver for the RSX-11M V3 Operating System," Proceedings of the Digital Equipment Computer Users Society, Vol. 4, No. 2, pp. 563-566.

ACKNOWLEDGMENTS

Special thanks to Dr. J. W. Tippie of the Applied Mathematics Division of Argonne for numerous helpful discussions and suggestions. Thanks also to P. H. Cannon for his help in incorporating the CAMAC driver into the system.

BUREAU OF MINES DATA ACQUISITION AND PROCESSING SYSTEM

Donald N. H. Chi and Henry E. Perlee
Bureau of Mines
U.S. Department of the Interior
Pittsburgh, Pennsylvania

ABSTRACT

The U.S. Bureau of Mines' Pittsburgh Mining and Safety Research Center (PMSRC) has installed a centralized computer facility to provide on-line, high-speed (500,000 samples/sec) short-duration (<10 seconds) data acquisition; on-line, low-speed (<100 samples/sec), long-duration (weeks or months) data acquisition; and off-line, high- and low-speed data acquisition through control of such data storage devices as data-loggers, analog magnetic tape recorders, and satellite mini- or microcomputers. The computer facility, which is connected to the Center's various laboratories by both analog and digital data transmission lines, consists of a host PDP-11/70* front-ended with the two PDP-11/34's. The PDP-11/70 coordinates the functions of the two PDP-11/34's and processes the bulk of the data retrieved from the two front-ends. One of the 11/34's is dedicated to the high-speed data acquisition, and the other handles low-speed data acquisition, data-loggers, analog tape recorder, and satellite mini- or microcomputers.

INTRODUCTION

PMSRC is one of five research centers within the mining branch of the U.S. Bureau of Mines conducting or overseeing research related to the health and safety of the Nation's underground coal miners. The research areas at PMSRC include fires and explosions, respirable coal dust control, methane control and ventilation, roof support, explosives testing, industrial hazards, and mine communication and illumination.

Basically, there are three technical tasks for which the computer has found the most use, modeling physical systems, data acquisition and processing, and systems control; all computer-related administrative and financial functions are processed at the Bureau's Denver computer installation. The modeling tasks primarily pertain to the numerical solution of descriptive equations (mostly sets of partial differential equations) relating to the labyrinth of physical and chemical processes directly or indirectly associated with conditions hazardous to an underground coal miner. The data acquisition and processing task involves the reception of analog and digital signals transmitted from the various station laboratory transducers to the computer for storage and subsequent processing. The systems control task primarily concerns the programmed operation of relays for instrumentation, calibration, and testing purposes.

The four largest research facilities within PMSRC that use the data acquisition services are the Experimental Coal Mine, the Fire Research Facility, the In Situ Combustion Facility, and the Explosion Gallery. The Experimental Coal Mine consists of a double-entry coal mine 1,200 feet long with corridors 10 feet wide by 6 feet high in which coal dust explosions research is conducted. This facility uses 75 transducers that measure gas pressure and temperature, dust concentrations, air velocity, radiation intensity, and flame speed at 10 locations along the entries. The transducer signals, after amplification, are sent 500 feet to the computer over analog lines where they are sampled at about 2,400 samples/sec/channel. Typically, a coal dust explosion lasts less than 5 seconds and generates 1.8×10^6 data points.

The In Situ Combustion Facility involves a surface trench approximately 35 feet long by 8 feet high and 6 feet wide lined with refractory brick. Forty tons of a 95% coal, 5% cement mixture are cast into the trench to simulate a thick coal seam that contains a central channel of 1.5- by 1.5-foot cross section. The coal is ignited at one end of the channel, air is forced through the channel at various rates, and the spread of the fire, temperatures, gas product composition, smoke, pressure drops, enthalpy, radiative and convective flux, and ventilation velocity are measured. Transducer signals are acquired by a data logger and magnetic tape recorder, then sent to the computer over both analog and digital lines. Initially, each transducer is sampled once every 10 seconds, slowing to once every 15 to 30 minutes after 3 or 4 days of the burn, accumulating 10^6 data points. Paralleled

---

* Reference to specific equipments does not constitute endorsement by the Bureau of Mines

signal cables to a second data logger permits sampling speeds up to 25/second. Each test involves about 250 channels of information.

The Fire Research Facility contains two refractory-lined ducts, 30 feet long. Duct cross sections are 1 by 1 foot and 2 by 2 feet; the ducts are lined with timber or coal blocks. A fire is initiated at one end of the ventilated channel, and the rate of fire spread along the channel wall and other information are followed as described above in the In Situ Facility. About the same volume of data is generated and handled as in the In Situ Facility.

The Explosion Gallery consists of a 90-foot-long, 6-foot-diameter steel pipe used to conduct experiments similar to those run in the Experimental Coal Mine. Many of the experiments conducted in the Experimental Coal Mine are first run in this facility to get a feeling for the nature and magnitude of the processes. The facility is located 2,000 feet from the computer, and all data are transmitted and sampled in the same manner as for the Experimental Coal Mine. For laboratory grounds layout and the data lines network, see reference 1.

SYSTEM CONFIGURATION

Figure 1 shows the configuration of PMSRC's Data Acquisition and Processing Computer Facility. As the figure shows, the facility consists of three mainframes, a host PDP-11/70 front-ended with two PDP-11/34's. The upper PDP-11/34 in the figure is used exclusively for slow-speed (<100 samples/sec/channel) long-duration (days or months) data acquisition; the lower PDP-11/34 is exclusively for the high-speed (up to 500K samples/sec), short-duration (<10 seconds) runs.

Host Computer

The host PDP-11/70 has 384K words of memory, two TE16-EE magetic tape drives, two RP04 88 meg byte disk drives, one 1,200-line/minute LP11-RA line printer, one 1,000-card/minute CD11-A card reader, a Houston Incremental DP-5 plotter, two (4800 baud and 1200 baud) synchronous ports for communications with a Control Data Corp. CYBER-74 and a Burroughs B6700 computer in Denver, two DZ11's providing 22 asynchronous ports to support time-sharing users, and two DMC11's for high-speed (1 meg baud) communication with the two PDP-11/34's. The host is run under IAS supporting DECnet, MUX200 (a simulator for Control Data's UT200 remote batch terminal), FORTRAN IV-Plus, and BASIC. The host computer provides the bulk of the data-processing chores and all the outputting.

Data Acquisition Front-Ends

The high-speed PDP-11/34 front-end has 64K words of memory, one RP04 disk drive, an ICS11 with 16 voltage sense, 16 voltage interrupts, 16 latching relays, and 16 flip-flop modules, a DZ11 asynchronous multiplexer, a high-speed A/D subsystem (see below), and a DMC11 for communication with the host. Data retrieved by this subsystem are transmitted as analog signals over shielded, twisted pairs up to distances of 2,000 feet using only low-cost line-driving amplifiers.

The low-speed PDP-11/34 is almost identical to the former except it has 48K words of memory and does not have a high-speed A/D subsystem. Both systems run under RSX-11M supporting FORTRAN IV and DECnet. The voltage sense and voltage interrupts are used to sense experimentally generated event markers, e.g. times of ignition, and the latching relays and flip-flops operate relays in the laboratories for instrumentation calibration and testing.

Figure 2 shows the configuration of one of the four A/D subsystems designed to achieve a maximum throughput rate of 500,000 samples/sec for a total of 128 channels for a duration of 2.0 seconds. This subsystem consists of four modules, each containing the following hardware connected as shown in figure 2:

1. Analogic AN5864 12-bit A/D converter with 32 differential ($\pm$ 10 v) multiplexer inputs and a maximum sampling rate of 125 KHz;

2. A buffer with a storage capacity of 1meg 16-bit words is provided by the Monolithic Extended Memory (EMU). The EMU is a dual-port, semi-conductor, read/write memory that is UNIBUS and software plug-compatible with RF-11/RS-11 disk controller and fixed-head disk subsystem;

3. DR11-C, a general digital interface permitting communication between the EMU and the PDP-11/34 bus;

4. X4, a 16-bit parallel interface between the A/D and the EMU. (not shown in figure 2)

In addition, a single COMTEL-SEG Crystal Clock with a switch-selectable pulse rate from 0.1 to 13 MHz drives all four modules simultaneously. The trigger-channel shown in the figure trips the EMU, which in turn sets high 1 of the 16 parallel output bits of the DR11-C connected to the 11/34 bus. As the controlling program in the 11/34 sees this bit, it triggers the A/D via the DR11-C and EMU to start sampling data at the rate determined by the clock until the buffer is filled. Subsequently, the data are transferred to the RP04 disk and then forwarded to the host for further processing. The EMU has the ability to store data at rates up to 1 MHz. This unit can also be operated in a cycling mode where the EMU repeatedly cycles data through the buffer stopping on command from the 11/34. We believe this is probably one of the fastest digital data acquisition systems available. If desired, the complete sequence of controlling and processing steps can be fully automated.

The low-speed PDP-11/34 system, in addition to providing real-time low-speed data acquisition, also communicates with microcomputer-based remote data acquisition systems and controls the operation of an analog magnetic tape recorder and a data logger. The latter can accommodate up to 1,000 input channels at 25 samples/sec, and the mag tape has 14 channels with band widths up to 300 KHz.

Because this 11/34 does not have an A/D, analog signals generated in the laboratories are converted to ASCII and transmitted to one of the computer asynchronous ports over either shielded, twisted pairs or phone lines if modems are available. Some of the pairs in these cables also serve as control lines. Each 11/34 supports its own console, which is used primarily for operator control and systems debugging.

SOFTWARE

The 11/70 supporting software includes scientific subroutine packages such as International Mathematical and Statistical Libraries, BMDP, and SFORTRAN (Structured Fortran implementing the Chapin or Nassi-Schneiderman logical charts). All data acquisition, control, and plotting routines have been developed by the Bureau of Mines personnel. Two of these are briefly mentioned below.

PMSRC Plotting Package

The PMSRC plotting package is an integrated plotting package, patterned after DISSPLA but modified to fit within 20K words. Basically, in a few DISSPLA-like statements, the user specifies page size, location of the physical origin, axis lengths, number of tick marks, scaling, legends, and their position and arrays of data to be plotted in engineering units. The users do not need to do any data conversion. This package will be made available through DECUS.

Data Acquisition Package

This package performs such functions as automatic calibration and testing of signal conditioning instrumentation, spectral analysis, filtering, and plot preparation. Past experience has shown that experimentalists constantly change their processing requirements, and hence, code update and maintenance functions occupy a large portion of the programming staff's time. To minimize such demands, the package makes use of a data base containing all necessary parameters to process any specific experimental setup. This data base has the attribute of unlimited expansion capability for future changes. Each file of experimental data submitted for processing is accompanied by a copy of the data base, containing parameter updates obtained during the data acquisition process. For example, for each transducer, a record is created containing such information as the multiplexer channel number, type of sensor, file location of the calibration and engineering unit conversion coefficients, plotting parameters, type of filtering, and types of processing. If the user changes the types of sensors, it is a simple matter to change these records without changing the processing programs. This package is sufficiently general to process data generated and acquired by any means within the Center.

SUMMARY

The Bureau of Mines has installed what we believe is a unique data acquisition and processing system that may satisfy the requirements of many research laboratories. It has a powerful large-scale computer as a back-up for the large number-crunching jobs. It satisfies the high-speed, short-duration and the low-speed, long-duration data acquisition requirements, and has time-sharing and batch subsystems for program development. It is also equipped with the best available scientific subroutines for data processing, and the system allows expansion to handle increasing workloads.

REFERENCE

1. U.S. Bureau of Mines High-Speed Data Acquisition System, R. W. Markley and H. E. Perlee, Application of Computer Methods in the Mineral Industry, Society of Mining Engineers of AIMMPE, New York, New York 1977, pp. 395-403.

FIGURE 1

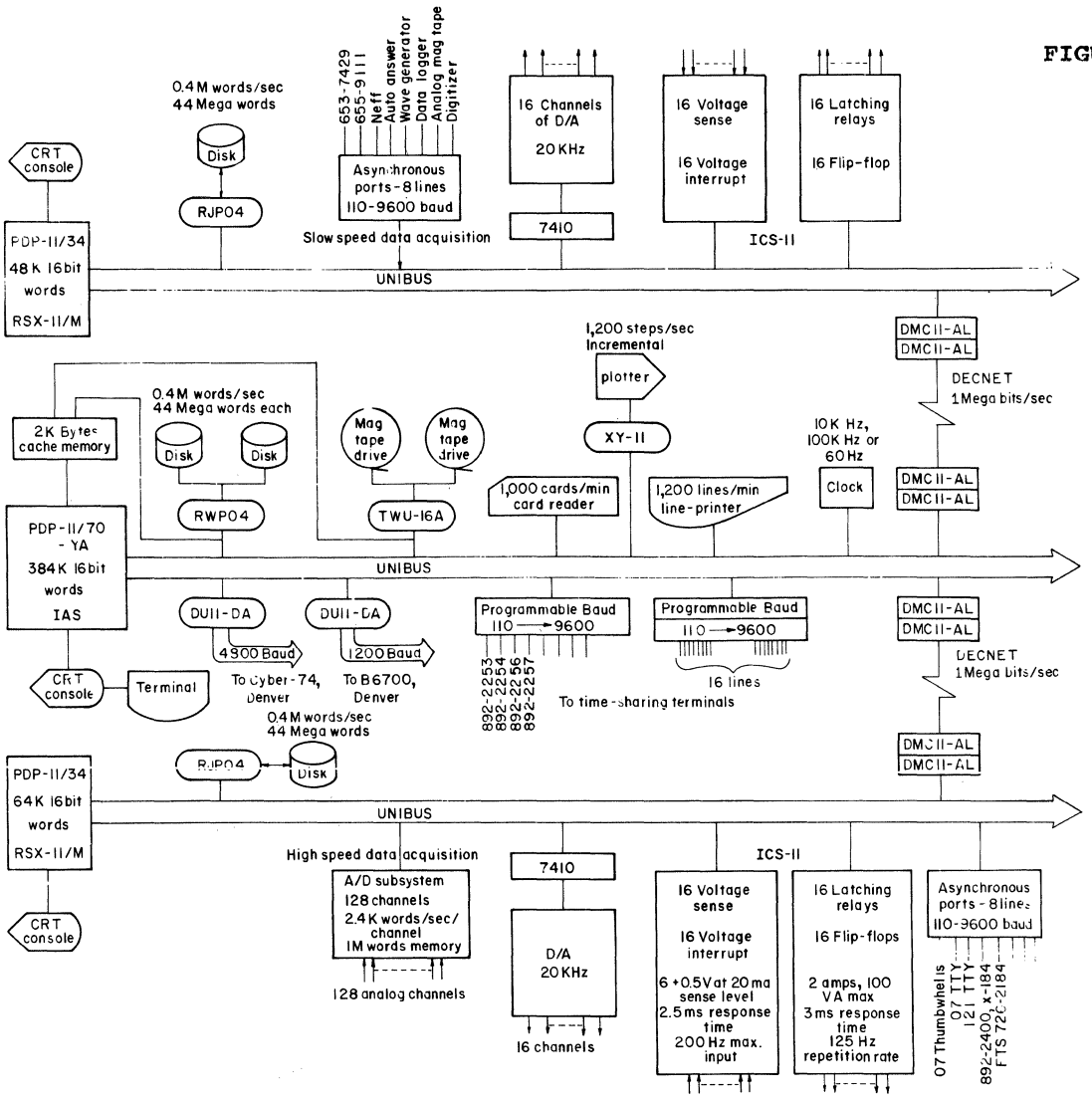
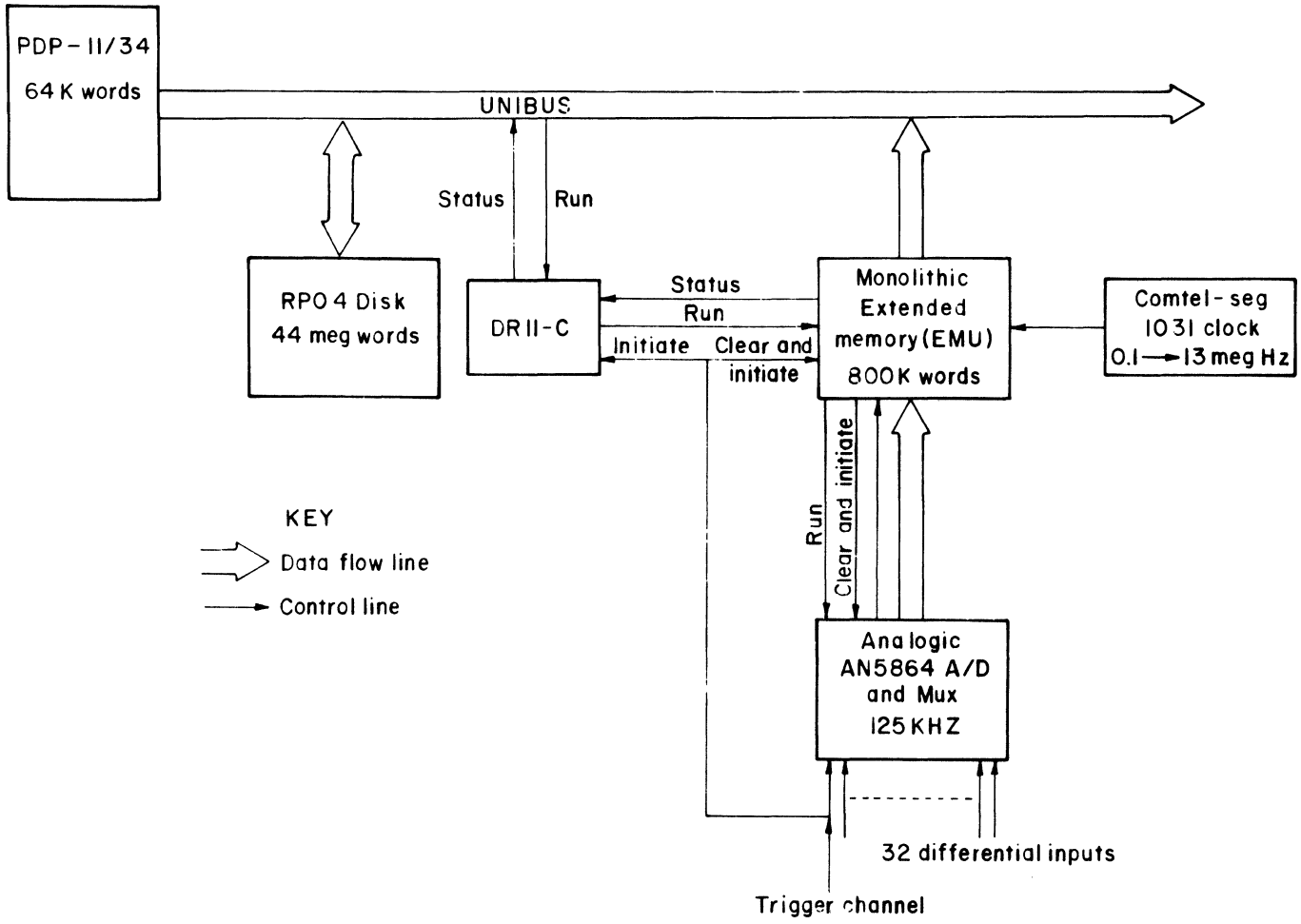

0.4 M words/sec
44 Mega words
Disk
RJP04
653-7429
655-9111
Neff
Auto answer
Wave generator
Data logger
Analog mag tape
Digitizer
Asynchronous ports - 8 lines 110-9600 baud
Slow speed data acquisition
16 Channels of D/A
20 KHz
7410
16 Voltage sense
16 Voltage interrupt
ICS-11
16 Latching relays
16 Flip-flop
CRT console
PDP-11/34
48 K 16bit words
RSX-11/M
UNIBUS
DMC11-AL
DMC11-AL
DECNET
1 Mega bits/sec
1,200 steps/sec
Incremental
plotter
XY-11
0.4 M words/sec
44 Mega words each
2K Bytes cache memory
Disk
Disk
RWP04
Mag tape drive
Mag tape drive
TWU-16A
1,000 cards/min card reader
1,200 lines/min line-printer
10K Hz, 100K Hz or 60 Hz
Clock
PDP-11/70 - YA
384K 16bit words
IAS
DUII-DA
DUII-DA
4900 Baud
1200 Baud
To Cyber-74, Denver
To B6700, Denver
Programmable Baud 110 → 9600
Programmable Baud 110 → 9600
16 lines
To time-sharing terminals
892-2253
892-2254
892-2256
892-2257
CRT console
Terminal
0.4 M words/sec
44 Mega words
RJP04
Disk
PDP-11/34
64K 16bit words
RSX-11/M
CRT console
UNIBUS
High speed data acquisition
A/D subsystem
128 channels
2.4 K words/sec/ channel
1M words memory
128 analog channels
7410
D/A
20 KHz
16 channels
ICS-11
16 Voltage sense
16 Voltage interrupt
6 +0.5V at 20 ma sense level
2.5 ms response time
200 Hz max. input
16 Latching relays
16 Flip-flops
2 amps, 100 VA max
3 ms response time
125 Hz repetition rate
Asynchronous ports - 8 lines
110-3600 baud
07 Thumbwheels
07 TTY
121 TTY
892-2400, x-184
FTS 726-2184

FIGURE 2


PDP-11/34
64 K words
UNIBUS
Status
Run
RP04 Disk
44 meg words
DR11-C
Status
Run
Initiate
Clear and initiate
Monolithic Extended memory (EMU)
800K words
Comtel-seg 1031 clock
0.1 → 13 meg Hz
Run
Clear and initiate
KEY
Data flow line
Control line
Analogic AN5864 A/D and Mux
125 KHZ
32 differential inputs
Trigger channel

INTERACTIVE GRAPHICS SUPPORT FOR MINICOMPUTER SYSTEMS

Steven J. Choy
Harry Diamond Laboratories
Adelphi, MD 20783

ABSTRACT

An interactive-graphics, minicomputer-based FORTRAN IV, software package called GRAPHELP has been developed at Harry Diamond Laboratories. The current system provides device-independent, interactive support for both direct view storage and refresh CRT terminal systems using serial communication lines. Functions supported include both absolute and relative vectors of four varying line textures, user definable scaling, windowing, clipping and 128 nested subpicture display files for refresh graphics. In addition, routines are provided for graphics input, interactive screen erase control and, for refresh terminals, selective erase. Higher level axes drawing routines are provided for data plotting applications in either linear or logarithmic form. This report presents a functional overview of the system software capability along with a collection of sample engineering applications using the graphics library on a DEC PDP-11, RSX11-D based system.

INTRODUCTION

GRAPHELP is a set of FORTRAN IV callable routines to represent data in pictorial form both interactively and passively on a CRT graphic display. Currently, output drivers exist for the Imlac PDS-4 refresh display system and the entire family of Tektronix 401X storage tube type displays. Included is support for the enhanced Graphics Mode for the Tektronix 4014 and 4015 high resolution display terminals. In addition, drivers exist on a partial basis for an Adage GP-430, DEC GT-40, and for output only, the Versatec printer/plotter.

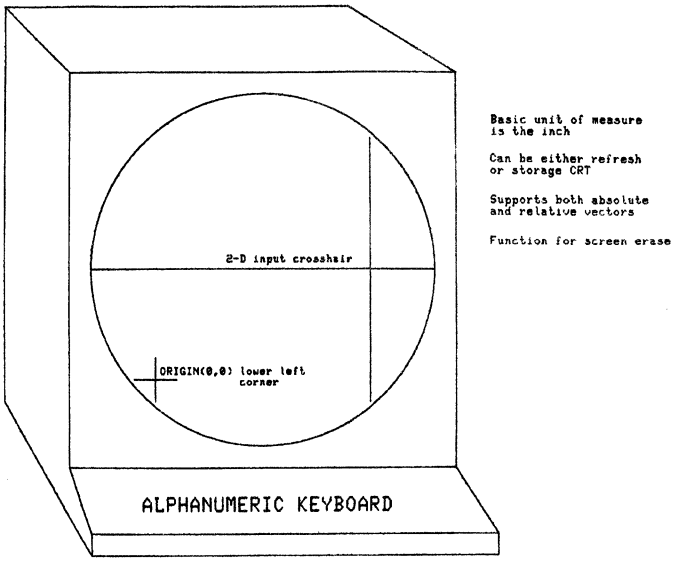
Because minicomputer applications at Harry Diamond Laboratories (HDL) span a wide range of requirements, GRAPHELP was designed to be general enough that it could support those applications ranging from general data plotting to design layout. As discussed later, the software provides this graphics support with maximal terminal transparency to the user programmer. Thus the target terminal of the application is not of primary concern to the user.

The software library has been in active use for three years on a PDP-11/45 at HDL using the RSX11-D operating system. Output to the terminals is accomplished through direct calls to QIO from FORTRAN. All terminals are connected to the PDP 11 via the DH11 multiplexor at communication rates of 9600 baud. The high communication speed allows for a high degree of interaction between user and machine because of the quick response of the computer on output.

FUNCTIONALITY

The GRAPHELP package provides support for interactive graphics in both the storage CRT and the refresh CRT environments. The two basic levels of subroutines are (1) a primitive device coordinate level and (2) a higher, user coordinate level.

At the primitive level, a canonical pseudo graphics terminal device is defined as the target for input and output (see figure 1). This device can operate in one of two states. At initialization time this terminal is in an alphanumeric communications state. In this state the terminal is treated as a conventional teletype terminal. The programmer may use standard FORTRAN READ and WRITE statements to communicate with the terminal. The second state provided is a graphics state. In this state the terminal is treated as a device for communicating graphics data to and from the computer. When in this mode, the pseudo device terminal contains an absolute and a relative vector generator which uses the inch as its basic unit of measure. The screen origin of the terminal is defined to be the lower left hand corner



GRAPHELP ideal interactive terminal

FIGURE 1

of the CRT. In addition, the terminal contains a hardware character generator for picture annotation and a function to erase the screen.

Two methods for input exist on this pseudo graphics device; an alphanumeric keyboard and a two dimensional X-Y locator. (The locator is actually implemented by use of a tablet, a lightpen, or thumbwheels.). Routines are provided to query the locator in either the terminal's coordinate system or the user's coordinate system. Keyboard input is queried by use of either a conventional FORTRAN READ statement or by use of a GRAPHELP call for single character input.

When the pseudo terminal is used as a refresh terminal, additional functions are available. These include multiple intensities, selectable blinking of picture elements, and individual subpicture definition and manipulation. Subpictures are essentially independent display segments that allow a user to selectively modify portions of the display without affecting other parts of the display. GRAPHELP supports up to 128 user-definable subpictures. These subpictures may be individually created, displayed, appended to, redefined and erased. Using the subpicture capability on a refresh terminal allows the user to present an application dynamically and/or interactively via subpicture redefinition. Any user subpicture may "call" or link to another user subpicture to a level of six deep. This allows for complex picture data structures in design layout systems. In addition, GRAPHELP provides separate display lists for each of the two terminal modes descibed previously. Thus, applications may send out temporary prompts in the alphanumeric display list without disturbing the applications graphics display.

All GRAPHELP programs using the storage CRT as the primary output may also use the refresh CRT for output with no modification to the source program or object relinking required. Those GRAPHELP programs using the refresh CRT for the primary output may also use the storage CRT, but functions specific to the refresh CRT will be ignored when the storage CRT is used. Thus, those applications requiring subpictures will not run correctly on the storage CRT.

At the higher level of GRAPHELP, a general two-dimensional plotting package is provided for general data plotting and other general user applications. Graphic data is expressed in the user's own coordinate system, and appropriate routines exist for scaling and translating the user's coordinate system to the primitive device coordinate system. This scaling can be either linear or logarithmic. In either case, clipping boundaries (windows on the data) may be constructed in the users coordinate system to enable "zooming" of a picture. At the users option, the clipping can be disabled. The user may define up to eight different coordinate systems and save and restore these different coordinate systems as needed by the application. The default user coordinate system is the canonical pseudo device coordinate system.

At this high level the user does not have to worry about the lower-level graphics functions that are being performed to implement this higher level two-dimensional graphics. The user can perform all his graphics routines in his own coordinate system without having to worry about scaling transformations, vector overflow on the screen, or any of the other common headaches associated with graphics programming. This level is totally device independent.

For general data plotting, subroutines are available to draw both linear and logarithmic axes. In addition symbolic display of floating point numbers can be generated with an autoformatting capability. The user can optionally select the output precision of a number in terms of the maximum number of decimal places displayed. Data-scaling routines that are compatible with CalComp routines are also incorporated in GRAPHELP, along with routines to plot data lines. For interactive viewing of data plots, a conditional screen erase is provided where the erase condition is a function of keyboard input.

The next section presents a few of the varying applications at HDL that use GRAPHELP for interactive graphics support.

SAMPLE APPLICATIONS USING GRAPHELP

A large number of the computer graphics applications at HDL are related to the area of viewing collected data. For example, the air gun facility obtains experimental data in the form of high speed "streak" films. The traces on the films are then digitized using an automated digitizer to obtain time-displacement data. GRAPHELP is subsequently used to display reconstructed film scan digitizations.

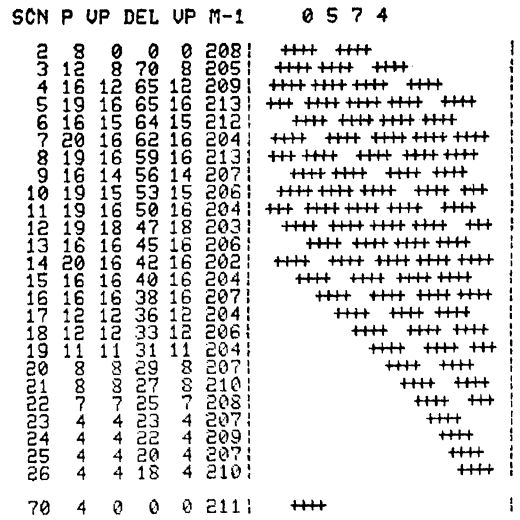


FIGURE 2

Computer analyses are then performed and the results are plotted again using GRAPHELP. Figure 2 shows a typical output of the reconstructed scan and figure 3 shows a typical X-Y linear graph of the computed results.

The shock and vibration test facility uses a GRAPHELP-based program to interactively view multiple data plots. Using this program, an engineer graphically and mathematically constructs an ideal input shock pulse using individual sine, triangular, and square waves or combinations of these. A shock spectrum transformation of the idealized input pulse is performed and the result is plotted on the CRT along with the originally specified upper and lower limits. If the resulting output function is not within the required limits, the engineer can alter the input pulse and redo the transformation. Once the desired pulse is created, an actual test is performed using the idealized pulse as input data to the shock machine. The machine-generated test pulse is recorded and then digitized and used as input into the program. The transformation is applied to the test pulse and compared graphically with the transformed output of the ideal

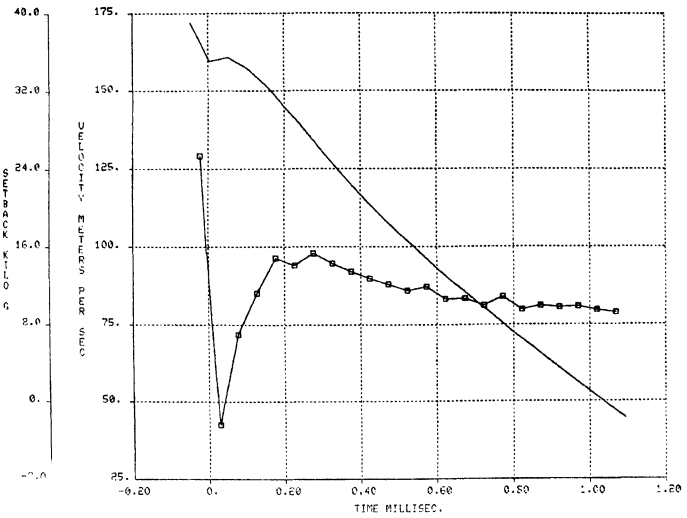


FIGURE 3

pulse. Figure 4 shows a plot of an actual test pulse and its resulting transformation. Use of this technique enables better simulation of the real world environment.

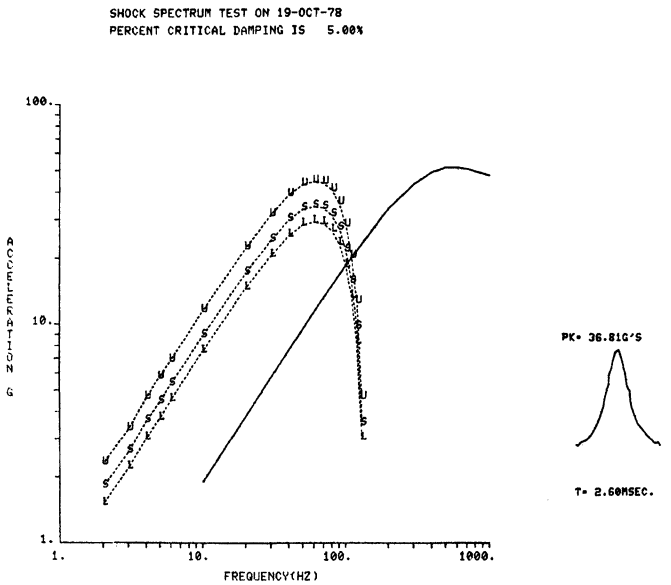


FIGURE 4

An additional area of use for GRAPHELP at HDL is in design layout. One example of use is an interactive system for generating block diagrams and flow charts for technical reports. Figures 1 and 5 show typical outputs produced by this GRAPHELP based program. Another example of engineering design layout is that of computer aided integrated circuit (IC) mask layout. Figure 6 shows a typical IC layout constructed interactively using a GRAPHELP

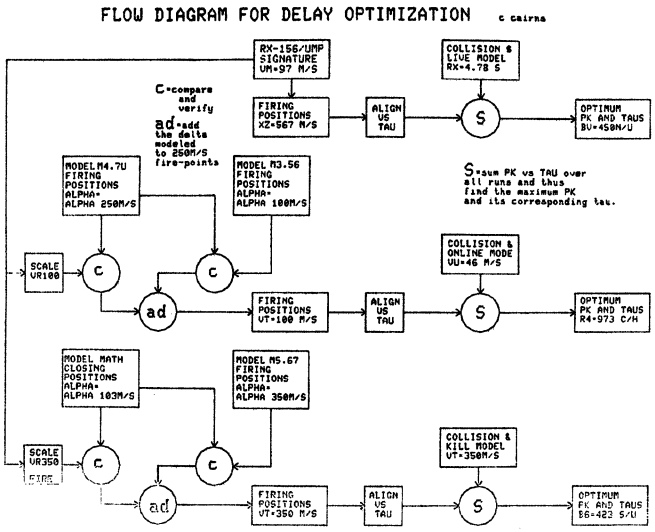


FIGURE 5

based program. The picture shown in figure 7 was magnified using the crosshair cursor and the clipping facility.

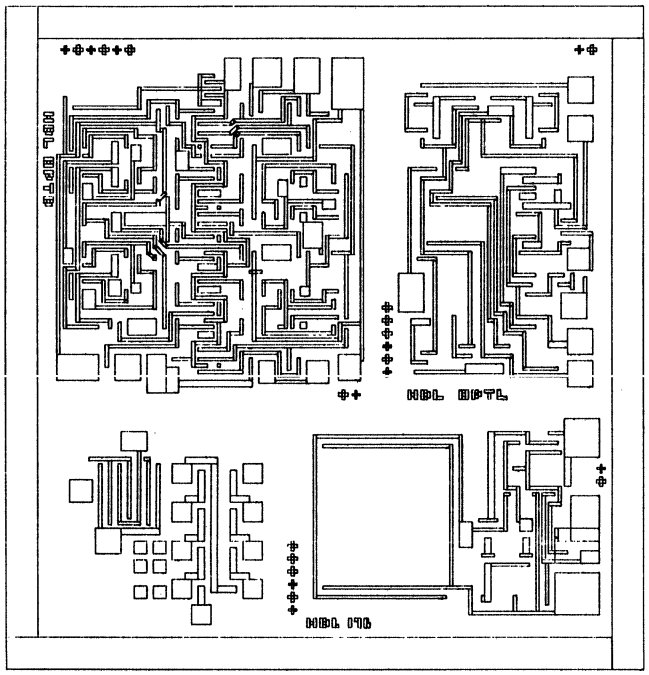


FIGURE 6

As an example of dynamic computer graphics, figure 8 shows one frame of a simulated missile-target encounter that was developed using GRAPHELP's subpicture capabilities. The figure shows three views of the encounter. Utilizing the subpicture facility, the views are updated for each time increment of the simulation, giving the viewer a continuous update of the simulated encounter.

For those engineers and scientist that do not wish to write computer graphics programs to plot test data, a generalized

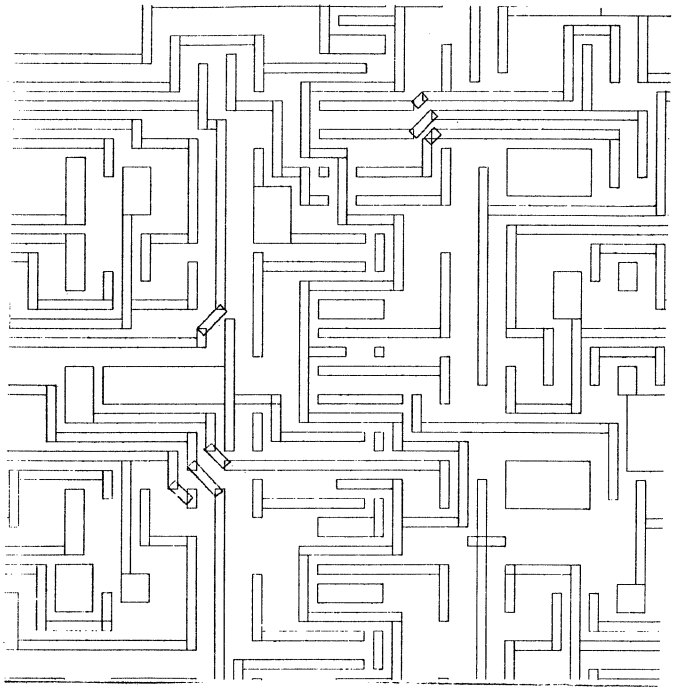


FIGURE 7

two dimensional plotting task is provided for non-programmers. The task is essentially an interactive gateway to

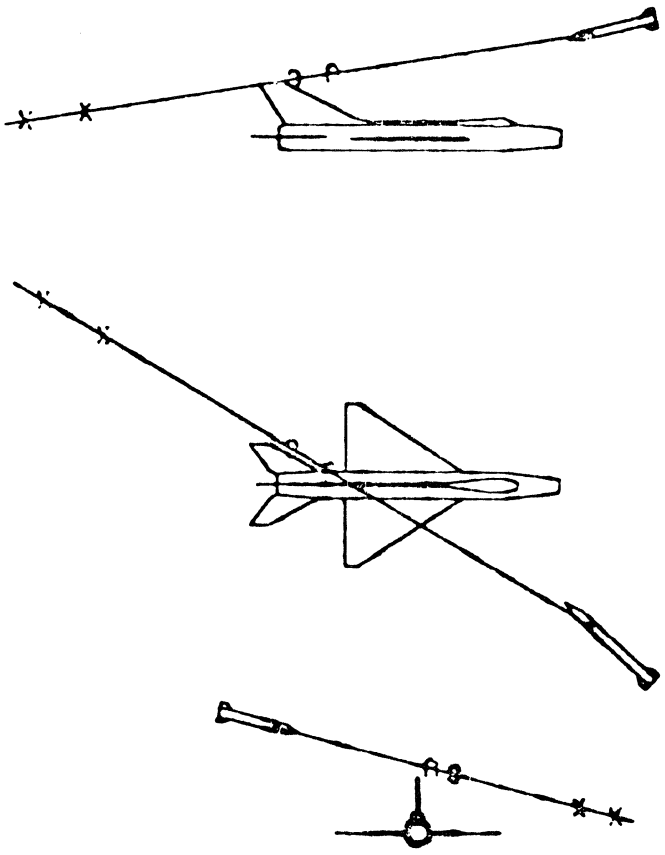


FIGURE 8

GRAPHELP plotting subroutines. Users may specify up to fourteen different sets of

data in multiple files. The input files must be in ASCII readable form, but no real restrictions are placed on the format content of the data. Functions available to the user include, text specification for axes, data identification, legends, titles; windowing and zooming of data; line texture and symbol plotting choices; command file input and output; and arbitrary line annotation. The command language combines english like statements in conjuntion with 2-d graphic input. If any coordinate argument is missing from a command, it is queried from the user by the task via the graphic input device of the terminal.

CONCLUSION

Minicomputer-based systems provide an ideal vehicle for interactive graphics systems. The quick response time, coupled with high output rates on the communication lines allow users to interact with the computer in an interactive environment. Basing the graphics interface on a high level language such as FORTRAN decreases the time to learn to use the system effectively for scientists and engineers. The GRAPHELP system implemented at HDL is one demonstration that interactive graphics may be used on small systems as a tool to support a wide variety of engineering applications.

PDP-11 IMPLEMENTATION OF A PROPOSED ANSI DATA EXCHANGE STANDARD

Paul J. Dionne, E. Richard Hill, John Bower, Anne Medford, Debbie Mathisen
Battelle, Pacific Northwest Laboratories
Richland, Washington

ABSTRACT

The Interlaboratory Working Group for Data Exchange (IWGDE) was formed in 1975 to address the problem of exchanging massive amounts of environmental data among the Department of Energy's (DOE) community of national laboratories. The IWGDE found no efficient means of exchanging these data and so created an exchange standard using 9-track tape. This Standard has been submitted to the American National Standards Institute (ANSI) for consideration as an ANSI Standard and is now under review.

Using the Standard solves the problems that plague researchers who wish to exchange data, specifically differences in computer hardware and configurations, operating systems and operating philosophies.

Basically, the Standard defines a Data Descriptive File (DDF) followed by a Data File (DF). The DDF contains control parameters and the necessary data descriptions for interpretation of the data records and data elements in the DF. Data are stored in the DF as a series of tag/value pairs. Tags identify the values they accompany.

The Standard is implemented on several IBM and CDC configurations, most recently on PDP-11/70s. The Standard has been used successfully for over a year and is available to others who are interested. The PDP-11 version is written in FORTRAN IV-PLUS and runs on RSX-11/IAS operating systems.

INTRODUCTION

The Interlaboratory Working Group for Data Exchange (IWGDE) was formed in an ad hoc manner during 1975. The need for data exchange and information coordination was recognized early in regional studies programs and representatives from each of the DOE laboratories began meeting to define the problem and develop solutions.

It was found that each laboratory had large amounts of various types of environmental and demographic data which were of interest to others. There was no effective or efficient means of exchanging these data. Each request for information had been handled with a variety of responses, none of them standard. Worse yet, the fact that some laboratories had IBM equipment while the rest had CDC equipment compounded the problem because of differences in hardware configurations, operating system software and computing philosophies, both by the manufacturer and by the laboratory using the equipment.

The IWGDE quickly decided that a common method of data and information exchange should be used and set about to develop a DOE Standard. The first implementation was done at the Oak Ridge National Laboratory (ORNL) in 1976 and documented in that year's Annual Report.(1) Since that time the DOE Standard has been submitted to the American National Standards Institute for consideration as an ANSI Standard.(2) It is under review now.

Staff at ORNL implemented the Standard in PL/1 for IBM equipment while Brookhaven National Laboratory (BNL) and Los Alamos Scientific Laboratory (LASL) staff, with financing from the Savannah River Laboratory (SRL), began to implement the CDC version.(3) Level 1 versions are now operational at all laboratories. Argonne National Laboratory (ANL) is using the ORNL imple-

mentation while SRL has created a different IBM version for compatibility with their data handling system. Lawrence Berkeley Laboratory (LBL) and Lawrence Livermore Laboratory (LLL) have created their own CDC implementations. The Pacific Northwest Laboratory (PNL) used the CDC version on the CDC 6600 run by Boeing Computer Services Richland for the Hanford contractors until the PDP-11 version was operational. See Figure 1 for the locations of the IWGDE Standard implementations.

The Level 2 implementation will still do nothing with the DDF but will handle multi-dimensional data in the DF. Level 2 will also have a set of higher-level subroutines available to the user which will make the job of creating an IWGDE Standard tape easier.

The Level 3 implementation will handle hierarchical data structures in the DF and will use the DDF information to crack the DF.

Laboratory	Location	Computer	Software
ANL	Argonne, IL	IBM	PL/1
BNL	Upton, NY	CDC	FORTRAN/COMPASS
LASL	Los Alamos, NM	CDC	FORTRAN/COMPASS
LBL	Berkeley, CA	CDC	FORTRAN
		PDP-11/45	FORTRAN IV-PLUS/MACRO
LLL	Livermore, CA	CDC	LLLTRAN
ORNL	Oak Ridge, TN	IBM	PL/1
PNL	Richland, WA	CDC	FORTRAN/COMPASS
		PDP-11/70	FORTRAN IV-PLUS/MACRO
SRL	Aiken, SC	IBM	FORTRAN
		PDP-11/45	FORTRAN IV-PLUS/MACRO

Figure 1. IWGDE Standard Implementations

The Digital Equipment Corporation's PDP-11 is a widely used minicomputer in most of the DOE community and, in many cases, is more accessible and less expensive than the large machines for use by research staff who wish to send or receive environmental data on industry standard 9-track magnetic tapes.

Rather than create a new implementation from scratch, we chose to convert as much of the existing CDC Level 1 implementation as possible. We felt that we could save time by not reinventing the wheel, by taking advantage of already created procedures and by facilitating communication between several technical implementations using the same subroutine names and philosophy. The PDP-11 version, in its present form, is capable of reading or writing Level 1 data and reading multi-file volumes.

DISCUSSION

Basically, the Standard requires the user to write two files on tape for every set of data or information to be exchanged, as shown in Figure 2. The first file is the Data Descriptive File (DDF) and describes the data to be transmitted. The second file is the Data File (DF) described by the DDF.

The Level 1 implementation of the Standard writes a DDF and DF. The DDF is not used except to manually read the data description. The DF contains tag/value pairs, or essentially, one-dimensional information. A universal program is available to read and print the DDF and DF of any Standard tape. However, in order to extract data from the tape and format it according to user specifications, a program must be written using the IWGDE subroutines.

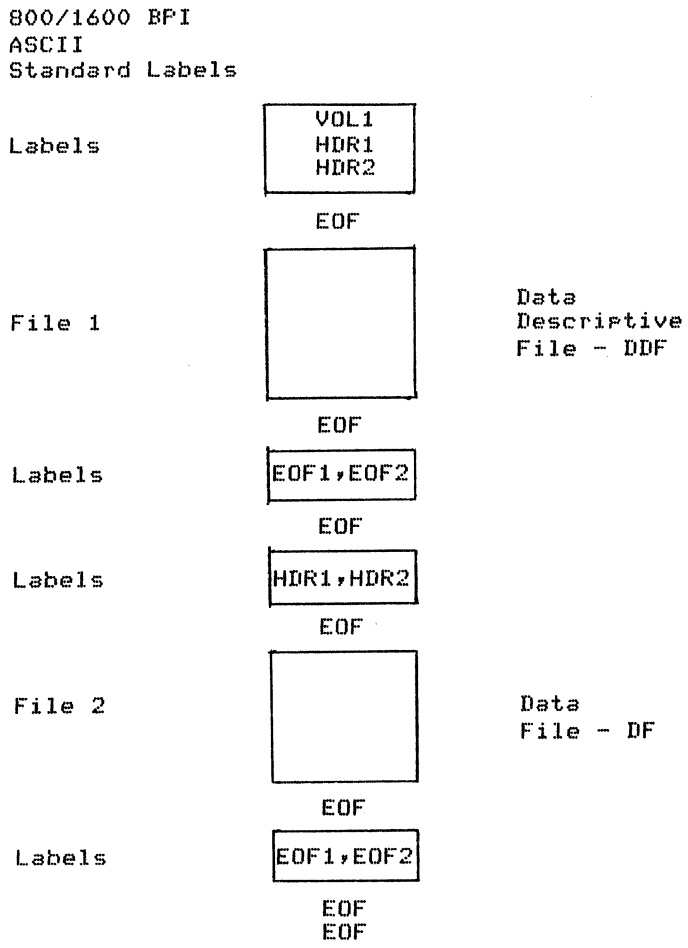


Figure 2. Tape Structure

When the Level 3 implementation is complete the researcher receiving the tape can mount the tape on the tape drive, call the standard read routine which will read the DDF, extract the appropriate pages, titles, headers and formats and read the DF into a file for printing or manipulation. This will apply for all generally accepted data structures.

This Standard has already been used as the basis for exchanging geographic data and associated thematic data which contain a wide variety of complex and interlocking data structures.(4)

Features

The IWGDE Interchange Standard eliminates common tape-cracking problems many of us have experienced when trying to write a tape for someone off-site or when trying to read a foreign tape we've received. For example, the Standard requires the use of standard labeled tapes which allows easy use across computers. The Standard requires the use of the ASCII character set, not binary or EBCDIC characters making the format machine independent.

The only machine-dependent part of the PDP-11 implementation is the collection of magtape primitives which are written in MACRO. The rest of the code is modular, using FORTRAN IV-PLUS. We use the primitives, along with the FORTRAN routines, for ease of installing the Standard at other facilities. The PDP-11 implementation is interactive and enables one to quickly peruse a tape sent from off-site using a generalized tape reading program.

Standard label processing is implemented in a set of FORTRAN IV-PLUS subroutines. These subroutines provide label reading, checking and writing and are used by the data interchange subroutines.

Reading a tape can be done with a generalized routine; however, each tape that is written requires an application-specific code.

The implementation of the Standard produces data in an exchange format. Once the data is on-site, it should be converted to a site-specific format. For example, if you request data from someone's ADABAS database on an IBM computer and you want to load the data into your DBMS-11 database, you must write a generalized translator to convert the IWGDE Standard into a DBMS-11 recognized data structure.

Right now the Standard is capable of creating one DDF/DF file pair (or one standard file) of information per tape. Later the Standard will support the capability to transmit several files per tape and multi-volume files.

A generalized tape reading program that we have created uses the data interchange subroutines, labeling subroutines and magtape primitive subroutines to read the DDF and DF of a standard tape. Standard label checking is performed. The user can specify the file and block within the file at which to start reading. Blocks within a file can be skipped.

SUBROUTINE DEFINITION

There are several levels of routines which can be generically classified as 'User', 'Intermediate' and 'Basic' as shown in Figure 3. 'User' identifies those routines which the user has available to him/her for the purpose of creating a program which will read or write a tape for the application at hand. 'Intermediate' identifies those routines which read and write the ANSI Standard label records, the DDF, and DF data. 'Basic' identifies the machine-dependent magnetic tape primitives and commonly used utility routines.

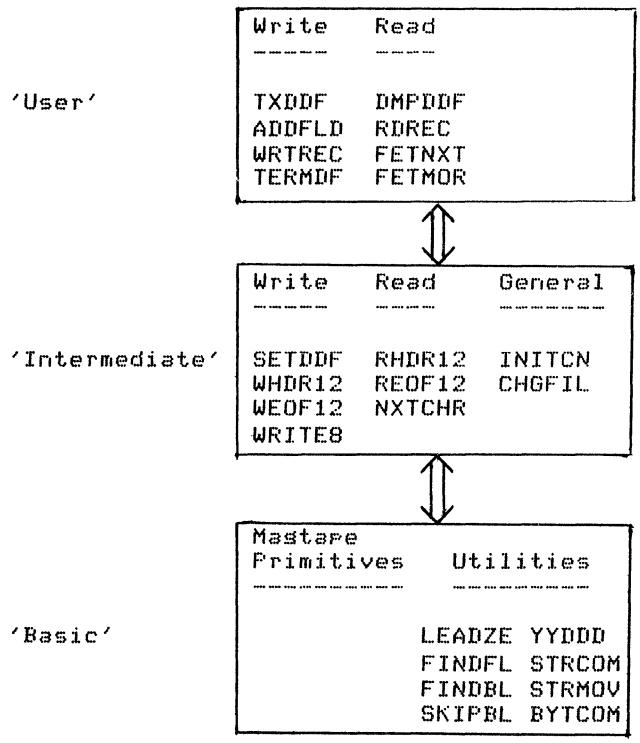


Figure 3. Subroutine Structure

'User'

Write

TXDDF is an optional routine provided for the convenience of the user. If called, TXDDF reads a user DDF descriptive file, writes the DDF on tape and leaves the user ready to write the DF.

ADDFLD adds a tag/value pair to the DF buffer.

WRTREC writes the logical record.

TERMDF closes the output file.

Read

DMPDDF reads and prints the DDF. (It logically 'opens' the standard file.)

RDREC reads the next logical record from the DF.

FETNXT gets the next tag/value pair from the current logical record.

FETMOR gets the next part of the value string from the current field. This subroutine is useful when the value string exceeds a reasonable maximum length. The entire string is obtained by successive calls to this subroutine.

'Intermediate'

Write

SETDDF formats the leader for the DDF file.

WHDR12 writes the ANSI standard HDR1 and HDR2 label records before the DDF and DF.

WEOF12 writes the ANSI standard EOF1 and EOF2 label records after the DDF and DF.

WRITE8 writes the current buffer to tape and EOF mark, if appropriate.

Read

RHDR12 reads the ANSI standard HDR1 and HDR2 label records and checks for correctness.

REOF12 reads the ANSI standard EOF1 and EOF2 label records and checks for correctness.

NXTCHR extracts a character at a time from the input buffer. When the buffer is empty, the next tape block is read.

General

INITCN interactively opens the standard tape file by reading or writing the ANSI standard VOL1 label record followed by calling RHDR12 or WHDR12, depending on the application.

CHGFIL switches processing from the DDF to DF file. It processes all labels between the DDF and DF files and processes the trailing labels for the DF file. It calls REOF12 and RHDR12 or WEOF12 and WHDR12 between the DDF and the DF depending on the application. CHGFIL also calls REOF12 or WEOF12 and reads or writes an EOF mark after the DF.

'Basic'

Magtape Primitives

The Magtape Primitives consist of several machine-dependent routines for reading and writing blocks of data from and to the tape along with a number of error routines. The Magtape Primitives are written in MACRO.

Utilities

LEADZE fills an integer string with the appropriate number of leading zeros.

FINDFL finds the requested file on the tape for selective perusal.

FINDBL finds the requested block on the tape for selective perusal.

SKIPBL skips a specified number of blocks on the tape to reach the section desired for selective perusal.

YYDDD calculates the year and day in the YYDDD format (78123) using the system subroutine IDATE.

STRCOM compares, character by character, two strings that are LBYTES long, and returns a flag indicating a true or false comparison.

STRMOV moves one string, LBYTES long, into another string.

BYTCOM compares the characters in the middle of one string with another string. A true or false flag is set depending upon the result.

EXAMPLE

An example demonstrates the way the IWGDE Standard is used to create an exchange tape. Figure 4 shows the two input data files which may be in card form or reside on your computer's mass storage device.

The DDF of Figure 4 is pre- and post-delimited by *DDF and *EODDF. These delimiters are used by the TXDDF subroutine. The first line of the DDF contains tag 000 which is reserved by the IWGDE Standard for designating the title of the database. The title for the example is "Author List." Notice later, in Figure 6, that tag 000 is not transmitted as part of the DF.

The second line, tag 001, is also reserved by the IWGDE Standard for the record identifier field. This tag must appear in each logical record of the DF, and need not be numeric.

```
*DDF
000     AUTHOR LIST
001     RECORD KEY
100     ORGANIZATION
101     ADDRESS
102     CITY
200     AUTHOR
201     TELEPHONE
202     LOCATION
*EODDF

BATTELLE, PACIFIC NORTHWEST LABORATORIES
POST OFFICE BOX 999
RICHLAND, WA 99352
PAUL J. DIONNE
(509) 946-2452
LIFE SCIENCES LABORATORY
DICK HILL
(509) 946-2675
MATHEMATICS BUILDING
```

Figure 4. Example DDF & DF Source Data

The third line, tag 100, is the first user defined tag. (Tags 000-009 are reserved by the IWGDE Standard for database name, record sequence, and other yet-to-be defined uses. All other tag numbers or names are available for use by the user. Again, tags need not be numeric. User defined tags in the DDF contain information about the data to be transmitted.) Tag 100 identifies the Organization Name which for the example value is "Battelle, Pacific Northwest Laboratories."

The fourth line, tag 101, identifies the Address which for the example value is "Post Office Box 999."

The fifth line, tag 102, identifies the City, State and Zip Code which for the example value is "Richland, WA 99352."

The last three tags, 200-202, define a set of repeating data. Tag 200 identifies an Author Name, tag 201 the author's Telephone Number and tag 202 the author's Office Location. For the example two of the authors are listed along with their telephone numbers and office locations.

The DDF is always one record long and that record is limited to 2046 bytes. The DF can contain as many logical records as the user wishes to define. The IWGDE Standard software handles the spanning of logical records which are larger than 2046 bytes.

Figure 5 is a flow chart of a program which will write the IWGDE Standard tape with the data of Figure 4. After initializing the variables and arrays, especially RECKEY, the program opens the DDF and DF source data files of Figure 4. TXDDF calls other routines which write the VOL1, HDR1 and HDR2 label records. It transfers all the DDF source data as text to the tape and then calls other programs which write the EOF1 and EOF2 label records for the DDF and the HDR1 and HDR2 label records for the DF.

To write the DF to tape, RECKEY is increased by one, and a line is read from the DF

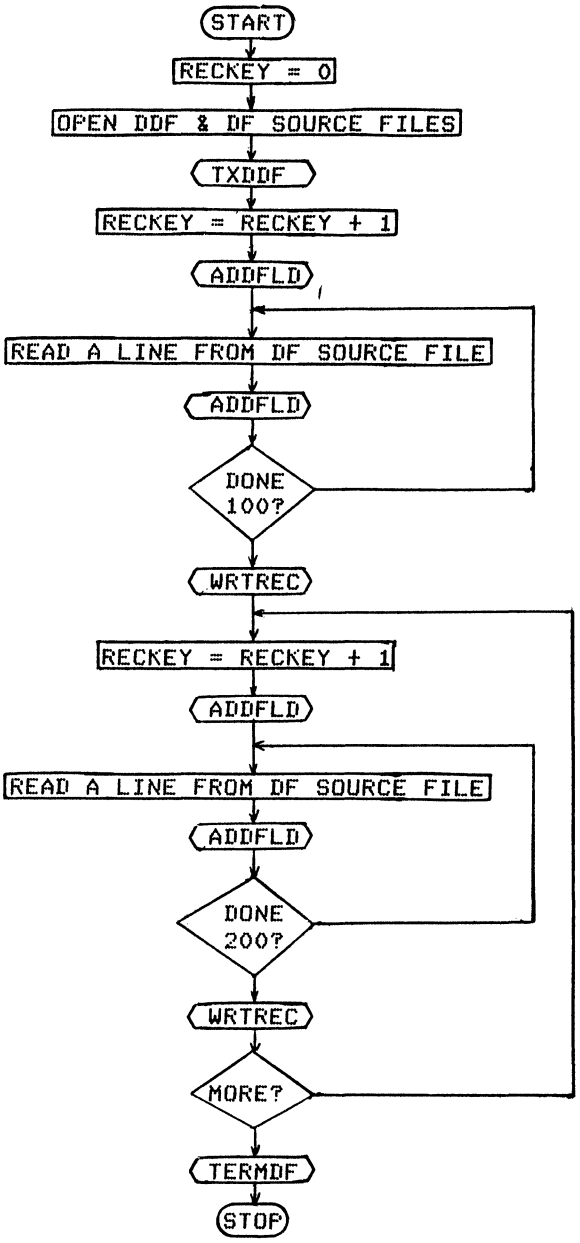


Figure 5. Example Program to Write an IWGDE Tape

source file and added to the current buffer. Two more lines are then read and added to the current buffer before the record is written to tape. The first DF record consists of RECKEY and the organization data associated with tags 100-102.

RECKEY is then increased by one again and added to a new current buffer. The data associated with tags 200-202 are read from the DF source file and added to the current buffer. That buffer is then written to tape as the second logical record. The program then continues to look for more data associated with tags 200-202 and writes as many of these records as there are data in the DF source file (in the example, one more).

Having written all the data to tape from the DF source file, TERMDF is called. TERMDF calls other subroutines which write the EOF1

and EOF2 label records and a logical-end-of-tape mark.

Figure 6 shows the results of writing the source data files of Figure 4 to tape using the IWGDE Standard with the example program of Figure 5. Each record of the file consists of the ANSI Standard Segment Control Word (positions 1-5), the 19 character IWGDE Standard leader, a variable length directory (whose length depends upon the amount of data in the record) and the body which is simply the data itself.

directory for each field of data. Each entry contains the starting position of the data, length of data (including terminators), and the tag which identifies the data.

For the example DDF of Figure 6, the record length (SCW) is 156 characters, the leader identifies the directory and itself to be 76 characters in length. It further states that each directory entry is seven characters long (2 + 2 + 3): the character count field is 2 characters long, the starting position is 2 characters long and the tag is 3 characters long.

```
00156_____000076___223_000120000111121001323101083610205442000749201105620209 66
;
AUTHOR LIST;
RECORD KEY;
ORGANIZATION;
ADDRESS;
CITY;
AUTHOR;
TELEPHONE;
LOCATION=

00135_____000048___223_00102001004102101204310219 63;
1;
BATTELLE,_PACIFIC_NORTHWEST_LABORATORIES;
POST_OFFICE_BOX_999;
RICHLAND,_WA_99352:
00108_____000048___223_00102002001502201151720223 32;
2;
PAUL_J._DIONNE;
(509)_946-2452;
LIFE_SCIENCES_BUILDING:
00107_____000048___223_00102002001602201151820221 33;
3;
E._RICHARD_HILL;
(509)_946-2675;
MATHEMATICS_BUILDING=
```

Note: Spaces are shown by underline character for clarity. Semi-colons represent field terminators, colons represent record terminators and equal-signs represent file terminators.

Figure 6. Example IWGDE Standard Tape

The Segment Control Word (SCW) is the integer number of characters in the record including the SCW and all terminators.

The leader contains two numbers. The first number is the length of the leader including the field terminator. The second 3-digit number describes the format of each directory entry. (A directory entry is comprised of three elements: character count of the data, starting position of the data, and the tag which identifies the data.) The first digit of the format descriptor defines the length of the character count field in the directory entry. The second digit defines the length of the starting position field in the directory entry. The third digit defines the length of the tag field in the directory entry.

The directory describes the data in the body of the record. There is one entry in the

Examining the directory, we find that tag 000 is 12 characters long and starts at position 00, tag 001 is 11 characters long and starts in position 12, tag 100 is 13 characters long and starts in position 23, and so forth. The directory and each field are terminated with an ANSI Standard non-printing field terminator (ASCII RS, equivalent to 30 decimal). We have chosen to replace the non-printing field terminator with a ';' for clarity. The DDF is one record long and is terminated by a file terminator, which is the ANSI Standard non-printing character FS (ASCII 28 decimal). The '=' is the character we have chosen to print for clarity.

The first record of the DF is 135 characters long. The leader identifies the directory and itself to be 48 characters long. The

directory says tag 001 is 02 characters long starting at position 00, tag 100 is 41 characters long starting at position 02, and so forth.

The DF contains three records, each with its own SCW, leader, directory, and body. The definition of the data in the last two records of the DF can be translated in the same manner as described above. In a multi-record file, record terminators are the non-printing ANSI Standard character GS (ASCII 29 decimal). The ':' is the character we choose to print for clarity.

BIBLIOGRAPHY

1. D. L. Austin and D. Merrill, "ERDA Interlaboratory Working Group for Data Exchange (IWGDE), Annual Report for FY1976". LBL-5329, Lawrence Berkeley Laboratory, Berkeley CA, September 1976.

2. ANSI Subcommittee X3L5, "Draft Proposal - American National Standard Specification for an Information Interchange Data Descriptive File". Working Paper ANSI X3L5/78-77F, American National Standards Institute, September 1978.

3. R. A. Wiley and C. M. Benkovitz, "User's Guide for the Implementation of the Proposed American National Standard Specification for an Information Interchange Data Descriptive File on Control Data 6000/7000 Series Computers". LA-6940-MS, Los Alamos Scientific Laboratory, Los Alamos NM, September 1977.

4. P. J. Dionne (editor), "Geographic Exchange Standard and Primer". PNL-2748, Pacific Northwest Laboratory, Richland WA, October 1978.

5. C. M. Benkovitz, "Facilitating Data Interchange Within ERDA", BNL-22595, Brookhaven National Laboratory, Upton, NY, April 1977.

6. E. R. Hill, J. C. Bower, P. J. Dionne, A. E. Medford, D. I. Mathisen and L. H. Gerhardstein, "User's Guide for the Implementation of Level One of the Proposed American National Standard Specification for an Information Interchange Data Descriptive File on Digital Equipment Corporation PDP-11/70". To be published, Pacific Northwest Laboratory, Richland, WA, August 1978.

ACKNOWLEDGEMENTS

The authors wish to identify those people outside of PNL who have contributed substantially to the PDP-11 implementation of the IWGDE Standard.

First of all, Dr. A. A. Brooks (ORNL) created the Standard which is being reviewed by ANSI. Dr. John Suich (SRL) provided the funds and encouragement for the implementation. Carmen Benkovitz (BNL) and Richard Wiley (LASL) created the original CDC version upon which the PDP-11 implementation is based.

A MEMORY RESIDENT OVERLAY HANDLER FOR RT-11 V3

D. Ritchie and Y. Kang
Fermi National Accelerator Laboratory
Batavia, Illinois 60510

ABSTRACT

A memory resident overlay handler for RT-11 SJ V3 is described. The handler allows the use of memory above 28K for program overlays. The handler has been written in such a way that no linker modification is required. Only slight modifications to the RT-11 SJ device drivers are required to handle I/O to addresses above 32K. With the exception of an initialization call, the handler's operation is entirely transparent to the user's program. Some restrictions on program organization are required.

INTRODUCTION

About every eight seconds, the Fermilab particle accelerator delivers protons to experiments during a "spill time" of approximately one second duration. During the spill the experiments acquire as much data as possible. The data is in the form of experimental events (up to several hundred events per spill and within the range of 50 to 1000 16-bit words per event). During the inter-spill period, the online program logs the events to tape from memory buffers or a spooling disk file and analyzes as many events as possible. The RT-11 SJ monitor is the simplest DEC monitor capable of satisfying these needs efficiently.

Without special modifications, programs in RT11 SJ must occupy no more than approximately 25K of memory (28K - monitor - device drivers). By dividing a large program into segments not needed in memory at the same time and loading these segments from mass storage into the same memory region when called, the effective program size may be increased. The cost, however, is 10 to 100 milliseconds per call.

In applications such as event analysis in online programs, this time overhead can not be tolerated. With the memory resident overlay handler described below, one may reduce the time overhead to no more than 170 microseconds per overlay. (This figure assumes that the segment has been loaded once; the time for the initial load is similar to the standard overlay case. Careful programming insures that these initial overheads are experienced at the tolerable level of once per accelerator cycle or less.)

The size of the handler is 500 words, including tables adequate for a rather extensive overlay structure. This is significantly less overhead than alternative methods to make more memory available for programs. These alternatives involve using a more complex operating system, e.g. RSX-11M or the extended memory version of RT-11, RT-11 XM. In these cases, the operating system, although more powerful, requires more memory and other resources merely for its own use. Also, particularly in the RSX-11M case, the time overhead to handle event interrupts and process event data is increased. (More conditions must be checked and met before control can be given in a reliable way to the user's event read-out and processing routines.) The responsiveness of the system in handling its single most important task, that of spill-time data acquisition, is thus degraded.

The memory resident overlay handler consists of two parts: the first part is an initialization subroutine, MROI, which prepares some tables necessary for use at the overlay call.

The second is the overlay call service routine, MROSEV, equivalent to the handler which the DEC linker provides. This handler, however, will load segments into memory above 28K (as well as into the standard overlay area).

The handler requires a KT-11 memory management unit (and therefore, a PDP-11/34, 45, 55, or 60). A total of at least 32K of memory is required in order for the routine to be useful. The handler has been written in such a way that no linker modifications are necessary. The handler requires only slight modifications to the RT-11 SJ V3 device drivers (to handle I/O to addresses above 32K).

USE OF THE HANDLER

To use the memory resident overlay handler, one simply makes an initialization call:

```
CALL MROI(MEM,NRS)
```

where the integer input argument MEM specifies the top limit of the memory to be used (in order of increasing address) for the program, standard overlay regions, drivers, monitor, and additional overlay regions. If memory exists above MEM, it may be used for anything the user wishes: data buffers, histogram storage,etc. The handler will not modify it. MEM should be in units of 1024 words. NRS is an integer output argument which, if the initialization is successful, reports the number of region sets allocated by the handler within the addresses 0 to MEM K specified. (A region set is the collection of overlay regions laid out for the program by the RT-11 linker. Region set 1 is the usual overlay area immediately above the root and below the monitor, drivers, and Fortran free space area. The additional region sets begin at 28K physical address.)

Note that a special condition occurs when NRS equals or exceeds the maximum number of segments specified for any region. In this case, once the program has called all the overlays, segment reads from the program load device will stop. All segments are in memory and the handler simply switches the map from one to another as each is called.

If the initialization is not successful (due, for example, to the linking requirements not being observed), NRS is set negative and a message printed. Once the handler has been successfully initialized, one may then proceed to call various overlays. See the section on handler operation for an understanding of how the initialization works.

LINKING REQUIREMENTS

There are two linking requirements: (1) The root segment must end just before a 4K word boundary. (2) The highest overlay region must end just before a 4K word boundary. Unless these requirements are met, NRS is set to -1 in the initialization call.

We accomplish the first requirement using the /U:20000 linker round switch. Please refer to the example given below. First, we write and compile a simple dummy routine defining a unique .CSECT or COMMON block name. Next, we specify this module somewhere in the root segment linker command line along with the /U:20000 switch. When the linker asks for the section name to round, we give the .CSECT or COMMON block name defined above. The linker will then extend this program section so that the root segment ends just before a 4K word boundary.

The second requirement is best met by adjusting the size of a dummy array in one of the segments occupying the highest region. The size should be increased until the second condition is satisfied.

LIMITATIONS AND RESTRICTIONS

The present implementation of the memory resident overlay handler is limited to maximums of 16 region sets, 10 regions, and 50 segments. Increasing the number of regions and segments requires only a simple change of assembly parameters and reassembly. Increasing the maximum number of region sets requires more involved program modification. The present maximums are adequate for most Fermilab applications.

The implementation goal of a simple handler necessitated a few additional programming restrictions, besides the usual restrictions on overlay calls and references as stated in the linker manual. Observing the following rules will satisfy the restrictions. Under certain cases, one may bypass the rules and still have a correct program. A thorough understanding of the overlay technique and the memory management hardware is necessary to determine the allowed bypass cases.

1. Only overlay calls from the root segment may have arguments. Making an overlay call with arguments from another overlay, such as a CALL SEG2(A,B,C) from SEG1, is not allowed. However, the reference, CALL SEG2, from SEG1 is allowed, as is CALL SEG1(D,E,F) from the root segment. Nested overlay calls with arguments are prohibited because the arguments may reside in an area of memory no longer accessible after the call.

2. Overlay segments may not do direct memory access I/O to buffers within the overlay area. The I/O requires the physical addresses of the buffers. Only the virtual addresses are available to the segment unless special calculations are done. Unformatted direct access Fortran statements such as READ (1'10) ARRAY and I/O using SYSF4 (SYSLIB) read and write routines are the sort of operations prohibited by this rule.

   Note that operations such as Fortran formatted I/O and I/O involving root segment buffers are allowed. Fortran formatted I/O operates correctly because the actual I/O involves a buffer in the Fortran free space region (above the overlay region and below the drivers and monitor). Here, as with the root segment, the mapping is such that the virtual address is always the same as the physical address. The format conversion transfers the data between this buffer and the variables in the segment. These CPU transfers require the virtual addresses which are available. (They are provided through the standard internal argument passing of the Fortran I/O system.)

SYSTEM ROUTINE MODIFICATIONS

The handler specifies all addresses for segment reads by the "odd-bit" method. This allows physical addresses anywhere in the range 0 through 124K to be specified in a manner consistent with the normal use in the 0 to 32K region. The drivers for the program load devices have been modified to accept this.

In this method, the 18 bit address is put into a two-word block (high 2 bits, low 16). The address, with its odd bit set, of this two-word block is given to the monitor in the standard I/O call. It is passed to the driver. Upon finding the odd bit set, the driver proceeds to obtain indirectly the 18 bit address from the block for placement into the appropriate device registers. Since even addresses are normally specified to program load devices and since these are handled in the usual way, regular driver use is not affected. Further, because one still only requires one word for the memory address specification, it is not necessary to change the length of queue elements within the monitor.

OVERLAY HANDLER OPERATION

Please refer to Figure 1 for a typical memory layout on a machine with at least 40K.

Initialization

The initialization call causes a jump instruction to the MROSEV overlay call service routine to be put into the first two words of the DEC-provided overlay handler ($OVRH). (The linker inserts this handler begining at the base address of every overlayed program.) The initialization call also sets up some tables for the use of MROSEV.

The information needed for these tables is developed from the segment table following the DEC overlay handler. This table contains the load address, the relative block number, and the length in words for each segment in the program. The primary tables set up are the segment-to-region/region-set table (SEGTRR), and the region/region-set-to-segment table (RRTSEG). A table called the region/- region-set use-bit table (LRUBT) is also prepared. Initially, of course, the tables show no segments in memory.

Overlay Call

When a call to an overlay is made, the MROSEV handler gains control via the dummy subroutine inserted in the user's program by the linker and the jump instruction inserted by MROI. The segment number times 6 and the memory address at which to enter the overlay are parameters to this call. Using the segment number, MROSEV then examines SEGTRR to determine whether or not the segment is in memory in some region set.

Segment In Memory

If the segment is in memory (the segment's region set number as entered in SEGTRR is non-zero), the handler changes the KT11 registers to map that region set into the overlay area. It also updates the region/region-set use-bit table and the current region set number. It then enters the segment at the desired entry address. The time required for this in-memory call is approximately 100 microseconds.

Note that the handler always uses the KT-11 memory mapping unit in kernel mode. This allows other uses to be made of the user and supervisor modes and mapping registers. It also simplifies certain aspects of the use with existing programs. (For example, SPL instructions and debugging HALT's continue to perform as expected.)

Segment Not In Memory

If the segment is not in memory, the handler calls the RSRCH subroutine to select a region set into which the segment should be read. (Remember that a segment must be put into the region for which it was linked, but that this may be in any region set.) The handler then reads the segment into the region of the selected region set. If the region previously contained a segment, it resets the tables appropriately and updates them to reflect the new segment in memory. With the segment in memory, it proceeds as with the in-memory case. The time required for the not-in-memory call is composed of the handler setup and finish-up time, and the overlay read-in time. The first is 200 microseconds. The second is dependent on the overlay size and disk position, and is a minimum of 11 milliseconds for a 1000 word overlay from an RK05 disk.

Overlay Return

When a return from a segment is made, the handler segment return routine is first executed. (It's address was placed on the stack at the overlay call.) This routine simply removes the stacked parameters and takes the actual segment return if the stacked previous region set number and the current one are the same. If they are not the same, the routine returns to the previous region set by switching the memory map back. It then removes the parameters and takes the actual segment return. In the map switch case, the return time is 70 microseconds. In the no-map switch case, it is 20 microseconds.

Region Set Selection

The region set select subroutine (RSRCH) uses an approximation to a "least-recently-used" algorithm (1) to select a region set. MROSEV sets a bit in the use-bit table corresponding to a particular region/region-set whenever a segment residing or just loaded into that area is called. RSRCH scans the use-bits for a particular region over all the region sets. Use-bits found to be set are cleared and the corresponding region set is skipped. The first use-bit found clear stops the scan. The corresponding region set is returned.

Certain applications may require a different selection algorithm for efficient or correct operation. The selection algorithm is isolated in a single subroutine to make it easy to replace with an alternative routine.

In particular, an alternative RSRCH and some utility routines have been developed which permit the user to specify those segments allowed to be loaded into region sets 2 and higher. RSRCH operates as described for these segments. For all others, it selects region set 1. In this way, the handler may be used with existing overlayed programs in which some segments obey the restrictions of no arguments, etc., while others do not.

EXAMPLE

```
C       PROGRAM ROOT
          COMMON /PARAM/ P4,P5,P6,P7,P8,P9
          COMMON /ANS/ A4,A5,A6,A7,A8,A9
          DATA   MEM/40/
          CALL MROI(MEM,NRS)
          IF (NRS.EQ.-1) STOP 'MROI FAILED'
          CALL SEG1(ARG1)
          CALL SEG2(ARG2)
          CALL SEG3(ARG3)
          STOP
          END

          SUBROUTINE SEG1(A)
          CALL SEG4
          CALL SEG7
          RETURN
          END

          SUBROUTINE SEG2(B)
          CALL SEG5
          RETURN
          END

          SUBROUTINE SEG3(C)
          CALL SEG9
          CALL SEG6
          RETURN
          END

          SUBROUTINE SEG4
          COMMON /PARAM/ ........
          COMMON /ANS/  .......
          RETURN
          END

          SUBROUTINE SEG5
          CALL SEG8
          RETURN
          END

          SUBROUTINE SEG6
             .
          RETURN
          END

          SUBROUTINE SEG7
          COMMON /ANS/ .......
          WRITE (5,205) A7
205       FORMAT (E10.3)
          RETURN
          END

          SUBROUTINE SEG8
          COMMON /PARAM/ ........
          READ (4,203) A8
203       FORMAT (E10.3)
          RETURN
          END

          SUBROUTINE SEG9
          DIMENSION IDUMMY(1)
          RETURN
          END

          SUBROUTINE DUMMY
          COMMON /SNTNEL/I
          RETURN
          END
```

BATCH FILE

```
 R MACRO
*MROI=MROI
*MMINIT=MMINIT
*MMERR=MMERR
 R FORTRAN
*ROOT=ROOT
*SEG1=SEG1
*
*SEG9=SEG9
*DUMMY=DUMMY
.R LINK
*EXMPL,EXMPL=ROOT,MROI,MMINIT,
MMERR,DUMMY/U:20000//
*SEG1/O:1
*SEG2/O:1
*  .
*SEG9/O:3//
*BOUNDARY SECTION? SNTNEL
```

Based on the load map, EXMPL.MAP, the length of the array IDUMMY in SEG9 is increased by the amount needed to cause region 3 to end just below the next higher 4K word boundary. Then, SEG9 is recompiled and EXMPL is relinked.

(MROI contains both the initialization and the MROSEV service routine. MMINIT is called by MROI. It initializes the KT-11 so that the initial memory map with memory management on is the same as that with it off. MMERR handles any memory management error traps.)

REFERENCES

(1) Madnick, Stuart E., and John J. Donovan, Operating Systems, McGraw-Hill, New York, 1974. Page 155-156.

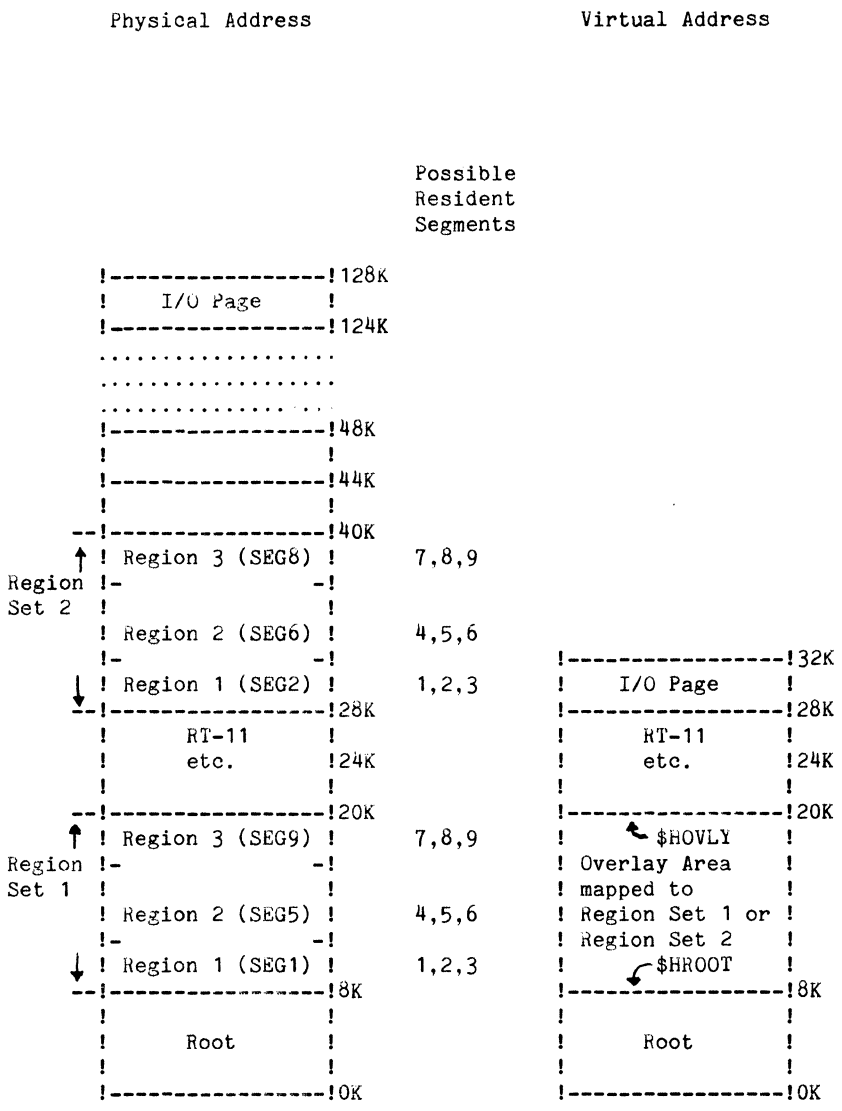


Figure 1. Memory with segments 1, 2, 5, 6, 8, and 9 loaded.

SOFTWARE DEVELOPMENT FOR A SIGNAL PROCESSING TASK: A COMPARISON OF LABFORTH WITH FORTRAN AND ASSEMBLY LANGUAGE

Ronald M. Harper and David J. Sirag
Department of Anatomy and The Brain Research Institute, UCLA
and Laboratory Software Systems
Los Angeles, Calif, 90049

ABSTRACT

A biological signal acquisition and analysis system has been implemented on the LSI-11 computer in a fast interactive language, LABFORTH, operating under RT-11. LABFORTH is a structured, high level language which allows ready access to assembly language, fast execution, compact code, and extremely rapid development time. The task was to acquire a large number of biological signals at a variety of different sampling rates over very long time periods. The data consisted of both analog and event (point process) data acquired during sleep and waking states. Analytical routines consisted of time series analysis and point process procedures, as well as a variety of data manipulation and display utilities. A package similar to this developed in assembly language and FORTRAN provided a basis for comparison of these languages with LABFORTH. The principle differences between developing the LABFORTH version and the assembly language/FORTRAN version was a markedly reduced time for program development and a great reduction in length of source code. The reduced length of source code has dramatically eased program maintenance over the earlier FORTRAN/assembly language version. The interactive capability of LABFORTH has allowed very easy modification of the routines for unique situations. It thus has proven to be an excellent laboratory language for the LSI-11.

INTRODUCTION

A variety of biological and physical science applications require the acquisition of a large number of different analog signals as well as the capability to acquire event, or point process data. the analog signals may differ widely in bandwidth, and the event data may arrive asynchronously. At the same time, it is frequently necessary to synchronize the incoming data to an external clock reference signal. If the data is taken from analog magnetic tape, then the reference clock is often a time encoded signal which can be displayed on paper as well as stored on analog tape.

An example of such a recording situation is provided in the recording of physiological data during sleep. In this case, data is normally derived from electroencephalographic (EEG), electrocardiographic (EKG), respiratory, electromyographic (EMG), body movement, and eye movement analog channels. At the same time, event data such as the occurrence of spike discharges from single neurons are simultaneously present. In the case of sleep data, EEG, respiratory, cardiac and EMG data possess vastly different signal bandwidth requirements, ranging from 0-4 hz to 0-3khz. At the same time, the discharge of single neurons of the brain may occur on 6-8 channels at rates up to several hundred discharges/sec for each channel.

An additional requirement for sleep data is that these recordings must be collected over very long time periods ranging from 8-72 hrs. Other disciplines such as respiratory physiology and some of the physical sciences have other unique acquisition problems, but there is a common requirement of a need to accept both analog data and event data with wide bandwidth and different data rates, and a need to store these data on a mass storage device with the additional requirement to gather specific data events in reference to a independently referenced time code.

We have developed such a general data acquisition package on the LSI-11. It acquires a variety of analog and event data, and stores these data, using a double buffered technique to prevent lost samples, on a mass storage device. It also incorporates an analysis package to provide access to particular portions of the stored record, and a variety of statistical and time series packages to summarize the data.

The emphasis in program development was on flexibility in sampling and prolonged recording capability. For these reasons, considerable attention was focused on capability to accept widely different sampling rates and procedures for data compression and file manipulation. Future editions to the system will incorporate routines which are optimized for maximum throughput at fixed sampling rates.

LANGUAGE SELECTION

Because the package imposed extreme demands on timing for the LSI-11, it was essential that portions of the program run at assembler speed. At the same time, it was necessary that a large portion of the programming be in a high level language whereever possible for speed of program development, readability of source code, and ease of program modification. The system also required that the capability for additional extension of the package be a prime consideration.

For these reasons, the high level language LABFORTH was chosen for the development language. LABFORTH is a stack oriented, high level interactive language which utilizes threaded code procedures, together with a fast compiler (Harper and Sirag, 1978a,b). Its execution speed is very fast, and the interactive capabilities of the language allows very rapid develop program development. This ability to immediately modify and test routines extends to assembly language programming. Thus, even assembler code, which is an integral part of the language, can be written and tested in an on-line, interactive fashion. LABFORTH is inherently extensible; that is, new routines can be added by making a definition consisting of previously defined elements. Thus, new features to the language can be readily added, and become a permanent part of the structure. LABFORTH belongs to the class of structured languages; there is no GOTO statement, and the program flow is inherently modular.

Although there are other interactive languages which allow immediate testing of high level routines, such as BASIC or FOCAL, these languages are interpretative, and hence inherently slow. Moreover, addition of features to the lanuage, such as the incorporation of routines to access the A/D converters or parallel interface require the recompilation of the entire source language of FOCAL or BASIC. Addition of a line of assembly code into LABFORTH requires only the insertion of the line into the source text, where it is immediately compiled and becomes part of the language. FORTRAN usually has I/O routines for laboratory peripheral use, but is not interactive, and is not readily extensible. Any additional feature to the language must be compiled as a separate subroutine and linked to the main body of code. This is a process which has been standardized in versions of FORTRAN for the LSI-11, but is not convenient to use.

ACQUISITION PACKAGE

The package as implemented consists of an acquistion portion and an analysis portion. The acquisition portion has the capability of sampling up to 15 analog channels (plus the time code on the sixteenth channel), and 11 digital I/O channels.

It was essential to save as much space as possible on the mass storage device, since up to 72 hours of data needed to be recorded. For this reason, the A/D values were stripped to 8 bits.

The sampling of the A/D is under the control of the programmable clock which is ticking at 1 msec intervals. A variety of sampling rates can be used for the A/D converter which are clock multiples of this basic tick rate. At present, 8 sampling groups, each of which possesses 5 analog channels, can be set up at different clock rates. For the sleep acquisition package, the current channel set up and sampling rates are 1 channel of EKG at 256 samples/sec, 2 channels of EEG at 64/sec, 3 channels of respiration and 1 of EOG at 16/sec, and 1 channel of behavior code and 1 of motility at 1/sec. However, the configuration could have contained as many as 5 channels of data being sampled at 256 samples/sec. The operator is interrogated at setup time for the configuration of sampling rates and assignment of data channels. This configuration is printed, and is also stored in a file for later recall on analysis.

The clock count is also stored in a double precision buffer. A single precision buffer was not used, since it would overflow after 65 seconds at the 1 msec tick rate. This buffer is read after the occurence of an interupt from a digital event on the I/O card. The principle use of these data is for recording transient events, such as the discharge of single neurons. However, a very large number of these events may appear, and it is important to preserve space in recording these times of occurences as well. To aid in conservation of both

space and time, only the lower 12 bits of the time buffer are recorded, along with 4 bits of event channel information. The remaining part of the time information is recorded only when the upper 20 bits changes i.e. only after 4096 ticks. Since this occurs at 4 second intervals at the 1 msec clock rate, little time or space is used for event data, while accuracy of time between intervals is preserved.

An incrementing Binary Coded Decimal (BCD) time code which was simultaneously written on an ink writing polygraph and stored on analog tape for correlation of the two media, is decoded on the fly by the acquisition routines. The routines also read the current time as kept by the programmable clock on the double precision clock buffer, and stores both the BCD code and the clock tick time together with the acquired data. It is important to note that, because of variation either in analog magnetic tape speed or in generation of the BCD code, the BCD code may arrive asychronously. Thus its exact time of occurrence cannot be predicted, yet knowledge of its time of arrival is necessary for correlation of polygraph and digitized data. Thus, the programs must decode the BCD signal, and also note its time of arrival on the programmable clock time.

The data are stored in files on the mass storage device in record lengths that are determined at configuration set up time. At present, data are stored on floppy diskettes which are automatically swapped on filling. However, the system is also set up for serial devices such as industry compatible magnetic tapes, as well as large disk storage devices. The standard RT-11 device drivers are currently used in the system. However, special purpose drivers for even higher data rates are planned.

ANALYSIS SYSTEM

A flexible analysis package forms part of the system. The analysis package allows reconstruction of the stored data, and enables random access to portions of the recalled records.

Time and space requirements during recording precluded organization of the data for convenient analysis. Data was, however, recorded with all necessary information so that an intermediate program could, after data acquisition, organize the recorded data for rapid access and analysis. There were two such routines in the anaysis system. The first, PREANAL, creates a record directory and notes the location of each word of each record, so that the record may later be found by indexing through the directory. This technique is necessary because the records have varying lengths, caused by the varying number of triggered events. A permanent file is made of the directory. There are a total of 3 files associated with the data; the configuration file (.CON), the directory (.DIR), and the data (.DAT).

The second data organizing routine is called READC. It is used at the beginning of each analysis session to read the configuration file, the record directory, and the 0th data record. It uses the configuration file to simulate the recording of sample data in the record. However, instead of recordng data, it notes the location into which the data would be stored. Using this technique, a sample map is created. The map is organized by channel number; thus, all data from a given channel can be collected by using the sample map for the channel to locate the data. Various routines then are available which use the above data representaion. The COLLECT routine gathers the data from the specified channel (using the sample map), converts it if necessary to floating point, and stores the data into an array. These procedures essentially allow random access to individual elements of sampled data, even though they may be located on a serial device. Once in the array, the data can be manipulated by a number of statistical or anaytical routines, such as time series analysis routines.

A variety of analytical packages have been incorporated into the package, including FFT routines and point process programs. This collection of programs is being steadily enlarged as the package develops.

COMPARISON WITH OTHER LANGUAGES

At this point, the relative merits of writing the package in LABFORTH as opposed to a combination of FORTRAN or Assembly language routines can be evaluated. There is for this evaluation a point of comparison, because a similar package had been developed for the PDP-12 computer some years earlier (Harper et. al., 1974; Mason et. al., 1974; Pacheco et. al., 1974; Harper and Mason, 1978). This previous package was developed primarily for sleep analysis and was written in FORTRAN with assembly language subroutines, and in PAL-8. That package cost approximately 7 man years of development time. Six months of that time was devoted to the analog to digital conversion routines. The equivalent routines in LABFORTH, which additionally included recovery and random acess utilities, required 6 weeks in LABFORTH. Program development time of other routines is more difficult to estimate, since some aspects of the analysis portions have changed between the two systems. However, on selected routines, development speed has been between 4-10 times faster in LABFORTH than with the previous package.

A principle advantage of programming in the LABFORTH language is the degree of capability for expanding the system. Additional language features for new I/O devices or new mass storage devices, or new display devices can be readily added to the language with no recompilation of previous source code.

The resulting LABFORTH code is much more compact than the previous package. The PDP-12 package occupied several hundred pages of source code bound in a 4 inch thick volume. The acquisition package alone occupied over a 100 pages of assembly source. The LABFORTH acquisition routine occupies 6 pages of source code. This small number of pages required for the source code greatly facilitates program maintenance.

The LABFORTH version of the program runs very fast. Where speed is of highest priority, assembly language code was incorporated into the LABFORTH source. The high level code is also very fast in execution, and exceeds the speed of FORTRAN source code in such analytical routines as the Fast Fourier Transform. Use of the stack features of LABFORTH is instrumental in providing fast execution.

Because the program is modular, memory size is not as significant a problem in this task, except where very large files are required. however, the very large routines are typically ocupying 10 to 30 % less space than equivalent assembly langage and FORTRAN routines. This largely results from the inherent tendency to reuse routines which are already defined in LABFORTH, rather than executing IN-LINE code. As a result, programs tend to grow linearly in LABFORTH , rather than exponentially as with other languages.

The key to the success of LABFORTH as applied to this acquisition and anaysis package is that it is sufficiently fast and flexible to perform tasks which previously could only be carried out in assembly language. However, snce it is a high level language, and, more important, an inteactive high level language, program development is very fast. The earlier version of this package required portions of the system to be written in assembly language, and portions to be written in a high level language, i.e. FORTRAN. The result was a non-integrated collection of individual programs that were developed at great expense, and was difficult to maintain. This version in LABFORTH, however, was developed quickly, and resulted in an integrated compact package that forms a readily maintainable system which is fast in execution.

SUMMARY

An acquisition and anlysis package has been developed for gathering analog and digital data and storing these data onto a mass storage device. The acquisition routine has been written in LABFORTH, a high level interactive language with stack. features that allows fast execution compared with a similar package written in FORTRAN and Assembly language. Compared with the older package, this system was developed faste r, runs more quickly, and is more extensible than the previous version.

BIBLIOGRAPHY

1. Harper, R.M. and Sirag, D. J. LABFORTH: Replacing BASIC and FORTRAN with a fast interactive language . NCC Pers. Com. Digest,pp 245-250, 1978

2. Harper, R.M. and Sirag, D.J. LABFORTH: A Fast Interactive Language for the LSI-11. Proc. Digital Equip. Comp. Users Soc., 4, 4, pp. 957-961, 1978.

3. Harper R.M. and Mason, J. A Computer System For The Analysis Of Physiological Data During Sleep. In Press.

4.Mason, J. R. Harper, R.M., and Pacheco, R.F. Analysis of respiratory data during sleeping and waking. Proc. Digital Equip. Users Soc. 1, 2, pp. 567-571, 1974

5.Pacheco, R.F., Perga, A., and Harper, R.M. Time series analysis of physiological data during sleep and waking. Proc. Digital Equip. Users Soc. 1, 2, pp 551-556, 1974.

6.PDP and FOCAL are registered trademarks of Digital Equipment Corporation. LABFORTH is a trademark of Laboratory Software Systems, which developed the software described here.

TIME SHARE TERMINAL EMULATOR UNDER RT-11

T.L. Starr and L.T. Nieh
General Electric Company
Louisville, Ky. 40225

ABSTRACT

TSTE is a program which enables a PDP-11 under the RT-11 operating system to communicate with a remote computer, e.g. G.E. Mark III or DEC-10, over a telephone line at low data rate, 110 or 300 baud. Basic hardware requirements are a DL-11E asynchronous communication interface, and a Bell 103 or equivalent data set. The program normally operates in terminal mode where the PDP-11's console device appears to be a time share terminal to the user, and the PDP-11 emulates the functions of a time share terminal to the remote computer. In addition, the user can transmit an ASCII data file from the system RK05 disk, or store data received from the remote computer on disk. The program further provides a control character to generate a BREAK signal to the remote computer, and a control shift character which allows any control character to be transmitted to the remote computer. The program will be made available through the DECUS program library.

INTRODUCTION

Out of tens of thousands of PDP-11's in use today, most are self-standing systems that operate without the benefits of being able to communicate with a remote computer or terminal. This paper presents a computer program, Time Share Terminal Emulator (TSTE), that enables a PDP-11 operating under RT-11/V2C Single Job Monitor to communicate with the outside world while imitating a time share terminal. The program transmits ASCII data at 300 baud, and requires moderately priced communication devices - i.e., a DL-11E interface and a Bell 103 equivalent data set. Using this program, the PDP-11 is turned into a super intelligent terminal that supports a variety of computation and communication activities. Some of the applications that we find useful are —

1) transmission of locally generated finite element model data for analysis on a remote computer.

2) transmission of remote computation result back to the PDP-11 for post-processing.

3) transmission of locally edited program and data files to a remote computer.

4) transmission of data and result files to a remote terminal.

5) transmission of programs between PDP-11's.

HARDWARE REQUIREMENT

1) PDP-11 Computer Supporting RT-11V2C
2) DL11-E Asynchronous Interface
3) Bell 103 Compatible Data Set
4) Console Device - 30 cps or faster
5) RK11/RK05 Disk

The program TSTE is currently operating on a PDP-11/10 with 28K words memory, a DL11-E interface, a Prentice DC22 data set, a Tektronix 4010 as console device, and a RK11/RK05 disk. The DL11-E is installed according to the DL11 Asynchronous Line Interface Engineering Drawing. For communication with G.E. Mark III time sharing service at 300 baud, the following jumper selectable options on the DL11-E are used:

7 data bits
even parity
1 stop bit
300 baud transmit and receive

The 25 ft. cable supplied with the DL11-E connects it with the data set. The Prentice DC-22 Data Coupler is a self-contained modem compatible with the Bell System 103A Data Set. Data coupler options are selected as follows:

RS-232C Interface
Data Access Arrangement - CDT-1000, Manual Dial Originate
Half-duplex

The DAA option provides a data button on the telephone instrument to connect the data coupler with the phone line.

Since the console device echoes all transmitted and received characters, it must operate at a rate equal to or faster than the DL11-E interface in order to support sustained data transmission. The Tektronix 4010 operates at 9600 baud, however, a LA36 Decwriter II operating at 30 cps probably would work also. The RK11/RK05 disk is used to store data files that are either transmitted or received from the remote computer.

SOFTWARE REQUIREMENT

TSTE is a MACRO-11 program that operates under RT-11/V2C Single Job Monitor. It uses less than 4K words of memory. The source program and write-up will be available from the DECUS PDP-11 library.

MODES OF OPERATION

Terminal Mode

This is the default mode of operation. The user communicates with the remote computer through the console device. User inputs at the console keyboard are transmitted to the remote computer, while outputs from the remote computer are printed on the console printer. Since the data set operates in half duplex, the user and the remote computer take turns in sending data. User can generate a BREAK signal by typing a CTRL K character. With G.E. Mark III time share service, type CTRL K once to issue a BREAK while the user is inputing; type CTRL K twice to issue a BREAK while the remote computer is outputing.

File Transmission Mode

The user enters the File Transmission Mode by typing a CTRL T character. TSTE reads the file RKØ:OUT.DAT and transmits its contents to the remote computer. The data set again causes the console printer to print the characters as they are transmitted. In essence, file transmission is analogeous to using the paper tape reader with a conventional time share terminal. To assure compatibility with paper tape input format and timing requirement, TSTE provides rubout fill characters following each line feed character.

File Receive Mode

The user initiates the File Receive Mode by typing a CTRL R character. TSTE opens a file RKØ:IN.DAT and writes all the subsequent console dialog in that file. However, the user must type a CTRL D character to close the file in order to make it permanent before exiting from TSTE. Otherwise, RKØ:IN.DAT will not appear in the file directory. File receive is analogeous to using the paper tape punch with the conventional time share terminal.

Control Shift

Certain control characters, e.g. CTRL C and CTRL R, are recognized and intercepted by the RT-11 keyboard monitor or TSTE, thus they cannot be sent to the remote computer from the keyboard. TSTE provides a control shift character, CTRL N, which when followed by an alpha character signifies that it is to be sent as a control character. For example: a CTRL N followed by an X would send a CTRL X (line delete for G.E. Mark III) to the remote computer. Since most control characters are not printable, the echo will not appear on the console printer. This feature allows TSTE to communicate with a PDP-10 remote computer which shares common control characters with the PDP-11.

PROGRAM MESSAGES

Normal Messages

All TSTE messages are preceeded by a "#" sign.

1) On start-up, TSTE prints

    # READY FOR REMOTE COMPUTER

2) On entering file transmit mode (CTRL T), TSTE prints

    # TRANSMIT OUT.DAT FILE

3) On leaving file transmit mode, TSTE prints

    # OUT.DAT FILE TRANSMITTED

4) On entering file receive (CTRL R), TSTE prints

    # OPEN IN.DAT FOR INPUT

5) On leaving file receive mode (CTRL D), TSTE prints

    # IN.DAT CLOSED

Fatal Errors

Four fatal error conditions cause TSTE to abort and return to RT-11 monitor. TSTE prints an error message and exits to RT-11.

1) Fetch RK disk handler error

    # DEVICE NOT FOUND

2) Open file error - file receive or file transmit mode

    # FILE OPEN ERROR

3) Write error - file receive mode

    # FILE WRITE ERROR

4) Read error - file transmit mode

    # FILE READ ERROR

Non-Fatal Errors

Two non-fatal errors cause TSTE to print a warning message, but to continue processing.

1) Attempt to open an already opened file - file receive mode

    # FILE ALREADY OPENED

2) Attempt to close an already closed file - file receive mode

    # FILE ALREADY CLOSED

Normal Exit

TSTE normally exits by user typing two CTRL C's.

Dynamic Operation with TSTE

Since the remote computer considers TSTE to be a time share terminal, the remote computer will wait for input for a reasonable amount of time before terminating the session due to time out. Consequently, the user can exit from TSTE, by typing two CTRL C's, perform some local file manipulations or computations, and then re-run TSTE to resume remote processing. This dynamic enter/exit capability allows a remote processing session to send and receive a number of files taking advantages of computing capabilities of both the PDP-11 and the remote computer. By switching the data set in receive mode, TSTE communicate with a remote terminal or another PDP-11 which is calling in with its data set in the originate mode.

PROGRAM LIMITATIONS

TSTE relies on asychronous data communication, thus there is no provision for transmission error detection. However, processor-to-processor data transfer eliminates an intermediate data storage medium such as paper tape or cassette while suffering equally from data transfer errors. One solution to detect transmission errors is to have the remote computer send back the transmitted file, and then do a file comparison locally.

TSTE has not been used at 1200 baud, although this has been limited by the hardware, i.e., the data set and the console device.

BEIN11: ON-LINE BEHAVIOR INPUT

Stephen Walker and Martin Reite
University of Colorado Medical Center
Denver, Colorado

ABSTRACT

BEIN11 is a FORTRAN program for computer entry and editing of primate behavioral observations as they occur using an on-line LSI-11 microcomputer. BEIN11 permits real time data collection, bypassing traditional event recording techniques like audio tapes and checklists. Additional advantages are accurate entry timing, automatic error checking of the entries, a real-time display of previous entries and currently active entries, and the ability to edit the entire behavioral entry sequence immediately after the session is terminated.

Entries are structured around a focus subject, who is the originator or recipient of all behaviors scored. The entries themselves are one or two character mnemonics representing specific predefined behaviors taken from a suitable taxonomy and a 4-digit number describing any other animal involved in the specific behavior. The program keeps track of when behaviors are entered and displays the currently active behaviors on the CRT terminal to give the observer a time-locked summary of what has happened and is happening. The program automatically times to a preset length then displays the entire sequence of entries with time of occurrence before the session is appended to the focus subject's data file. The program includes an expandable library of 64 different possible behavior entries. The program is adaptable to monitor and quantify real time behavior observation from any species for whom a taxonomy can be developed.

BEIN11 is a PDP-11 FORTRAN program designed to permit real time collection of behavioral data from animal subjects (we presently use it for infant monkeys) living in social groups. One or two character symbols representing specific behaviors are entered on a key board terminal as they occur, and active entries (those not terminated and lasting more than 1 sec) are displayed on a CRT. The repertoire of behavioral entries is based on a single animal subject called the focus who is defined before the behavioral session is started, and to whom or from whom all behavior entries are directed. Most behavior entries can be electively modified with a 4-digit number representing another animal that might be involved in a specific behavior. For example, an entry indicating the focus subject is being groomed can be modified to state who is doing the grooming. The behaviors are taken from a behavior taxonomy appropriate for the species, and the human observer observes the subject for a finite period of time (preset to 3 or 5 minutes but variable) called an observation session. At the end of a session, the program stores the data so as to provide duration (with absolute start and stop times), frequency, and sequence of all entries.

The behavior taxonomy library contains 64 items divided in 5 Subcategories designed to facilitate the observer's behavior entry. Table 1 lists the present library. Subcategory 1 contains entries which define the focus animal's physical relationship with its mother. One of these entries is always active, and any subsequent entry from Subcategory 1 automatically stops the previous entry. Subcategory 2 defines the focus animal's physical activity level; again one of these entries is always active, any subsequent entry of a member of Subcategory 2 replaces the current entry. On the CRT display, entries in both Subcategories 1 and 2 cause their display column to be replaced with the latest entry. Subcategory 3 contains entries whose active length of time can be defined (called duration items). Up to 6 of these entries may be displayed as active at one time. Entries in Subcategory 3 are stopped by reentering the exact entry. Subcategory 4 contains entries whose active duration is very short (called frequency items) and are assigned a 1 second duration. Subcategory 4 has one display column on the CRT. Subcategory 5 is for control commands and behavioral entries which are undefined. These entries are displayed in a separate column and are written to the data file. This Subcategory is useful for defining new behavioral entries not presently in the entry library, and serves as an error detector, since all entries not in Subcategories 1 through 4 automatically appear in Subcategory 5.

SUBCATEGORY 1:

K = Cradle by Ma
E = Enclosed by Ma
S = Passive support by Ma
T = Contact with Ma
X = Proximity to Ma
A = Apart save level
V = Apart other level
Z = Mother separated

SUBCATEGORY 2:

R = Quiet rest
J = Active rest
M = Movement
C- = Carried by -
L = Locomotion
P- = Play with -

SUBCATEGORY 3:

K- = Cradle by - (Not Ma)
E- = Enclosed by - (Not Ma)
S- = Passive Support by - (Not Ma)
T- = Contact with - (Not Ma)
X- = Proximity with - (Not Ma)
N = On the Nipple
BE = Bedding Exploration
DR = Drink
EA = Eat
F = Fight in pen by others
IA- = Initiate Aggression on -
G- = Initiate Groom on -
IM- = Initiate Mount on -
IN- = Initiate Genital Exploration on -
IO = Inattentive Object Exploration
MI- = Ma initiate aggression on -
MM- = Ma receives mount from -
W- = Ma receives aggression from -
OE = Object exploration
OO = Oral object exploration
OS = Orality on self
RA- = Receive aggression from -
RG- = Receive groom from -
RM- = Receive mount from -
RN- = Receive genital exploration from -
RS- = Restrain by -
SL = Slouch
TT = Temper Tantrum

SUBCATEGORY 4:

- = Activity count
D- = - approaches focus
B = Convulsive jerk
O = Coo
I- = Initiate play to -
Q = Squeal
IT- = Initiate threat to -
IU- = Initiate punishment to -
IX- = Initiate social exploration to -
Y- = - leaves focus
H = Mark for edit reference
ND = Nipple denied by Ma
NO = Noise from outside pen
NP- = No response to play overture from -
RP- = Receive play overture from -
RT- = Receive threat from -
RU- = Receive punishment from -
RX- = Receive social exploration from -
SK = Scratch self
ST = Startle reaction
SU- = Submit to -
WE = Wean

SUBCATEGORY 5:

START
EDIT
SAVE
OTHER ENTRIES

TABLE I

A listing of the monkey behavior taxonomy currently used with BEIN11. Items followed by a dash (-) can be optionally modified with a 4-digit identifier (See text for discussion).

Operation of BEIN11 is best explained by a step by step description. After the program is loaded the computer prompts the observer for the focus subject's number, the observer's name, the experimental condition, and the number of the subject's group. After entering this data the program asks for initial conditions or entries which are active at the time the session is started. Subcategories 1 and 2 always need initial conditions; Subcategories 3 and 4 may or may not have any active behaviors. BEIN11's commands are START for beginning the behavioral sessions, EDIT for editing the entire behavioral entry sequence, and SAVE for appending the most recent behavioral sessions to a particular subject's data file.

The START command enables the timer, writes a start session entry in the data, and requests the current date and time of day from RT11 and stores them with the data. The program is now in operation awaiting new behavioral entries as it times the session. Each behavioral entry causes the time of occurence to be recorded. Next the entry is split into the mnemonic and optional modifying animal number. The mnemonic is compared with the behavioral entry library to determine its Subcategory. The new behavioral entry is then inserted into the proper CRT display column. If the entry is from Subcategory 1 or 2, it replaces the existing entry on the display. If it is from Subcategory 3 this display area is searched to determine if the new behavioral entry is a terminator for an active entry or the start of a new behavior. If it is from Subcategory 4 or 5 it will be displayed once on a single line and need not be terminated. Following each behavioral entry a single line 80 characters in length is then output to the CRT terminal. This line contains the time of occurence (measured in seconds from start of session) of the entry, as well as

all currently active behaviors. BEIN11 is then ready for another entry. At the end of the session time, an end of session entry is written to the data area with the length of the session included (3 or 5 minutes), and the end of session message appears on the CRT terminal.

After typing EDIT, the CRT Terminal prompts for a comment whose form is unrestricted other than a maximum 60 character length. This provides an opportunity for the observer to type in any special descriptive circumstances accompanying the observation session just completed. The entire session is then displayed along with time of occurence of each entry; each entry is also assigned an entry number for editing purposes. Typing E [entry number] causes the first 50 entries after the entered entry number to be displayed. Existing entries may be replaced by typing the entry number followed by the new text. Typing D [entry number] causes that entry to be deleted and the following entries to be moved up one. Typing I [entry number<sp>new entry] inserts a new entry before the specified entry number.

Typing SAVE causes the program to search the storage device for the already specified animal's file and append the just edited session to it. The program is then ready for a new observation session. Data are stored in the ASCII form seen during editing, and are easily deciphered for later analytic and statistical routines.

This program can be down line loaded to run on a satellite LSI-11 freeing up the host. This is the way we currently run it using a satellite Heath H11 microcomputer with an H9 video console remote from the main PDP11T03 system. The program is being expanded to include acquisition of concurrent physiological data. Library additions, deletions, changes in session time, or changes in entry grammar are easily implemented by a competent FORTRAN programmer.

Acknowledgements: This research is supported by USPHS Grant No. MH19514. M. Reite is supported by NIMH Research Scientist Development Award No. 5K02MH46335.

ATOMIC ABSORPTION SPECTROMETER READOUT AND DATA REDUCTION USING THE LSI-11 MICROCOMPUTER

Michael J. Allen and Robert W. Wikkerink
Lawrence Livermore Laboratory
Livermore, California

Some common instruments found in the chemistry laborotory have analog chart recorder output as their primary data readout media. Data reduction from this medium is slow and relatively inaccurate. This paper describes how to interface a single LSI-11 microcomputer to PERKIN-ELMER models 603 and 303 Atomic Absorption Spectrophotometers.

DEFINING THE PROBLEM

Due to the large volume of samples that are processed through our laboratory and the excessive number of manual steps required for data reduction, there is a great need to automate the process so that the chances for errors and the time the operator spends running the machine are minimized. In our laboratory there are two PERKIN-ELMER Atomic Absorption Spectrophotometers. A Model 603 with LED digital readout and a Model 303 with chart recorder readout. When using the 603, the operator manually copies numbers from the LED display while the sample is being asperated. The hand gathered data is then reduced to parts-per-million or micrograms-per-milliliter by using a calculator. The Model 303 on the other hand, requires the operator to make quantitative measurements by measuring the peak height from traces on chart paper either manually or by analog computer.

GOALS

A. Automatically generate card image ASCII files for use in later calculations. These files to be output to paper tape or disk.

B. Generate printed pages suitable for storage in a logbook.

C. Free the chemist from tedious manual data gathering and reduction.

SYSTEMS EVOLUTION

Since the PERKIN-ELMER Model 603 is the primary data gathering instrument, this paper will be mostly concerned with describing the program and the hardware that was required to interface the LSI-11 to it. Current plans for the Model 303 will be discussed in part.

The program was first written in FOCAL, then in assembly language to run in an available LSI-11 with 4K core memory and an ASR-33 Teletype for I/O. The LSI-11 has since been upgraded such that it now has 28K memory, RT-11, dual floppy disks, EIS/FIS chip, and a TI silent terminal. In addition BASIC is now running on the system and a similar AA program has been written in the BASIC language.

IMPLEMENTATION CONSIDERATIONS

A. FOCAL: Too slow, no mass storage capability without RT-11, 4K memory is barely adequate, and grossly inadequate string handling.

B. MACRO-11: Difficult to make changes in program, has mass storage capability, 4K memory is adequate, very immune to 'glitches', and adequate string handling.

C. BASIC-11: Requires 16K memory minimum, good mass storage capabilities, excellent string manipulation, and the program may be easily changed.

D. HARDWARE: The PERKIN-ELMER communications interface provided a suitable RS-232-C serial interface port for connection to the LSI-11 microcomputer running at 1200 BAUD.

As soon as 16K memory, dual floppy disks, and RT-11 became available the FOCAL version was abandoned and work proceeded on the MACRO-11 and BASIC-11 versions.

OPERATIONAL MODES

The operational modes available include BEGIN, RESET, MANUAL, MANUAL BATCH, DUMP, and TAPE. A detailed discussion of each operational mode follows:

A. BEGIN: BEGIN mode provides for manual entry of certain parameters which do not change for a given sample set. Examples: Element designation, sample volume, procedural blank, procedural blank code, dilution factor, and mass-unit code. The BEGIN mode also permits the operator to select for input either ABSORBANCE or CONCENTRATION data.

B. RESET: RESET mode provides for the modification of the dilution factor parameter, entered in the BEGIN mode, without entering or modifying any other parameters associated with the BEGIN mode.

C. MANUAL: The MANUAL mode provides for manual entry of the solution identification number, amount of sample, the sample name, and the number of aliquots required for this analysis. In addition, the operator may also elect to utilize "Method-of-Additions"

in which case the computer will request the number of data points required. Two through five data points are acceptable. The terminal requests '0:' to indicate that the sample requested has no 'spike' added to the sample being analyzed. After processing the '0:' data, the terminal requests '1:' data and so forth until all points have been entered

Using two points, the algorithm follows the standard form. For three or more points, the computer performs a least-squares fit to a straight line.

D. MANUAL BATCH: This mode is identical to the MANUAL mode except that the operator is interrogated only once for the initial values with the additional request to input the total number of samples to analyze.

The hardcopy device prints all data in the same form as for MANUAL mode and is suitable for logbook entry. The computer treats the solution identification as a floating point number and increments it by one for each sample. The low three characters of the sample name must be digits and are converted to an integer in order to be incremented by one for each sample analyzed.

Under MANUAL BATCH mode, the computer outputs the usual information for the log page and awaits data from the Model 603 AA unit. Method-of-Additions analyses are handled likewise with no operator communication with the computer. The MANUAL BATCH mode is suitable for sample changer operation if desired.

E. DUMP: The DUMP mode puts the final data in card image format onto the floppy disk (MACRO-11 version only). The result is an ASCII file which may be entered into a database. The BASIC version automatically outputs card images in a sequential disk file.

F. TAPE: The TAPE mode punches the final data in card image format onto paper tape, on the ASR-33 terminal, for storage or entry into a database.

The program mostly operates in the two modes: BEGIN and MANUAL.

OPERATIONAL DESCRIPTION:

In all modes where interaction between the operator and the computer is required, the RUBOUT character deletion command is allowed. In the MACRO-11 version, RUBOUT echoes a backslash for each character that is deleted. If more characters are deleted than are actually in a given field the program ignores them. The BASIC version operates with RUBOUT as described in the BASIC manual.

The program must be initialized in the BEGIN mode as the MANUAL mode requires that the BEGIN mode parameters be previously entered.

MACRO-11 VERSION RESTRICTIONS: Any time the TAPE mode is called, the program must be restarted in BEGIN mode as the data buffer pointer is reset at the end of the tape punching sequence, effectively eliminating the card images from the data buffer. While in the MANUAL mode, the LSI-11 waits for the PRINT button on the AA unit to be activated, collects the readings until the requested number of aliquots has been satisfied, prints out a header on the terminal, calculates and prints micrograms-per-ml, standard deviation, net micrograms-per-ml, and fractional standard deviation for the sample currently being analyzed.

The SAMPLE ID, SOLUTION ID, net micrograms-per-ml and FSD are written to the storage buffer in strings of ASCII characters. The program then prints the MODE prompt again. At that point the operator can return to the BEGIN mode to change parameters, continue with the MANUAL mode for the next sample or go to the TAPE mode for outputting the stored data to paper tape via the terminal low speed paper tape punch.

The program allows the operator to analyze samples in CONCENTRATION or ABSORPTION by changing the mode switch on the AA unit and by answering the absorbance question that is asked in the BEGIN mode.

The operator may also elect to do standard additions whenever sample matrix problems are suspected. The program allows two to five point standard additions depending upon the operators preference.

With the addition of RT-11 and dual floppy disks, the MACRO-11 program was modified slightly to allow the operator to write the memory stored data to the default device with the DUMP mode. In the DUMP mode the program requests a file name and then writes the file with a .DAT extension. At the end of the write, the program prints ALL DONE and exits to monitor so the operator is forced to restart the program in the BEGIN mode.

In the RT-11 version buffer space has been reserved for 200 samples before the data must be dumped to the floppy disk whereas in the non RT-11 version the maximum allowable buffer space is limited only by the size of memory. In the non RT-11 version, if the operator asks for the DUMP mode instead of TAPE mode, the program ignores the dump request and outputs the memory buffer to the low speed paper

tape punch. In the RT-11 version either dump to DK: or tape is allowed. This is done with conditional assembly code so the program is useable on either LSI-11 system without major program modifications. The data that is transferred to DK: with the DUMP mode are written in ASCII format for editing purposes.

BASIC-11 FILE STORAGE SCHEME: BASIC-11 is programmed to open a sequential file which has been previously dimensioned as a string array. When BASIC is first started, a filename is requested and BASIC creates a null file of that name with a .DAT extension. As each sample is analyzed in the MANUAL mode, BASIC opens the sequential file, reads the strings into an array, adds the next string element to the array, then closes the sequential file. The constant updating of the sequential file provides for the preservation of data in the event of a power failure. However, repetitively updating the sequential file creates imbedded empty files on the device directory which may, in turn, create housekeeping problems on small floppy disk systems. The periodic use of the RT-11 .SQUEEZE command may prove necessary in such cases.

EXAMPLES

A. MACRO-11: Figure 1 shows an example of BEGIN mode parameter entries. Note that a carriage-return simply retains the previous value of a parameter.

```
MODE:  B

ABSORBANCE? N
STANDARD ADDITIONS? Y
SPIKE CONC. IN PPM: 2.0

ELEMENT:    : NM
VOLUME: ******* : 1
BLANK: ******* :
BLANK CODE:  : 13
DILFACT: ******* : 1
SASAMP: ******* : .01
MUCODE:  : 10
```

(Figure 1)

Figure 2 (See Page)

Figure 3 shows how the operator uses the DUMP mode to transfer the data to a file on the default device. Note that the "=" sign is required.

```
MODE: D

AT THE ASTERISK TYPE FILE NAME IN
THE FOLLOWING FORMAT.
FILENM=
6 CHARACTER FILENAME IS MAXIMUM

PWL=

ALL DONE.
```

(Figure 3)

Figure 4 shows an example of how easily the dilution factor can be changed in the RESET mode.

```
MODE: R

DILFACT: 100.000 : 10

MODE:
```

(Figure 4)

B. BASIC-11: Figure 5 shows an example of parameter entries in the BEGIN mode. Note that in order to retain the previous value of any parameter, it must be retyped as the operator progresses through the BEGIN mode. The RESET mode may be used to change the dilution factor without disturbing or retyping any other parameter.

```
ENTER FILE NAME (6 CHARS. MAX) ABC123
NEW FILE? Y

MODE: BEGIN

ABSORBANCE? Y
STANDARD ADDITIONS? N
STANDARD CONCENTRATION IN PPM:5
ABSORBANCE OF STANDARD: .2

ELEMENT: :CA
VOLUME: 0 :1
BLANK: 0 :0
BLANK CODE: :15
DILFACT: 0 :1
SDSAMP: 0 :.001
MUCODE: :13
```

(Figure 5)

Figure 6 illustrates the RESET mode.

```
MODE: RESET

DILFACT: 1 :2.5

MODE:
```

(Figure 6)

Figure 7 (See Page)

Figure 8 (See Page)

Figure 9 illustrates a card image file dump from either the MACRO-11 or the BASIC-11 versions.

700036	PWL027	SR10	19.6111	.000633
700037	PWL029	SR10	13.3055	.000934
700038	PWL030	SR10	1.38888	.020000
700039	PWL031	SR10	8.27777	.001501
700040	PWL036	SR10	7.94444	.001564
700041	PWL039	SR10	7.88888	.001575
700021	PWL045	SR10	19.7777	.000628
700043	PWL085	SR10	12.8055	.000970
700044	PWL087	SR10	9.36111	.001327
700045	PWL075	SR10	6.76666	.001836
700046	PWL079	SR10	7.62222	.001630
700047	PWL081	SR10	9.43333	.001317
700048	PWL086	SR10	5.75555	.002158
700049	PWL030	SR10	52.2222	.000238
700050	PWL035	SR10	25.5555	.000486

(Figure 9)

SYSTEM PERFORMANCE

The Model 603 AA unit and the LSI-11 have been operating together for approximately one year now. A comparison was made between the old system of hand collecting and calculating the data versus letting the LSI-11 do it. Before automating the system, it took three to five minutes to collect and hand calculate one sample. After automating it takes 40 seconds to analyze and machine calculate the same sample. The whole analytical process has been speeded up by a factor of 4 to 7 which is a significant improvement.

FUTURE PLANS

Since the addition of RT-11 and dual floppy disks to the system BASIC has been added. Future plans call for the addition of the PERKIN-ELMER Model 303 and a DIONEX SYSTEM 10 Ion Chromatograph both of which have chart recorder output. This will be accomplished by adding an A/D converter to the LSI-11 and utilizing analytical programs designed to operate under DEC'S MULTI-USER BASIC.

ACKNOWLEDGMENT

Many thanks to Bob Heft who provided the inspiration and support for this project and to Bill Steele for his many practical suggestions. A special note of thanks to co-author Bob Wikkerink who wrote the MACRO-11 version of this program for without his help, this paper would not have been written.

Work performed under the auspices of the U.S. Energy Research & Development Administration, contract No. W-7405-Eng-48.

Figure 2 shows two examples of MANUAL mode operations. The first is for a sample using CONCENTRATION values for data input. The second sample was analyzed using two-point Method-of Additions. Note that all floating point numbers are represented in a six-digit format. Note also that the MANUAL BATCH mode is not available.

```
MODE: B

ABSORBANCE? N
STANDARD ADDITIONS? N
ELEMENT:  : CA
VOLUME: ******* : 1
BLANK: ******* : 0
BLANK CODE: : 15
DILFACT: ******* : 1
SDSAMP: ******* : .001
MUCODE: : 10

MODE: M

SOLUTION ID: 700049
AMOUNT: ******* : 1
SAMPLE ID: NUR004
BLANK CODE: 15
HOW MANY ALIQUOTS? 5
VOLUME:  1.00000
!!!!! START AA UNIT!!!!!

4.56  4.62  4.63  4.71  4.64

ELEM.       DILFACT       UG/ML         S.D.          BLANK       NET UG/10       S.D.          F.S.D.
CA          1.00000       4.63200       .000447       *******     4.63200         .000447       .000097

MODE: M

SOLUTION ID: HCC010
AMOUNT: 1.00000 :
SAMPLE ID: HCC010
VOLUME: 1.00000
BLANK CODE: 13
HOW MANY ALIQUOTS? 4


STD ADDITION:  0

  0.06  0.06  0.07  0.06

CONCENTRATION: .062500       FSD: .005000

STD ADDITION: 1

  1.04  1.02  1.02  1.02

CONCENTRATION:  1.02500      FSD: .005000

SAMPLE CONC: .129870

ABSOLUTE SIGMA: .000954

RETAIN STD ADD'N NUMBER ? Y
```

(Figure 2)

Figure 7 shows examples of MANUAL mode operations. Note the date and time are automatically printed by the computer for logging purposes.

```
MODE: M

SOL ID: 700002
AMOUNT: 1
SAMP ID: NUR004

VOLUME: 1

BLANK CODE: 15

HOW MANY ALIQUOTS FOR RUN # 2  :3
   0.002     0.001     0.002
AVERAGE MACHINE READING:    0.0017

TODAY IS 24-OCT-78 THE TIME IS 15:18:10

ELEM.       DILFACT       UG/ML         S.D.          BLANK         NET UG/10     S.D.          F.S.D.
CA          10.E-01       4.167E-02     1.44E-02      0.0           4.167E-02     1.44E-02      3.46E-01

MODE: MB
```

(Figure 7)

Figure 8 illustrates the MANUAL BATCH mode of operation. Note that the operator answers only initial values for solution identification, and sample identification. The computer continues from there for the number of samples designated. The operator need only operate the PRINT button on the Model 603.

```
MODE: MB

HOW MANY SAMPLES IN THIS BATCH? 2
SOL ID: 700003
AMOUNT: 1
SAMP ID: NUR104

VOLUME: 1

BLANK CODE: 15
HOW MANY ALIQUOTS FOR RUN # 3  :3
HOW MANY POINTS TO FIT? 2
STD. ADDITION: 0   0.002   0.001   0.001
AVERAGE MACHINE READING:   0.0013

STD. ADDITION: 1   0.017   0.017   0.017
AVERAGE MACHINE READING:   0.0170

TODAY IS 24-OCT-78 THE TIME IS 00:06:09

NET SAMPLE CONCENTRATION: 4.255E-01 UG/ML

ABSOLUTE SIGMA: 3.47E-04

SOL ID: 700004
AMOUNT: 1
SAMP ID: NUR105
VOLUME: 1
BLANK CODE: 15
HOW MANY ALIQUOTS FOR RUN # 4  : 3

HOW MANY POINTS TO FIT? 2
STD. ADDITION: 0   0.001   0.000   0.001
AVERAGE MACHINE READING:   0.0007

STD. ADDITION: 1   0.016   0.016   0.016
AVERAGE MACHINE READING:   0.0160

TODAY IS 24-OCT-78 THE TIME IS 00:08:05
NET SAMPLE CONCENTRATION: 2.174E-01 UG/ML
ABSOLUTE SIGMA: 1.77E-04

MODE:
```

(Figure 8)

APPLICATION OF MU-BASIC, VIRTUAL FILES TO MARINE CHEMICAL RESEARCH

George Kerr
Harbor Branch Foundation, Inc.
Fort Pierce, Florida

ABSTRACT

A PDP 11/34 system has been set up to reduce data resulting from marine chemistry analyses. A series of MU-BASIC programs operating under RT-11 has been developed providing the following virtual file applications:

1. Building and editing input data files.
2. Selection of data used in various statistical subroutines.
3. Storage and retrieval of reduced data in a master file.

The advantages and disadvantages of virtual file utilization on a small (32K word), floppy based system are illustrated.

INTRODUCTION

Marine chemical research usually entails analysis of numerous seawater samples for nutrients, metals, and particulate materials. Laboratory procedures are generally complex, tedious, and slow. However, sophisticated analytical instrumentation has been developed, and has found acceptance as a reliable and productive means to increase the efficiency and reduce human error of such procedures. Automatic methods free the investigator for less routine tasks and provide more rapid and precise determinations [1,2].

The amount of data resulting from these analyses is large, and is usually gathered by hand, organized into consistent groupings, and then reduced by routine statistical methods. With automated laboratory procedures, this is perhaps the most time consuming aspect of chemical oceanography. It is also the most amenable to computer applications, as has been realized in biomedical research laboratories [3]. The sheer volume of data and the number of computations involved in routine data reduction for interpretation of experimental results have compelled the marine chemists at Harbor Branch Foundation, Inc. to request the development of an automated data collection and reduction system.

In an effort to determine long-term trends, the marine chemistry group routinely samples several locations within the lagoon to determine the concentration of dissolved nutrients, heavy metals, and organic carbon. In addition, several physical parameters such as water temperature, salinity, and tide are monitored.

Samples returned to the laboratory are processed through several types of chemical analytic equipment. In each instance an ordered sample tray is prepared for analysis. Technicon AutoAnalyzers® are used for nutrient determinations. Heavy metal determinations are accomplished by anodic stripping voltametry and atomic absorption spectrophotometry. Infrared techniques are used in dissolved organic carbon determinations.

CHEMICAL LABORATORY CONSIDERATIONS

The marine chemistry group sought the assistance of the computer services group in determining nutrient, heavy metal, and dissolved organic carbon concentrations from raw instrument results, in monitoring the quality of procedures and sampling techniques, and in storing and later retrieving nearly 1600 weekly concentration and physical measurements. Also provision was to be made for future expansion in the number of concentration determinations processed.

All concentration determinations, independent of instrument, used similar techniques. These included the placement of blank and standard samples among water samples within instrument sample trays. Blanks are used to remove the bias produced by inherent in situ turbidity, and/or reagents added to the samples during processing. Standard samples of known concentration are used to calibrate the instruments for each sample tray.

In general, to produce known concentrations from raw instrument results the following were required:

1. A calculation of mean blank value to be subtracted from the water sample data.
2. A linear regression of standard samples to determine the relationship between instrument readings and known concentrations.

To monitor the quality of analytic processes, the following were required:

1. A Students' t-test to determine the equality of means between each sample tray's blank set and all blanks previously processed by the particular instrument.
2. A cumulative sum test between standard set slopes and the set of all standard set slopes determined from previous instrument runs of particular nutrients, heavy metals, or dissolved organic carbon.
3. The determination of the minimum detectable concentration for each sample tray processed.

4. An analysis of covariance between standard set slopes when multiple standard sets were used.

Sample trays would include samples from several stations. At each station multiple depths were sampled, and at each depth replicate samples were taken. To monitor sampling techniques, the mean, standard deviation and coefficient of variation between replicated samples were desired. Also, in two instances, an analysis of covariance between replicated samples at the same depth was required.

The chemistry group would accept or reject instrument results on the basis of a report generated for each sample set processed. Decision making by the software was not required except that water sample concentrations found below minimum detectable limits would be stored as one half the minimum detectable limit calculated for each sample set. Also, missing data was to be ignored in all calculations. Storage and retrieval of concentration and physical data was to be based on four identification levels: cruise number, station number, element (nutrient, water temperature), and replicate number.

COMPUTER SYSTEM CONSIDERATIONS

To allow simultaneous access to the PDP 11/34 by other users, the system to handle the request was written in MU-BASIC. Since all concentration determinations required similar statistical routines, these were written in subroutine form. Due to memory limitations, 3.5K words, programs required overlaying.

A separate data entry routine was written but not incorporated into the root concentration determination programs, (CDPs), for two reasons. One, the availability of the CRT terminal in addition to the in-laboratory hard copy terminal permits two chemists to enter raw data simultaneously. However, the unavailability of a second hard copy terminal does not permit simultaneous running of two CDPs from which reports were desired. Two, a separate data entry program permits the chemists to defer concentration determination until a more advantageous time.

To simplify data entry, information was entered in the order in which analytic instrument results were produced. Virtual files were selected to store the raw information for each process so that the accuracy of entered data would not be affected by disk storage and to conserve disk storage space. These two advantages of MU-BASIC virtual files result from the fact that, unlike sequential files, a conversion from binary to ASCII format does not occur.

Since the numbers of water samples, blanks, and standards and their ordering within sample trays could change, it was decided to create virtual tray layout files for each process which define the statistics to be applied in each case and the sample tray numbers to be used within each statistic. Virtual files are advantageous here due to their random access capabilities.

Under normal operating conditions there would not be sufficient floppy disk storage to contain concentration and physical results for extended periods of time. Nearly 42,000 words would be generated in six months. Therefore, intermediate master update files were used to contain results of the CDPs until a time when they could be used to update a master file contained on a separate floppy disk. Virtual files were used for intermediate files to conserve floppy disk space and to retain the accuracy of results. Updates to the master file were made several times daily during low activity periods.

The floppy disk containing the master file also stored a master file query program, (MFQP), and an MU-BASIC virtual string file containing information describing the contents of the master file. Queries of master file information are made during low activity periods. MU-BASIC virtual files were selected for the master file and the master description file to conserve disk space, to provide random access capabilities, and to remove the need to copy the entire file for each update. Since MU-BASIC virtual files emulate one dimensional arrays, use of a simple algorithm permits access to any element within these files. Also, a virtual master file would retain the accuracy of the final results of the CDPs.

THE CHEMICAL DATA PROCESSING SYSTEM

The final chemical data processing system, (CDPS), consisted of four CDPs, ten statistical subroutines, one raw data entry and edit program (DEP), one MFQP, one master file update program, (MFUP), eight virtual raw data files, four virtual tray layout files, thirteen sequential blank storage files, thirteen sequential standard set slope files, and two sequential work files.

A stream within CDPS is defined by analytic equipment type. Since all streams are similar, only the Technicon AutoAnalyzer® stream will be discussed in detail.

Figure 1 illustrates the AutoAnalyzer® stream. The DEP is used to enter, print, and edit the analytic instrument generated raw information later used by the CDP. Since five AutoAnalyzers® were run simultaneously, five virtual raw data files were used in this particular stream. Other streams where instrument turnaround times were slower required only one raw data file.

All files within a stream contain file names with similar roots. File suffixes define a particular nutrient in the case of the AutoAnalyzer® stream, and file prefixes, common between streams, define the file function. This naming scheme simplified file management. File names are created within programs from a terminal identification of the nutrient being processed. This is illustrated by the following example:

```
10   DATA "NH3", "NOZ+NO2", "NO2", "SIO4", "PO4"
20   FOR J=1 TO 5\READ N1$\NEXT J
30   DATA "NUT1", "NUT2", "NUT3", "NUT4", "NUT5"
40   FOR J=1 TO 5\READ N2$(J)\NEXT J
                  •
100  P1$="S1"\P2$="B"\P3$="S"\E$=".DAT"
                  •
200  PRINT "ENTER THE NUTRIENT NAME";\INPUT N$
210  FOR J=1 TO 5\IF N$=N1$ (J) GO TO 220\NEXT J
                  •
220  DEF FNA (X$, Y$, Z$)=X$ & Y$ & Z$
230  F1$=FNA (P1$, N1$(J), E$)
                  •
280  OPEN F1$ FOR OUTPUT AS FILE VF2
```

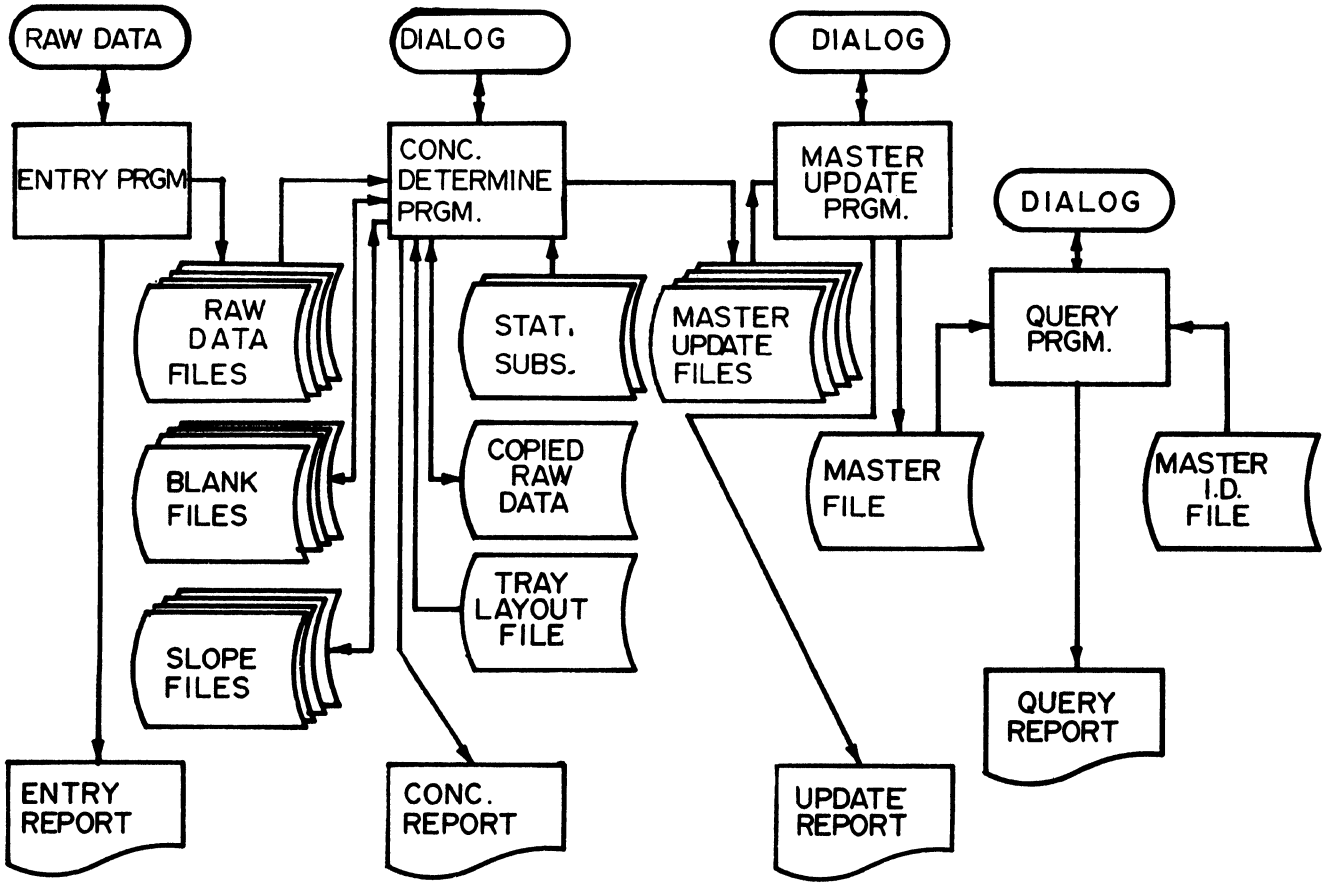


FIGURE 1. TECHNICON AUTOANALYZER® STREAM

Only one raw data file is processed by the CDP at one time. The nutrient to be processed is identified by the chemist and the appropriate raw data file is copied into the temporary raw data file. A separate virtual work file is used in each stream for temporary storage of raw data since these data could then be modified within the temporary file by dilution factors and blank values without destroying the original raw data.

Subroutines were written with common line numbers so they could be overlaid. The CDP uses the statistical subroutines in conjunction with pointers into the tray layout file. The following illustrates the interaction between a CDP pointer, Z, the tray layout file, VF1, and the raw data file, VF2, within a statistical subroutine used to calculate a mean.

```
5000 X=Z + 1
5010 N=VF1 (X)
5015 M=Ø
5020 FOR J=1 TO N
5030 P=VF1(X + J)
5040 M=VF2(P) + M
5050 NEXT J
5060 M=M/N
5070 RETURN
```

In the above example, the value of the variable N is read from the tray layout file. This variable is used to specify the number of raw data used in the mean calculation. The variable P, also read from the tray layout file, defines the virtual file positions of the raw data.

Sequential blank files and cumulative sum slope files contain blank means and standard set slopes used to compare each sample set run with previous runs.

Keyboard dialog with the CDP consists of identification of the nutrient to be analyzed, the entry of dilution factors by sampler number, and the entry of probability levels.

The final report consists of results of the various statistics and a listing of the final nutrient concentrations by sampler number.

Master update files contain identification information used in the MFUP in addition to concentration results. Identification information consists of a process code and week number, identifying the entire file; and a station number, sampler number, and replicate number identifying each concentration result. In the case of the AutoAnalyzer® stream, concentration identification information, with the exception of week number, does not change between runs. Therefore, they are stored permanently in the master update files. However, other instrument streams required entry of identification data.

The MFUP uses the identification information supplied with the concentration results in conjunction with a simple algorithm to store the concentration data in the appropriate master file location. Figure 2 illustrates master file storage by algarithm.

The master file is updated and queried during certain non-active periods of the day. The possibility of destroying previous master update file results with subsequent runs is present. To alleviate this possibility, the master update files are tagged with a default value in the location reserved for a week number. Upon data entry, a check is made of the week number within the appropriate master update file. If a default is detected, data entry is permitted. If a default is not detected, then data entry is not permitted.

The MFQP functions in a similar manner to the MFUP. Variables are selected for printing on the basis of a week number range, sampler number, and variable number. Again a simple algorithm much like that illustrated in Figure 2 is used to locate the master file positions of the desired information.

ADVANTAGES OF VIRTUAL FILES WITHIN CDPS

MU-BASIC virtual files were used extensively in the resulting CDPS to determine nutrient, heavy metal, and dissolved organic carbon concentrations from raw chemical analytic instrument results; to monitor the quality of procedures and sampling techniques, and in storing and later retrieving nearly 1600 weekly concentration and physical measurements made in the Indian River Lagoon, Florida.

Virtual files were found to have the following advantages on a small PDP 11/34 minicomputer:

1. A more accurate retention of data as compared with sequential file processing.
2. A four fold savings in disk storage space as compared with sequential file processing.
3. A memory saving in all instances where arrays exceeded one block in length.
4. Virtual files provide random access capabilities where sequential files do not.
5. Files need not be read in their entirety to reach the last file element as with sequential file processing.
6. When updating files, the entire file need not be copied as in sequential file processing.

CDPS ADVANTAGES TO THE CHEMIST

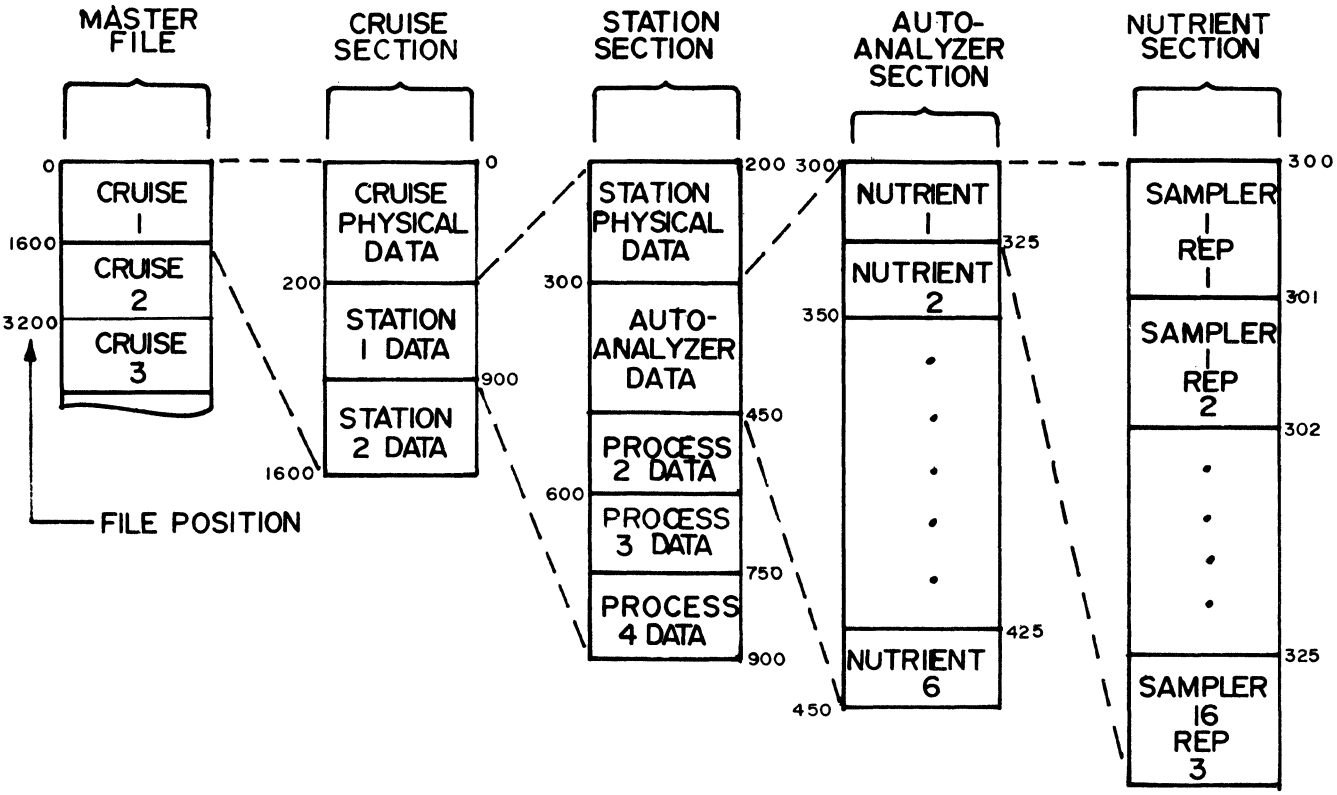
Prior to the creation of the CDPS, calculations involved in statistical analyses occupied nearly one half man day for each sample set. With the advent of the CDPS, similar analyses were processed in 15 minutes or less.

Before the CDPS all calculations were performed by hand entry and manipulation of data using desk top calculators. Intermediate and final results were copied into individual process notebooks for future reference. The adoption of the CDPS by the chemists has removed many sources of error inherent with previous methods, and has greatly simplified the correlation, interaction, and retrieval of information for the chemistry group as a whole.

REFERENCES

[1] Char, K.M. and J.P. Riley. 1966. The automatic determination of phosphate in seawater. Deep-Sea Res., 13: 467-471.

[2] Armstrong, F.A.J., C.R. Stearns and J.D.H. Strickland. 1967. The measurement of upwelling and subsequent biological processes by means of the Technicon AutoAnalyzerR and associated equipment. Deep-Sea Res., 14: 381-389.



NUTRIENT MASTER FILE POSITION = A+B+C+D+E

WHERE: A= (CRUISE NUMBER−1) X 1600 + 200

B= (STATION NUMBER−1) X 700 + 100

C= (PROCESS NUMBER−1) X 150

D= (NUTRIENT NUMBER−1) X 25

E= REPLICATE NUMBER

FIGURE 2. MASTER FILE LAYOUT AND STORAGE ALGORITHM

[3] Monstev, A.W. and H. Clan. 1977. Electromyographic diagnosis using a PDP-12. Proc. Digital Equipment Users Society, 3(4): 1015-1018.

[4] Digital Equipment Corporation, MU-BASIC/RT-11 User's Manual, Maynard, Mass., 1975.

ACKNOWLEDGEMENTS

The author wishes to express his appreciation to Mr. John Montgomery, supervisor of nutrient and trace metal chemistry; and his staff; and Mr. Edward Gallaher, supervisor of computer services, Harbor Branch Foundation, Inc. for their support during the development of CDPS and in the preparation of this paper.

This is contribution number 117 from the Harbor Branch Foundation, Inc.

GT-43 AIRPLANE FLIGHT SIMULATION

C. Frank Kyle and Phil Sherrod
Vanderbilt University
Nashville, Tennessee

ABSTRACT

This paper describes a GT-43 based program, called "PILOT" which simulates the flight of a light airplane. The simulation is accomplished by calculation in real-time of the major forces which act upon the plane. These include the forces acting on the wings, the tail and control surfaces, the engine thrust, drag, and landing gear forces. Joystick and potentiometer controls are used to fly the simulated airplane. Normal flight, take-offs, landings, crashes, and stalls are handled realistically.

"PILOT" is a program which runs on a GT-43 to simulate the flight of a light airplane. The pilot of the simulated flight sits before the GT-43 display screen and operates joystick, potentiometer, and push-button controls to control the flight. The real-time behavior of the airplane and its flight instruments are always displayed, but the pilot can select any of three different points of view from which to watch the behavior of the airplane. The first is the view of the external terrain which would be seen by a pilot onboard the aircraft. A second view shows the aircraft as seen by an observer in the control tower of the airport. The tower observer always tracks the flight of the airplane. He is equipped with zoom-lens binoculars for use when the airplane is not in the immediate vicinity of the airport. The third view is that of a distant observer who also has a zoom lens. He is free to look about as he wishes and is even free not to watch the flight so that blind instrument approaches can be practiced.

There are seven flight instruments which are continuously displayed. These range from simple to relatively complex. There is a fuel gauge, an airspeed indicator, and an altimeter which have obvious functions. There is an artificial horizon which displays the aircraft attitude in flight, and an automatic direction finder (ADF) which indicates the direction to the airport relative to the current aircraft heading. Two instrument landing system (ILS) devices are also included. A VOR/DME display indicates the bearing and distance to the aircraft from the airport. One airport runway is equipped with ILS transmitters. The last flight instrument is a glide slope indicator which functions on approaches to the ILS runway to indicate the position of the airplane relative to the nominal glide slope.

PILOT runs on a GT-43 equipped with 28K words of memory, floating point hardware, and an AR-11 A/D subsystem to which the controls are connected and which provides the real-time clock. PILOT runs under RT-11 and in its current form it nearly saturates the resources of the machine. It heavily loads both the CPU and the VR-11 display processor and we run it under single job RT-11 to save core.

The program consists of two basic modules, a display library written in MACRO-11 and the flight dynamics module written in FORTRAN. The display library, GTLIB, was written at Vanderbilt as a set of general purpose routines which provide efficient access by a FORTRAN program to the VR-11 and AR-11 hardware through a set of low level routines, and relatively sophisticated routines for driving the display. The display library is specifically designed to allow the calculation and generation of one display command list to be overlapped with the display of another. Routines are included to provide facilities for the efficient display of moving 3-dimensional objects. There are routines which translate or rotate either an object or the observer in the 3-dimensional space, for example. One routine will perform a perspective projection of the resulting scene, clip the result to the screen window size, and call low level routines to display the result. PILOT depends heavily on the availablity of these facilities. The removal of hidden lines is a problem of much greater difficulty. Because of limited computer resources the removal of hidden lines was not attempted.

PILOT originated as a demonstration program to illustrate the proper use of the GTLIB routines. For various reasons (users requested more functionality, it was a challenge, it kept us away from other work) it grew to have a life of its own. Basically the program consists of a routine which reads data describing the airplane and the surrounding terrain and then uses the GTLIB routines to generate the display image in real time. This routine does "Computer Science". This main routine calls a subroutine, MOVE, which calculates the motions of the airplane which result from the various forces acting on it. MOVE does "Physics".

MOVE, in turn, calls various other routines which calculate the forces acting on the airplane from its linear and angular velocities, and from its position when taxiing. WINGS is called, for example, with arguments specifying the airspeeds of a representative point on each wing. This information together with aircraft configuration data is used to calculate the force and torque which each wing contributes to the motion of the

aircraft. WINGS also calculates the effect of the ailerons and, if the angle of attack is too high, a flag is set which causes the main program to display a stall warning. In more severe cases an actual stall is simulated by reducing lift and increasing drag.

Such force routines are included to approximate the effects of the wings, the tail surfaces, the engine thrust and fuselage drag, the landing gear, and a special routine is included to detect crashes. These force routines do "Engineering".

Our experience has been that the simulation of flight can be very realistically accomplished with relatively crude approximations to the aerodynamic forces. The ground forces involved in taxiing are much more difficult because of the nature of the frictional forces between the tire and the surface. Momentarily large forces can result which are difficult to handle by ordinary numerical integration. A better solution seems to be to treat the non-skidding wheel as a constraint and then calculate the constraining force to determine whether a skid in fact occurs. In any case, the WHEELS routine simulates the forces resulting from a three point landing gear with rolling wheels equipped with brakes, and supported by springs and shock absorbers. The nose wheel is steerable. WHEELS is the largest of the force routines.

PILOT reads the data which specify the airport layout and the airplane configuration from data files. Consequently the simulation can be varied greatly without intimate knowledge of the program. We have experimented with a terrain layout containing two airports and with various airplane configurations. These changes are largely cosmetic and simple to effect. Parameters used in the flight dynamics calculations can be changed in the same manner except that more care must be exercised to produce an airworthy craft. Incidentally certain parameter errors, e.g. a weight with the wrong sign, produce surprising results when the simulation is started.

There are various directions in which PILOT could conceivably evolve from here. It is currently highly core limited but that could be alleviated by overlaying the fairly extensive initialization code. There are fairly simple ways by which the curvature of the earth could be simulated to permit the inclusion of multiple airports and navigation aids without the need to process them all for display all the time, since most would be hidden by the horizon, and to avoid the numerical problems inherent in handling data which represents the detail of an airport (scale of 1-10 feet) separated by large distances from another airport (scale 10-10,000 miles). In addition a better simulation of the flight is certainly possible by making better approximations to the actual performance of a specific real aircraft. Unfortunately, the current state of PILOT borders on exhausting the resources of the GT-43. It is currently fairly well optimized for run time and consequently no great improvement in that area is likely. Measurements indicate that the display and simulation times are comparable so further progress will probably await the availability of more powerful hardware. In the meantime the current PILOT is an enjoyable game, if not valuable instruction in display processing (and perhaps in pilot training?).

AN INEXPENSIVE SYSTEM FOR DIGITIZING PICTORIAL INFORMATION

Charles Kapps
Temple University
Philadelphia, PA.

and

Lawrence Mays
University of Alabama
Birmingham, AL.

ABSTRACT

One of the chief stumbling blocks in the development of picture processing systems has been the very high cost of the equipment necessary for digitizing pictorial information. Usually, this equipment consists of specially designed television cameras or scanning equipment which must be slow enough to allow for processing time in the computer. At the same time, digitizing should be fast enough to be practical.

This article describes a system which allows standard, "off-the-shelf" television equipment to be interfaced to a computer. The function of the equipment is to slow down the rate of information flow from the television equipment. Alternative designs are shown which allow for varying data rates as required by the particular system.

The cost of building the interface is quite low, and since standard, mass produced television equipment is used, overall cost of the system is very reasonable.

BACKGROUND

The problem of digitizing pictorial information can be broken into two parts. The first part consists of converting the picture into some form of electronic signal. Second, this electronic signal must then be put into some digital form which is useable to a computer.

Some of the simpler systems involve human interaction such as digitizing tablets, light pens, joysticks, and tracking balls. Because of the need for human interaction these systems are extremely slow. Most of these systems can, of course, be automated by use of light sensing devices coupled to some sort of electro-mechanical drive system which would in effect take the place of human hands and eyes in the system. Such electro-mechanical systems are not only quite expensive, but also slow because of the inherent slowness of mechanical operations.

The obvious solution to these problems is to remove the mechanical parts of the system and have a purely opto-electronic scanning system. Such systems of course are the basis for all modern television. Television-type systems are, of course, analog in nature, and the difficulties arise in converting this sort of information to digital information. The problem is, just the reverse of that encountered with mechanical systems. Television systems are too fast. It is possible to digitize television signals in real time, and this is, in fact, being done in some digital systems currently being used by the broadcast industry. Because of the extremely high data rates (at least 16 megabits/second), the equipment is extremely expensive. In addition, these data rates are much too fast to be processed by most general purpose computers, especially the less expensive mini and micro computers which are now being used extensively in laboratories.

There are two solutions to the problem of excess speed. The first is to use video equipment which is specially designed to operate at slower speeds. This has in fact been done, however, the equipment is usually extremely expensive primarily because of the non-standard design. An alternative solution, which this article explores, uses standard video equipment, and employs special techniques in the analog to digital conversion process which slow down the rate of information flow from the video equipment to the computer. The video equipment used can be either a standard, mass produced camera of receiver. As can be seen from the following, the digital conversion equipment is minimal. Consequently an entire system, capable of connection directly to a medium speed digital input/output port on a computer can be made to sell for under one thousand dollars.

Finally, some alterations are made to the basic, cheap system. For a moderate increase in cost, these allow the video system to be interfaced to the computer on a high speed, direct memory access port, or directly to the computer memory bus. The speed of the system can thus be improved. In effect, the speed becomes limited only by the capabilities of the computer.

THEORY OF OPERATION

As mentioned above, a standard television system generates 16 to 32 million bits of information per second. Our system reduces this information rate to a more manageable rate. To do this, one of two things must be sacrificed, resolution or frame repetition rate, (or perhaps a combination of both). Our decision was not to sacrifice resolution. As a consequence, it takes much longer than 1/30 of a second to scan an entire picture. Therefore, we cannot deal with moving pictures, at least not in real time. (For some applications, this may require the use of a stop action device such as is found in many video recording machines). Consequently, the remainder of this discussion will assume that the television camera is viewing a still picture, and scanning it over and over again at the normal rate of thirty interlaced frames per second. (The systems used by the authors are for standard American broadcast equipment. Clearly, the interface equipment could be modified in order to accommodate the various European and other television systems).

In order to accomplish a sufficiently slow sampling rate, only one sample will be made on each horizontal line. Thus, the resulting sampling rate will be 15,750 samples per second. This is well within the capabilities of less expensive analog to digital converters, computer interfaces, and mass storage devices. Using this strategy, a sample is made at some fixed delay time after the occurance of a horizontal sync pulse. The result is that a vertical line is scanned on the picture, as is shown in Figure 1.

It should be noted that the vertical sampling line is passed over twice, because of the interlace. On the first pass the odd lines are sampled, and on the second pass the even lines are sampled.

In order to sample an entire picture, it is necessary to sample successive vertical lines in a fashion similar to the way regular television scans a picture in terms of successive horizontal lines. Sampling of successive vertical lines is done by increasing the time delay between the horizontal sync pulse and the sample point. As the time delay increases, the vertical scan line moves to the right across the picture.

Since 1/30 of a second is required to sample each vertical line, a minimum of about 30 seconds would be required for sampling an entire picture with full available resolution. This, of course, does not allow for processing time, or latency time. (Latency is due to the fact that the television equipment is continuously scanning at 30 frames per second. If the processor misses the beginning of a frame, it is necessary to wait until the next frame begins. A modification to the system is proposed later to avoid much of the latency problem). The first application of this system made little use of software techniques for overlapping input/output time, and therefore required four minutes to copy a full resolution sample to magnetic tape for future processing.

PROTOTYPE SYSTEM

Figure 2 shows the schematic diagram of the prototype system. This system was designed to connect a standard television camera to a PDP-11 computer. The interface to the computer operated through an LPS-11 laboratory peripheral system with an analog to digital input and a digital input/output port.

The prototype system receives three inputs from the television camera; horizontal drive, vertical drive, and a standard video signal. (Of course, with the appropriate additional circuits, the vertical and horizontal drive signals could be derived from a composite video signal).

The system produces two outputs to the computer. The first is the sample value which conveys the brightness of the sampled point. This signal is in analog form which is then converted to digital by the LPS-11. An alternative structure would be to have included an A to D converter within the prototype system itself. The second output signal is an interrupt signal which informs the computer when the sample value is ready.

The system receives three digital input signals from the computer. The first is a multi-bit signal designated A_1 - A_9 which indicates the delay time from the horizontal sync pulse to the sample point. Treated as a binary number, this signal specifies the delay time in multiples of 125 ns. The other two signals from the computer are the Reset and Go signals. Reset simply initializes all flip-flops, and thus turns off the sample process. Go signals the start of the scanning of a vertical line.

Still referring to Figure 2, IC_{1A} and IC_{2A} form a circuit which produces a pulse at the beginning of a picture. In the interlaced system, the horizontal sync pulse for the first odd line immediately follows the trailing edge of the vertical sync pulse. On the first even line the trailing edge of the vertical sync pulse is approximately halfway between horizontal sync pulses. See Figures 3a and 3b (Note, that the horizontal and vertical drive pulses at this point have been inverted by Q_1 and Q_2). IC_{1A} is a one shot, the purpose of which is to lengthen the vertical sync pulse slightly so that it coincides with the first odd horizontal pulse.

IC_{1B}, IC_{3B}, IC_{3A}, and IC_{3B} are the state flip-flops and logic, and indicate when the system is in the "Go" condition. IC_{2C} and IC_{4B} Produce the reset condition.

IC_{4A} and IC's 5-8 form the delay circuit. The "Go" condition opens gate IC_{4A} which causes the horizontal sync pulse to load data bits A_1 - A_9 into the presettable down counter IC's 5-7. IC_8 is an 8 M Hz clock which causes the counter to count down once each 125 ns. When the count reaches zero, a borrow signal is generated which stops the clock pulses, and signals the end of the delay to the sampling circuitry.

The sampling circuitry consists of a sample and hold circuit (IC_{10} and IC_{11}) and two one shots (IC_{9A} and B). IC_{9A} actuates the sample and held circuit which samples the video signal. IC_{9B} interrupts the computer after a suitable delay to handle settling of the sample and hold output.

SPEED IMPROVEMENT OF PROTOTYPE SYSTEM

The primary speed limitation of the prototype system is the fact that only one sample is made for each horizontal line, or 15,750 samples per second. It

may well be possible that the computer is capable of sampling at a faster rate.

Two modifications to the system are shown here, which allow more than one sample per line. This would allow us to double, triple, quadruple, etc. the overall speed of the system. Limitations in speed would probably be the processing rate of the computer.

The first scheme for improving speed is simply a modification of the master counter of the prototype system (IC_{5-7} in Figure 2). This counter enables the analog sampling of the video signal on the count of zero. Recall that this is a nine bit counter, therefore there are 512 possible sampling points. Observing the modifications shown in Figure 4, we are in effect adding more bits to the more significant end of the counter. We would probably make a corresponding decrease of bits at the least significant end so that our total line count remains 512, (or thereabouts).

If IC_{21} is a two bit counter, and we remove two bits from the master counter, IC_{5-7}, we will get a signal to sample when we reach a count of N which comes from the seven bit input from the computer $A_1 - A_7$. However, the clock, IC_8 is not inhibited as before, but continues until IC_{21} reaches its maximum count. Thus, we get a sample when the number of pulses reaches N+128, N+256, and N+384. We are now scanning four vertical lines one each frame. Each line is one fourth of the way across the picture. (See figure 5.) The result would be that a picture could possibly be acquired in one fourth the time. Of course, we are assuming that the sample and hold, A-D conversion, and computer processing of a sample could occur 4 x 15750 = 63000 times per second. This allows slightly less than 16 mμ per sample, probably the maximum processing speed for most minicomputers such as the PDP-11/05.

We should note that there is no reason other than that of convenience, for the counters used in this system to be binary counters. Thus, the number of vertical scan lines need not be a power of two. We could therefore easily modify the system to scan 500, 750, 900 etc. vertical lines rather than 512.

An alternative method of scanning is shown in the modifications of Figure 6. This system makes a succession of samples on the consecutive clock pulses following the point where the master counter reaches zero. The main difference between this system is that instead of having scattered sample lines across the picture, there is a scan of a cluster of adjacent lines forming a band in the middle of the picture.

The disadvantage of this system is that more hardware is needed. Since the samples arrive in a cluster, too fast to be analyzed separately, individual sampling circuits are required for each point.

The advantage of the system is that the samples are not scattered, but lie within a region. This can be very useful if we are not sampling the entire picture, but merely an area within the picture.

The output is shown going to a bank of A to D converters, this could of course be replaced by one or a few A to D converters, suitably multiplexed. In fact, with multiplexing controlled by a pulse train, the data could be delivered to the computer at the same rate of speed as the Figure 4 system.

FURTHER SPEED IMPROVEMENT

The basic limitation of speed in the systems in Figure 4 and 6 are that the computers can only process the data just so fast. There is, however, one method left to produce a significant improvement. This is direct memory access. Direct memory access techniques allow the data from the A-D converters to be stored directly into the computer memory without requiring the computer itself to perform any processing. This can be achieved using the systems shown in Figures 2, 4, or 6 and simply connecting the outputs from the A-D converters to a direct memory access port.

It is possible for such a system to deliver information to a large computer at a rate of four million samples per second. However, the cost of such a system clearly could not be described as inexpensive.

Nonetheless, there are some reasonably priced systems for direct memory access to minicomputers. An alternative to direct memory access which may even have a speed advantage is a system which makes use of the fact that many computers are designed with a general purpose bus structure, such as the PDP-11's UNIBUS R.

In this system the video interface contains its own memory of 512 bytes. These hold the converted sam samples from the useful part of a vertical line. There is also an interface to the computer bus which allows the 512 byte memory to "look like" memory on the computer. There is an address switch which determines whether the 512 byte memory gets its address from the horizontal line counter as it does during writing, or from the computer bus as it does while reading. Figure 8 shows the general layout of such a system.

No control is shown on the address switch, as there are several possibilities. One method would be to have the switch operated at high speed, back and forth, depending upon whether the computer or TV interface is requesting use of the memory. This method would involve a fairly complex set of arbitration logic to avoid conflicts between the two contenders for the memory. A simpler method of control would be to forbid computer access to the memory while a vertical line is being scanned. Here the control of the switch is simply derived from the GO signal in the interface. This method is not really much slower than the first method, since it would be very difficult to take advantage of overlap possibilities, because the computer does not "know" exactly where the TV interface is in the scanning process. Circuitry could be added to allow this, but would not have any advantage over the dual memory system which follows.

The main advantage of the Direct Memory System is that it frees the computer of having to perform interrupt processing on every sample point. This could account for a considerable amount of processing time in a minicomputer configuration.

Another problem which this system helps to save is latency. Our original, prototype system could require as much as 1/15 second to complete a scan if we were unfortunate enough to initiate the scan just after the odd vertical sync pulse. Since we have a

counter in the system which keeps track of the horizontal line number, (see Figure 8) it would not be too difficult to keep that counter running continuously. This allows data to be picked up from whereever you are in a scan. What we would then do is start loading the memory in the middle, and eventually wrap around to loading the lower part. All that is needed is a second counter or timer which signals the computer when this wrap around is complete. This will always be 1/30 second after scanning starts.

An improvement over the Direct Memory System is a Dual Direct Memory System shown in Figure 9. This system is essentially like the Direct Memory System, except that there is a second memory. While one memory is switched to the computer bus, the other memory is switched to the TV interface. This allows the overlap referred to above, since the computer can process the contents of one memory at the same time as the TV interface is loading the other memory.

It naturally follows that the techniques of speed enhancement employed in Figures 4 and 6 can be used with either the single or dual direct memory systems. The cost for doing so would be the increase of memory required by the systems.

CONCLUSION

At the time of writing, the prototype system shown in Figure 2 has been constructed and in operation for the better part of a year. This system interfaces a standard TV camera to a PDP-11/05 computer. Not counting the cost of the TV camera or the LPS-11 interface to the PDP-11, the cost of the system was extremely minimal. (Parts costs were less than $100).

The system works quite well, and has been used for a number of experimental picture processing endeavors. One experiment involves the use of a line printer to produce a low resolution rendering of a picture. This kind of process is now becoming a popular novelty item. Our system requires a $7\frac{1}{2}$ second exposure, which is a little slow. Here, however, much use could be made of the speed improvement techniques discussed above. We can do so without the fear of getting too fast, because the low resolution required means that it is not necessary to sample every line.

Figures 10 and 11 show samples of pictorial output which was processed by our system. Figure 10 is a low resolution picture processed as described in the above paragraph. Figure 11 is a high resolution picture which employs nearly the full amount of resolution available. Output was generated on a Versatec D 900 electrostatic printer/plotter. Half tone gray scales are produced by printing a 4 x 5 array containing a varying number of black dots corresponding to the darkness of the area. The dots are randomly placed in the 4 x 5 array in order to produce a smooth texture.

Exposure times for these ten pictures was 7 1/2 seconds for Figure 10 and 2 1/2 minutes for Figure 11. The major part of the time involved was not due to the TV interface, but rather to the processing time needed for saving the pictorial data. More efficient programming could reduce exposure by 50 to 75 percent.

In general, the authors can see much possibility for development of techniques of this sort.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support given by the following persons: John Fowler for hardware support, Steven Lipschutz for photograph assistance, Marcia Kapps for drafting, and Jackie Harriz for acting as subject matter for Figures 10 and 11.

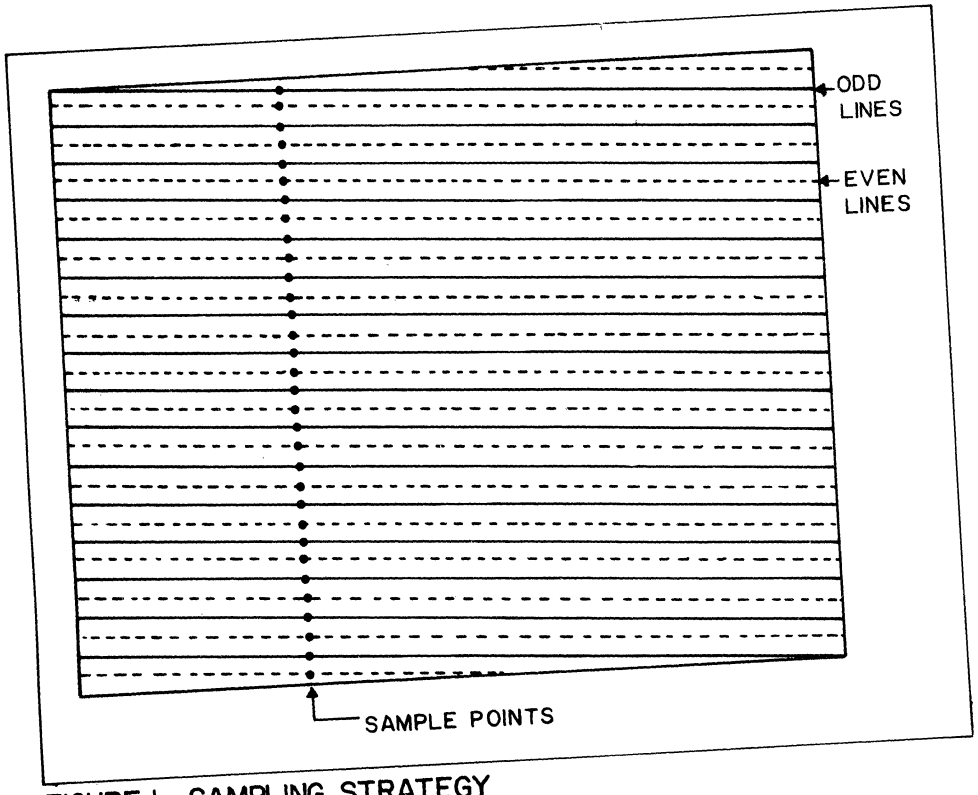


FIGURE I. SAMPLING STRATEGY

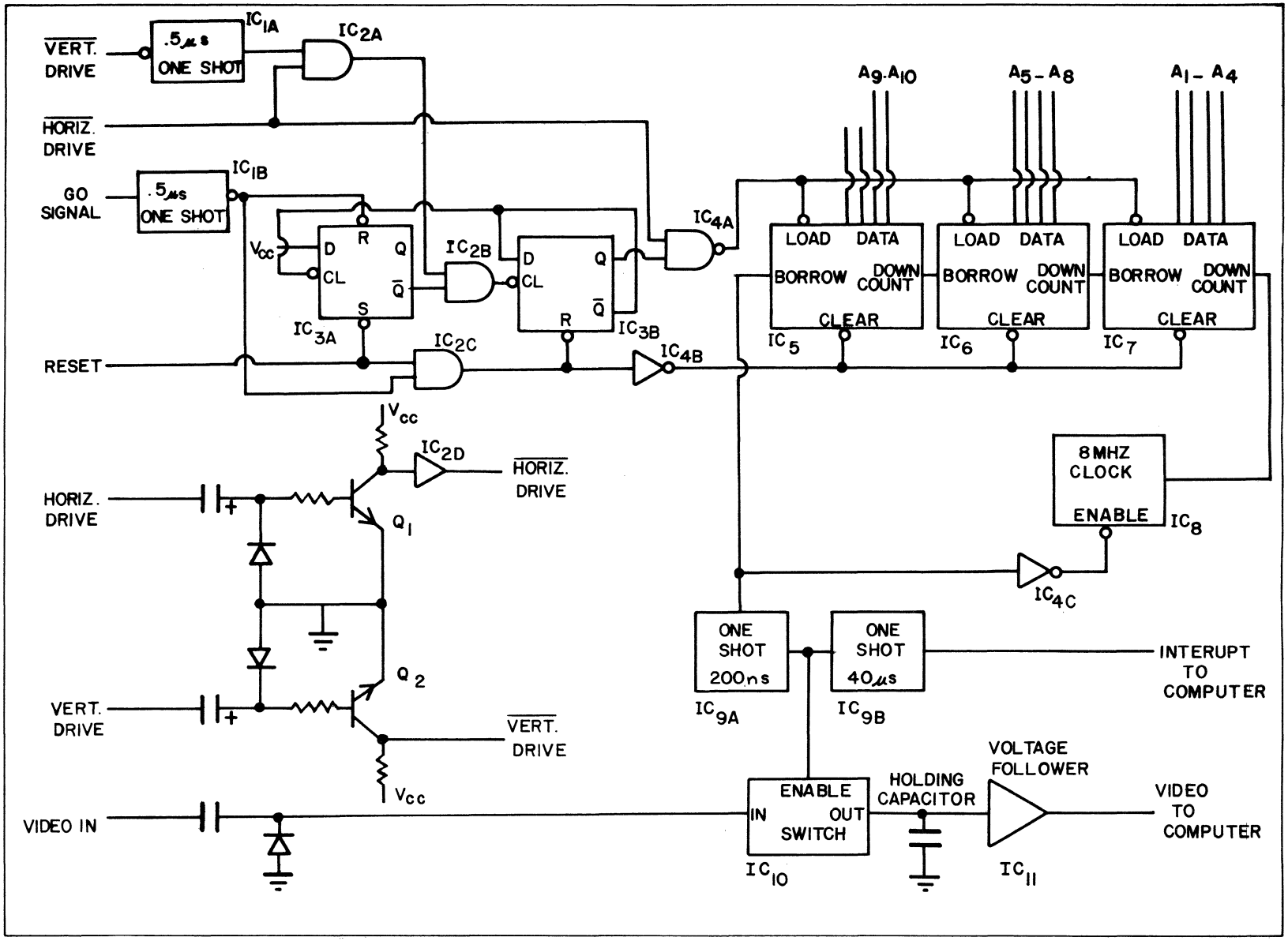


FIGURE 2. PROTOTYPE SYSTEM

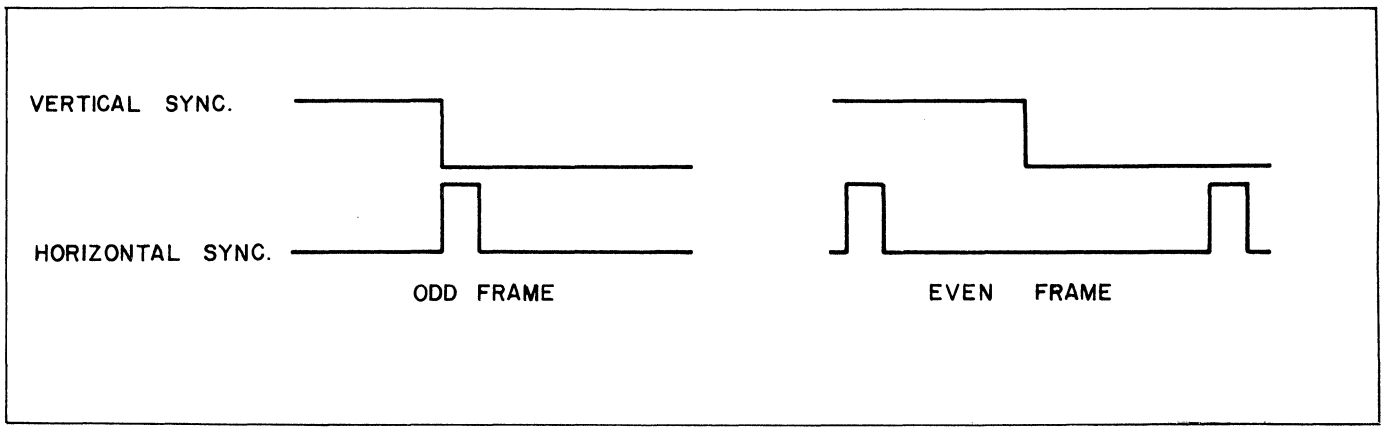


FIGURE 3. BEGINNING OF ODD AND EVEN FRAME

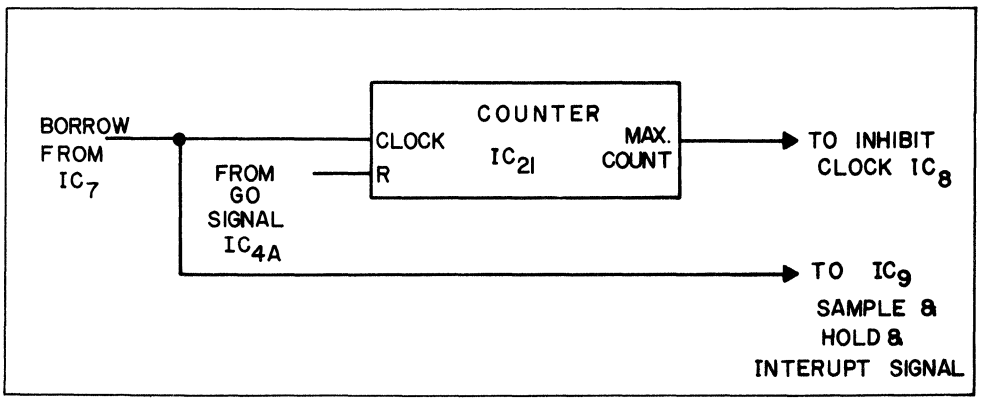


FIGURE 4. SYSTEM MODIFICATION FOR SAMPLING SEVERAL VERTICAL LINES ON EACH FRAME

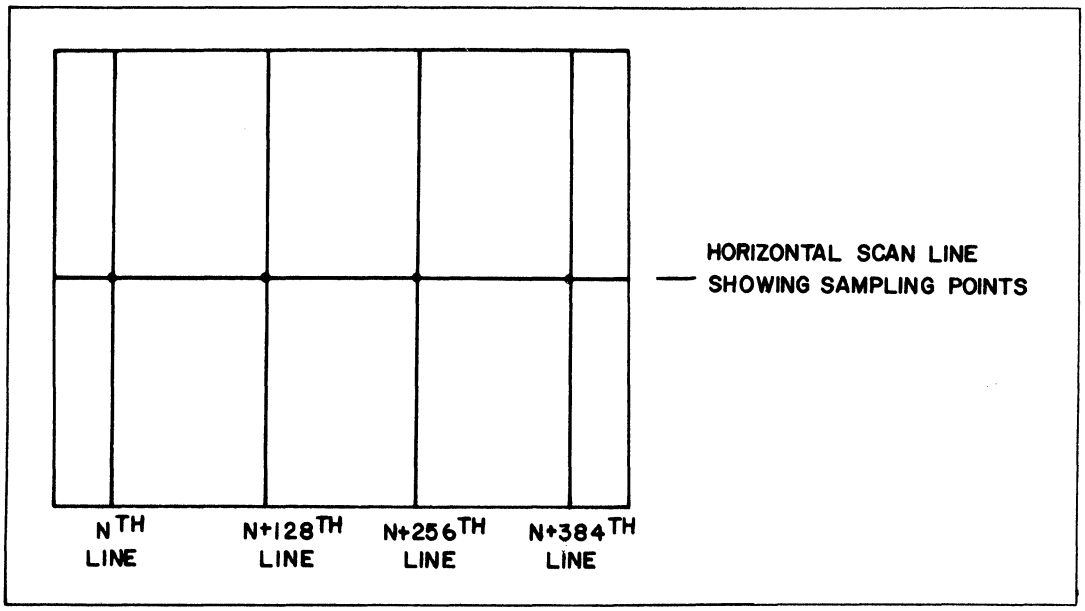


FIGURE 5. SCAN OF MULTIPLE VERTICAL LINES

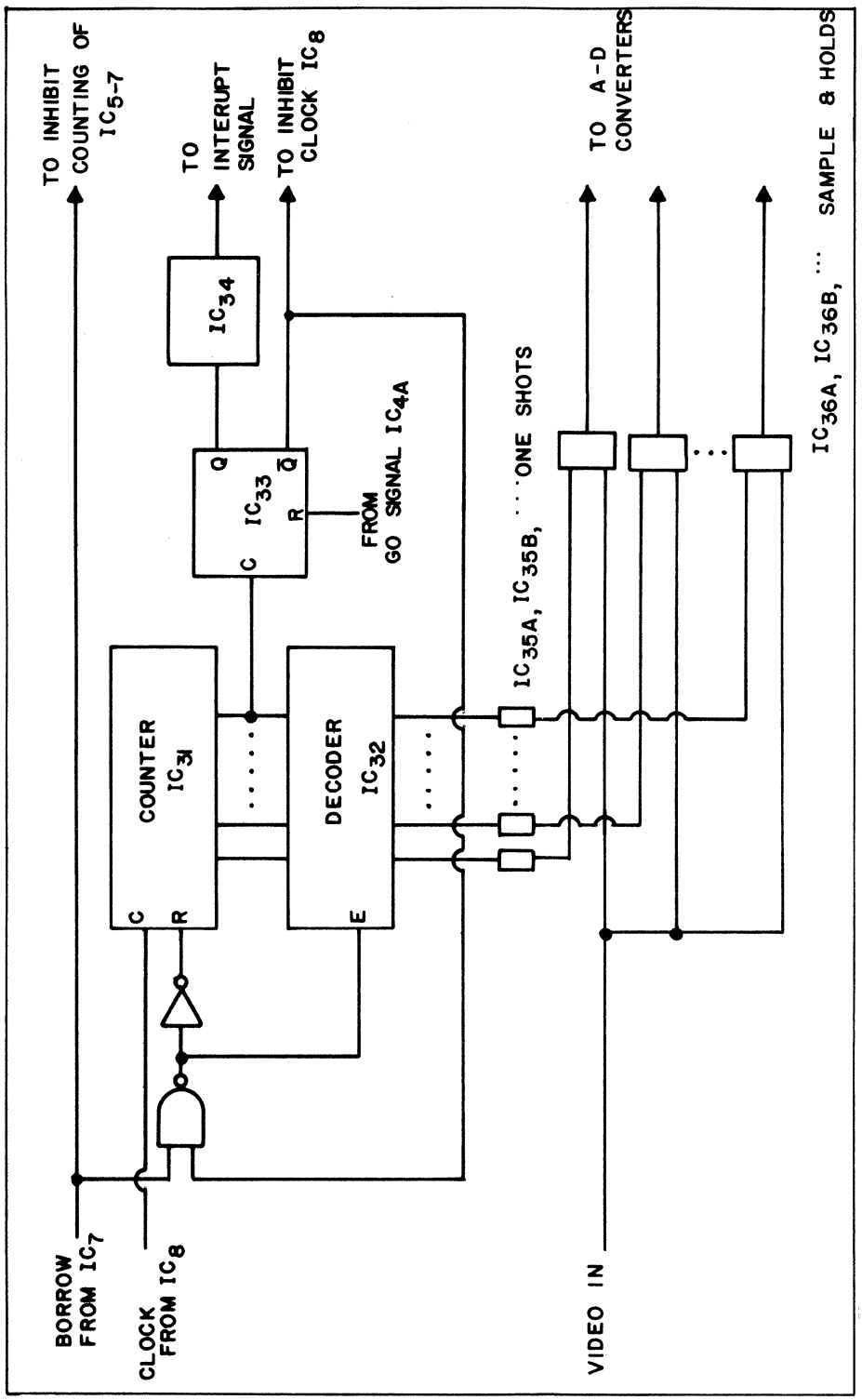


FIGURE 6. SYSTEM MODIFICATION FOR SAMPLING A BAND OF VERTICAL LINES

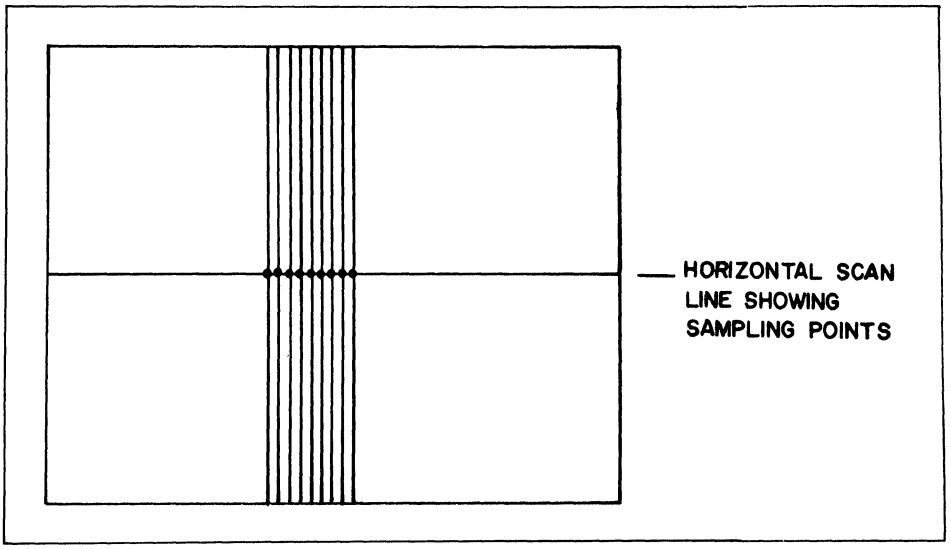


FIGURE 7. SCAN OF A BAND OF VERTICAL LINES

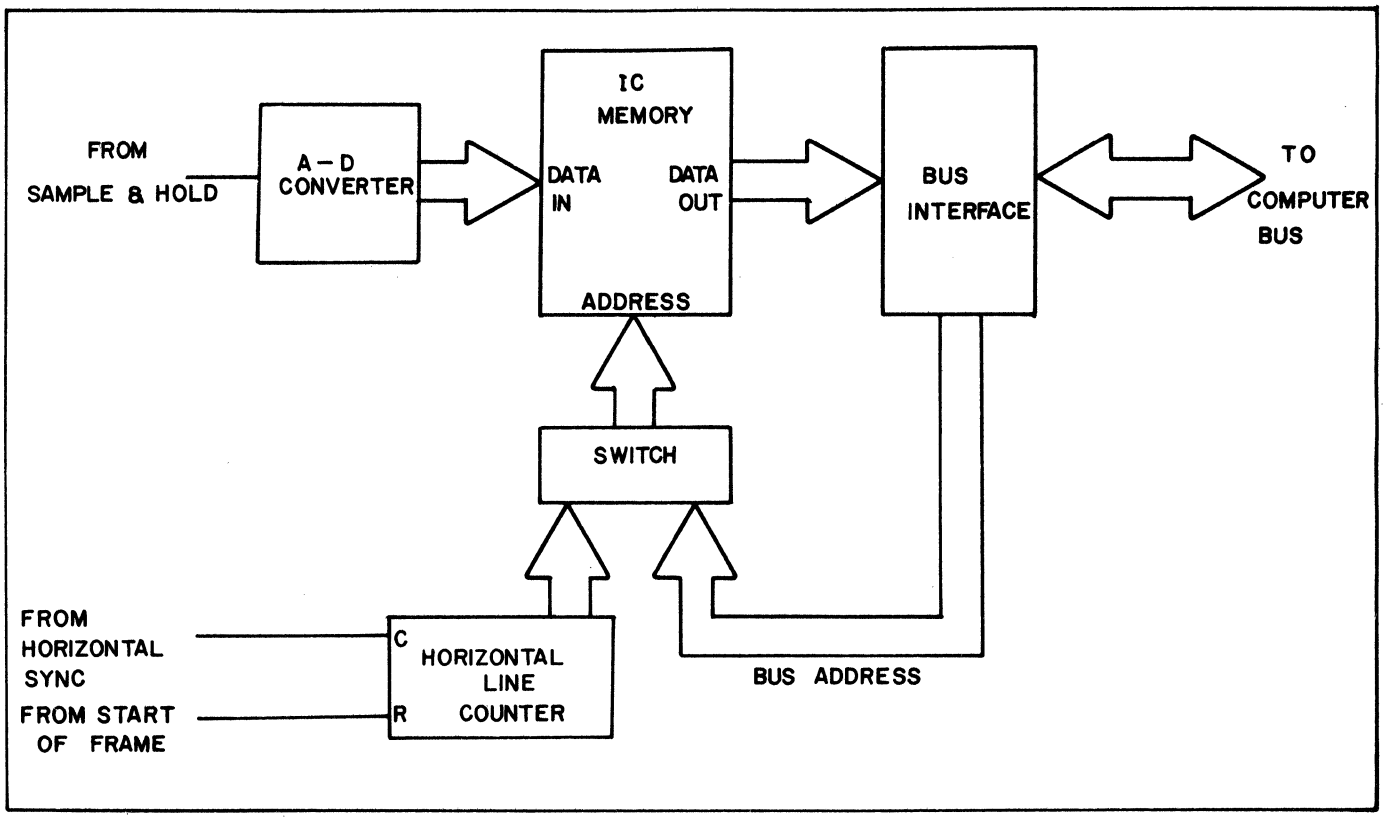


FIGURE 8. DIRECT MEMORY SYSTEM

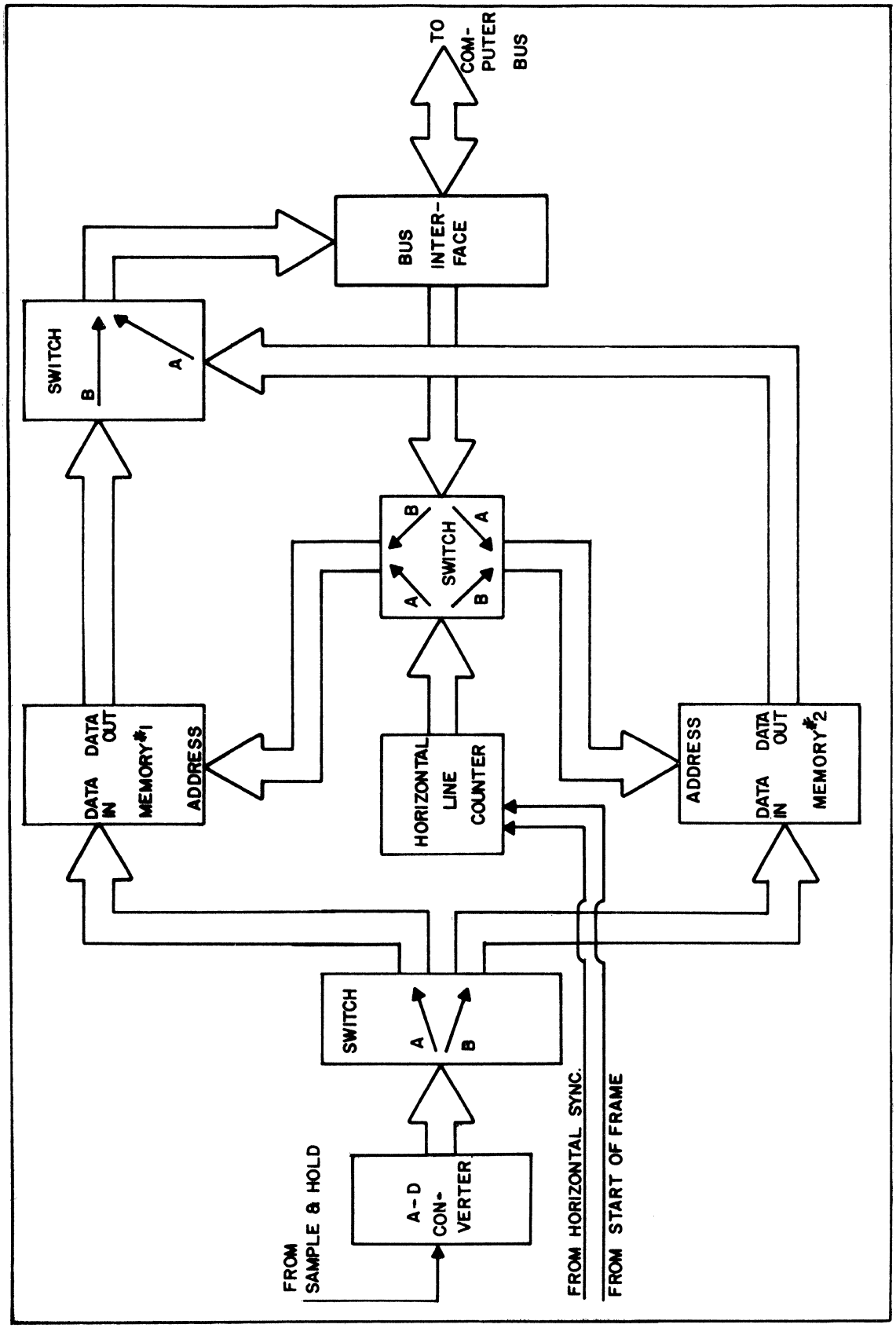


FIGURE 9. DUAL DIRECT MEMORY SYSTEM

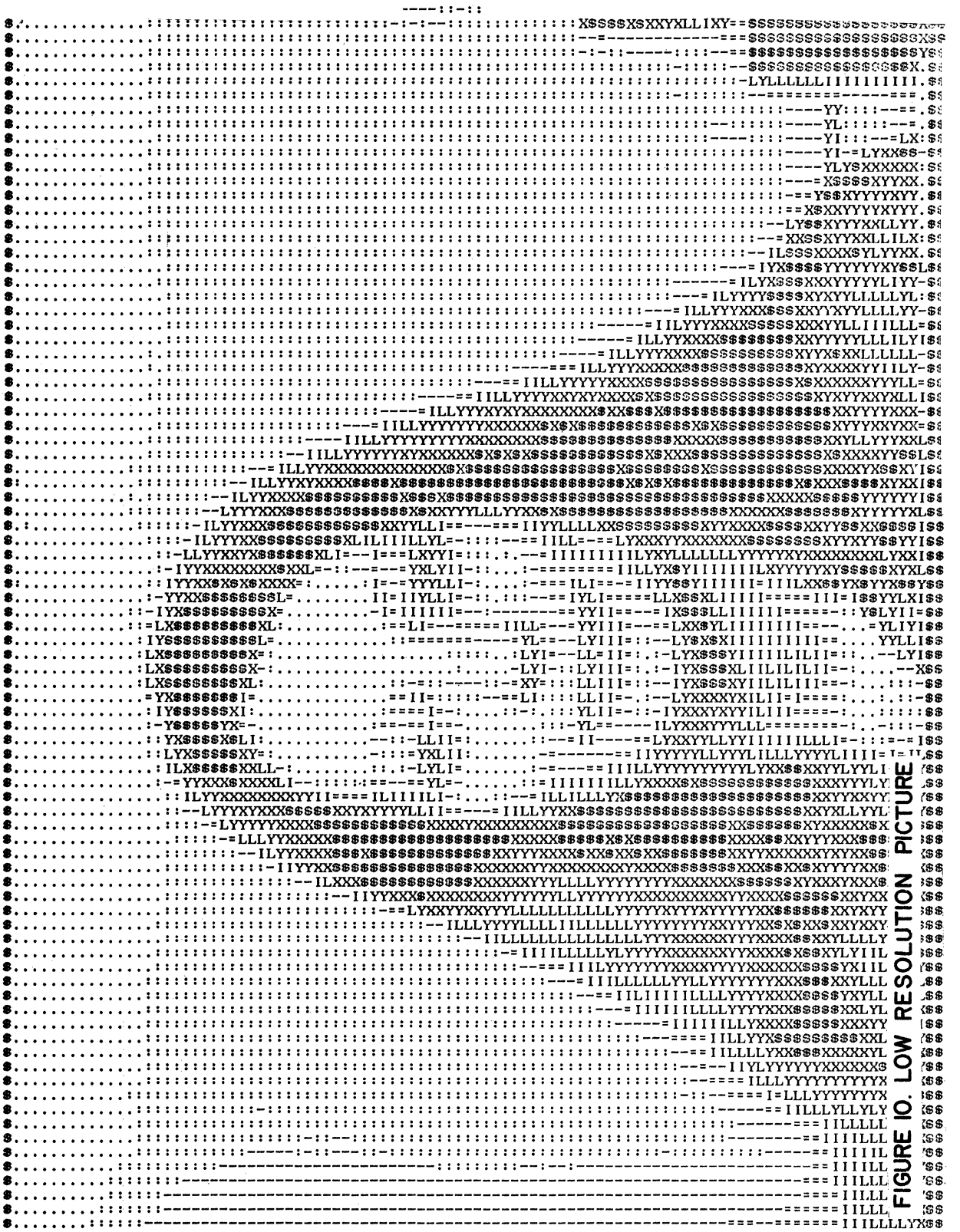


FIGURE IO. LOW RESOLUTION PICTURE

FIGURE 11. HIGH RESOLUTION PICTURE

THE MIK-11: INSTRUMENTATION INTERFACING MADE SIMPLE

Mr. Douglas Abbott
Standard Engineering Corporation
Fremont, California

ABSTRACT

Techniques are discussed for integrating microprocessors into CAMAC, the instrumentation interfacing standard. An LSI-11 based CAMAC crate controller and its supporting modules are described.

CAMAC INSTRUMENTATION

CAMAC (1) is an internationally recognized and supported standard for interfacing scientific and industrial instrumentation to computers. Its inherent modularity makes it extremely flexible and easy to use. Among its principal features are:

1. It is fully specified. Mechanical, electrical, and functional characteristics are sufficiently defined that modules are genuinely interchangeable.

2. Because it is a genuine standard, multiple sources are available virtually by definition. Supplier A's module is guaranteed to work with Supplier B's crate and Supplier C's crate controller.

3. CAMAC was developed by users for users and there exists a diverse community of enthusiastic users in a wide variety of disciplines supporting and extending the CAMAC philosophy.

4. It is well suited to industrial and other less than hospitable environments because of its rugged mechanical packaging and conservative electrical specs.

MICROPROCESSORS IN CAMAC

The rapid development of the microprocessor has led quite naturally to the concept of trating the computer as a component. Programmable intelligence now comes in small packages which can be liberally sprinkled around a system wherever they are needed. The real problem is how to package a microcomputer to be an effective "building block" component. CAMAC, with its well-defined interfaces is an excellent solution to this problem.

There are two basic approaches to incorporating microprocessors into CAMAC based instrumentation. A microprocessor can be "buried" within a CAMAC module to add capability that would not be possible otherwise. The module is considered "smart" but the processor itself is programmed and not visible to the user. Single chip microprocessors are well-suited to this approach.

The other approach is to incorporate a microcomputer into a CAMAC system controller so that it is explicitly visible and is still treated by the user as a general purpose computer. The LSI-11/2 is an excellent candidate for this level of CAMAC integration. Its physical size allows it to conveniently mount in a CAMAC module. The computational power and speed of the LSI-11 are compatible with the data transfer capacity of the CAMAC Dataway. PDP-11's are already widely used in applications suited to CAMAC so that existing software may be easily transferred.

THE MIK-11/2

Our CAMAC version of the LSI-11 is called the MIK-11/2. The basic unit is a triple-width CAMAC module consisting of:

1. The LSI-11/2 processor

2. Console terminal interface and real-time clock

3. Interface to the CAMAC Dataway

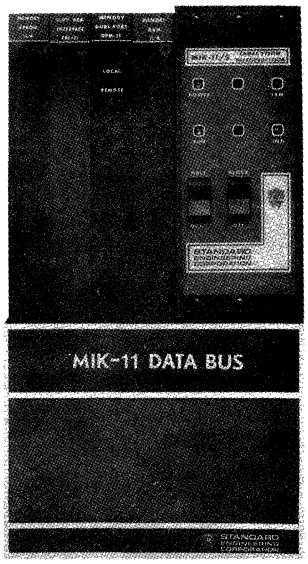


Figure 1: The MIK-11/2

These elements are actually individual single width modules mechanically held together by a common front panel (Figure 1). An additional single width module holds 16K or 32K words of RAM plus a 256 word bootstrap PROM. These four units plus additional peripherals and/or PROM memory are interconnected by a front panel snap-on bus module implementing the Q Bus (Figure 2).

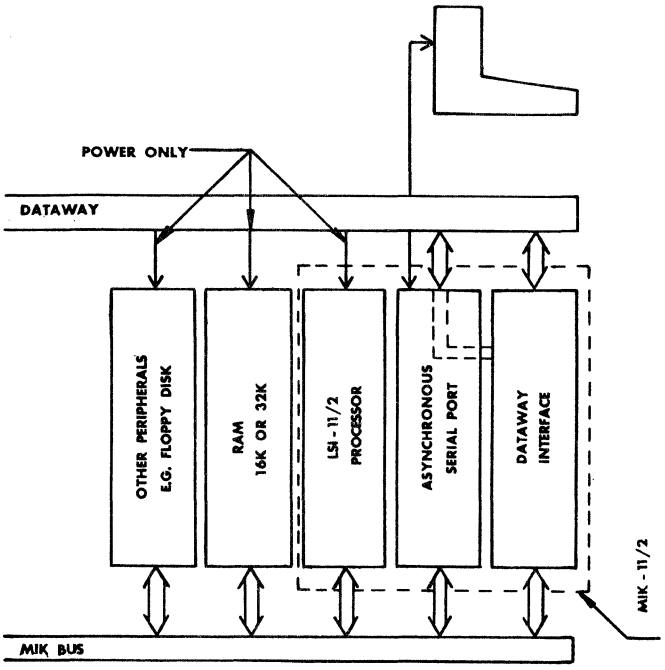


Figure 2: MIK-11/2 Block Diagram

Peripheral interfaces are also packaged in single width CAMAC modules. Asynchronous and synchronous serial ports and floppy disk are currently available. An adapter module allows any dual wide LSI-11 board to be used with the MIK-11. All MIK-11 modules draw power from the Dataway but only the Dataway interface and DMA module described below have any functional connection to it.

CAMAC Interface

The LSI-11 makes an excellent controller for CAMAC. A CAMAC crate may be considered to have a 9 bit address field--5 bits for station number (1 to 25) and 4 bits for sub-address (Ø to 15). The MIK-11's Dataway interface or "crate controller" maps the CAMAC Dataway into a block of 512 word addresses on the Q Bus (Figure 3). That is, each sub-address of each module has a unique bus address. Consequently, any LSI-11 instruction such as BIS or TST can be executed directly on a CAMAC register. Module-to-module transfers can be executed without transferring any data explicitly through the processor and module scans can take advantage of auto-increment addressing.

Each module is assigned its own interrupt vector. Thus when a module needs service, it asserts its individual LAM (Look-At-Me) line which vectors directly to the service routine.

Additional Dataway interfaces may be attached so that a single MIK-11 can control more than one crate. Each crate controller has its own unique address and vector spaces.

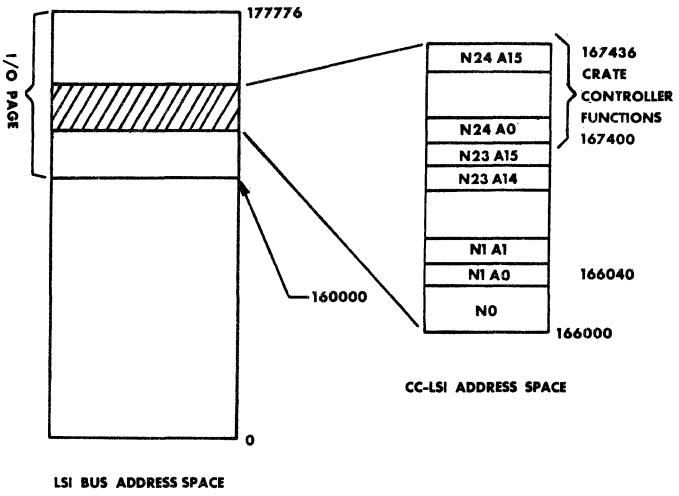


Figure 3: CAMAC Address Mapping

DISTRIBUTED SYSTEMS

The LSI-11, in the form of the MIK-11, is well-suited to distributed control and data acquisition systems. Several forms are possible.

Auxiliary Controller

A recently defined CAMAC standard (2) provides a mechanism for multiple controllers to reside within a CAMAC crate. If the controllers in a conventional CAMAC system (driven by a single host computer) are equipped with an Auxiliary Controller Bus (ACB), additional programmable controllers (auxiliaries) can be inserted in the system as necessary to relieve the host computer of time consuming housekeeping functions. For example the auxiliary may scan a set of alarms and report to the host only when an alarm condition is present.

The MIK-11 can be made into an auxiliary controller by replacing the normal CAMAC controller board with an auxiliary controller board (Figure 4). To the processor this is identical to the normal master controller.

Meaningful communication between the host and the auxiliary through the Dataway is only possible if the auxiliary can also be accessed by the host as a module. For this purpose a DMA module is provided which looks like a module on the Dataway and allows another controller to read and write the MIK-11's memory. This module also permits the MIK-11 to create a demand or interrupt back to the central system controller.

One could conceive of this configuration as a "macro module". That is, the system controller need only talk to the MIK-11 which, in turn, is programmed to process raw data from several input modules and distribute data

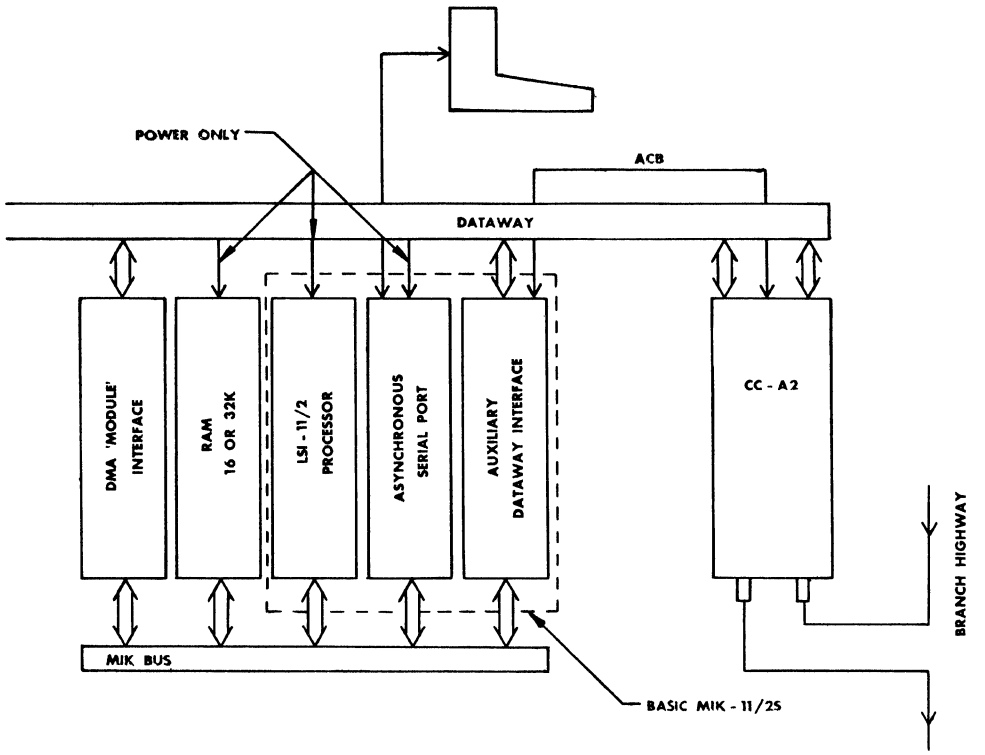


Figure 4: MIK-11/2 Auxiliary Configuration

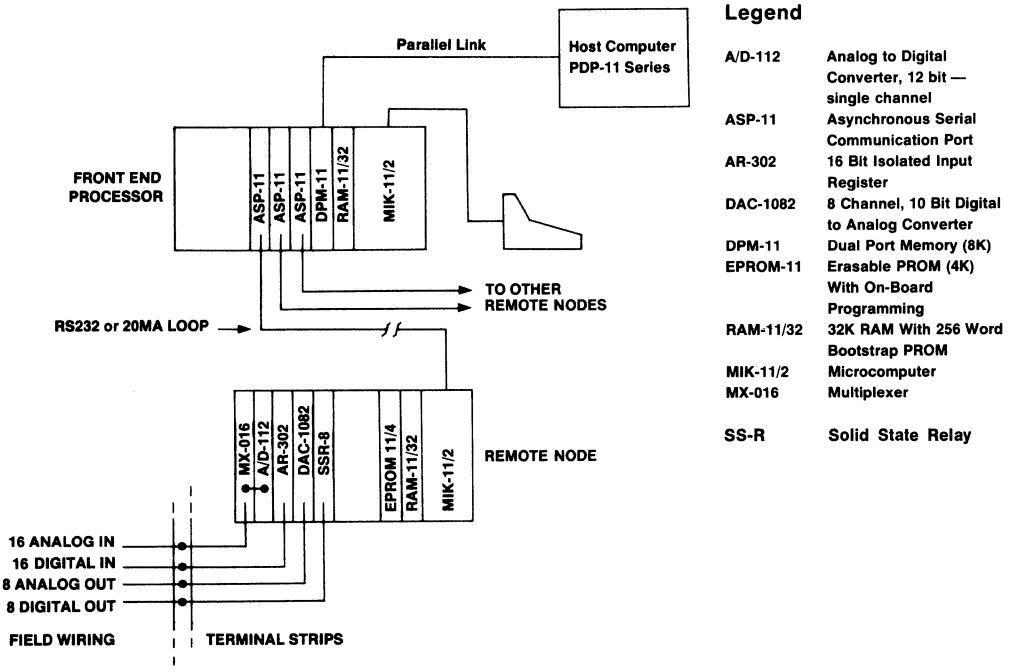


Figure 5: Typical MIK-11/2 Process Control System

to output modules. The implication of this approach is that the system controller's software is simplified in that it need only deal with one module rather than many.

Networks

The MIK-11 can also operate in a conventional network environment utilizing asynchronous and synchronous serial ports. Figure 5 is an example of such a network. In this case a simplification of system controller software is achieved in that it talks to the MIK-11's through DECNET or REMOTE-11 and need not know anything about CAMAC. This is an excellent technique for incorporating CAMAC-based instrumentation into existing systems with minimal impact.

Figure 5 also illustrates another approach to inter-processor communication, the Dual Port Memory. As its name implies, this 8K word block of memory has two independent ports allowing it to connect to two processors. Both processors may "simultaneously" access the DPM. This allows one or more MIK-11's to act as front end processors for larger machines. The example shows a MIK-11 serving as a communications front end.

CONCLUSION

Microprocessors and CAMAC have had, and will continue to have, a synergistic effect on each other in the rapidly developing field of computer based instrumentation and control. CAMAC gives the microcomputer access to a wide range of existing instrumentation combined with a convenient standardized packaging scheme suited to industrial environments. On the other hand, microprocessors enhance the inherent power and flexibility of CAMAC and provide the mechanism for incorporating it into distributed system environments.

Because of its unified bus architecture, the LSI-11 is particularly effective as a CAMAC controller and will no doubt find extensive applications in CAMAC systems.

BIBLIOGRAPHY

1. "CAMAC Instrumentation and Interface Standards", Institute of Electrical and Electronics Engineers, 1976.

2. "Multiple Controllers in a CAMAC Crate", U. S. Department of Energy document DOE/EV-0007, 1978.

HIGH SPEED SQUARE-ROOTING BY IN-FIELD ENHANCEMENT OF A PDP-11/45 FPP

G. A. MOYLE* & N. M. WILSON**
University of New South Wales,
Faculty of Military Studies,
Royal Military College,
DUNTROON ACT AUSTRALIA 2600

It is common in large scale scientific and engineering computer programs to find that square-root operations are frequently required. These calculations are usually achieved by means of time consuming software approximation routines. Significant reductions in overall computing times can be obtained by replacing these routines with special purpose hardware.

This paper describes the design considerations, performance characteristics and unusual interfacing utilized in a high speed, single precision square-root module constructed for the PDP-11/45 computer within the Chemestry Department of the University of New South Wales, Faculty of Military Studies.

The module uses a high speed multiplier, successive approximation register and digital comparitors in a square-root-by-recursion network to achieve an order magnitude reduction in the time required for a square root calculation.

INTRODUCTION

It is common in large scale scientific and engineering computer programs to find that square-root operations are frequently required. Typical of these operations is that required in Molecular Dynamics investigations where intermolecular distances, calculated from

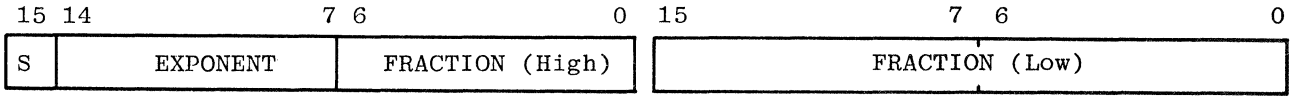
$$r = \sqrt{\sum_{i=1}^{3} | X_i^2 |}$$

are needed. Although the early pioneers of electronic digital computers were of the opinion that square-root operations would be implemented in hardware, most - if not all - general purpose computers forsake hardware implementation in favour of software approximation routines. However, the time consumed by these routines results in long run times, particularily for large programs requiring many square-root operations.

A hardware square-root module constructed and interfaced to a PDP-11/45 Floating Point Processor and the design considerations, operational characteristics and performance of this unit are the subjects of this paper.

DESIGN CONSIDERATIONS

The 32-bit single precision floating point format utilized by PDP-11 computers, of necessity, dominates the design of any hardware device which is to replace software routines. This format, detailed below, uses a 32-bit sign-magnitude convention comprising an 8-bit excess 200_8 exponent and a 24-bit normalized fraction with "hidden" most significant bit.



PDP-11 FLOATING POINT FORMAT

*At present on sabbatical leave with TEKTRONIX, INC., Beaverton, Oregon.

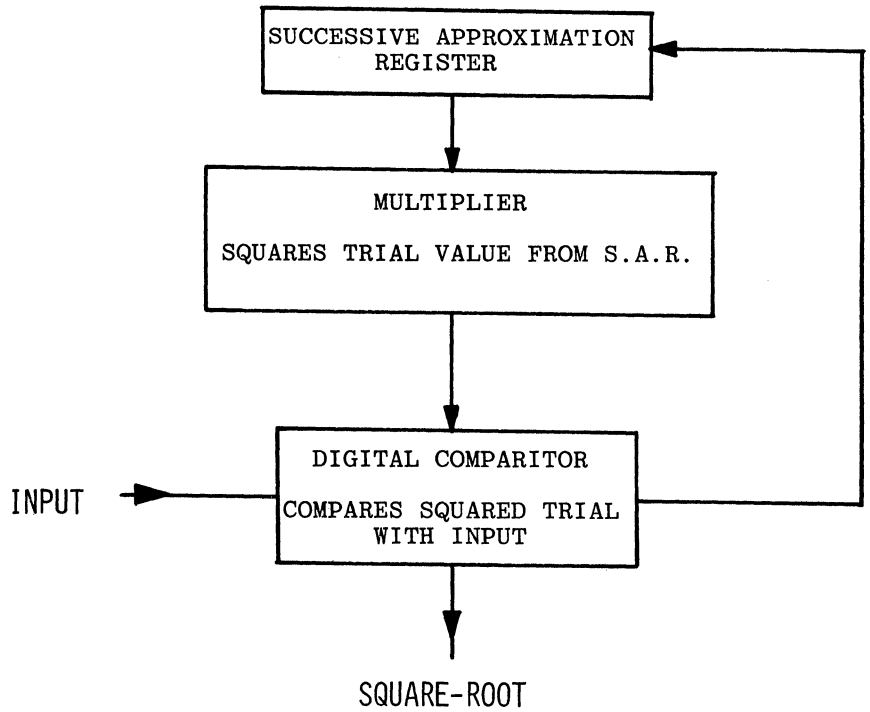
**Presently with DEFENCE RESEARCH CENTRE, Salisbury, S.A.

Physical and mathematical considerations remove any requirement to handle square-roots of negative quantities and hence the obvious simplification of ignoring the sign bit on input and setting the sign bit to positive on output may be adopted. In addition, the normalized format always ensures, except for the special case of zero which may be easily detected and allowed for, that the most significant bit (the "hidden" bit) of the fraction is always 1; that is, the fraction lies between 0.5 and 1.0, and hence, that the most significant bit of the result fraction must also be 1 without requiring recourse to explicit normalization.

Obtaining the square-root of the input number thus requires that the exponent be halved and the square-root of the fraction found. However, since the exponent may be odd, and thus have a remainder modulo-2, the fraction cannot be independently considered. Compensation for an odd exponent may be made by subtracting one from the exponent, prior to halving, multiplying the fraction by two - a simple left shift of the fraction by one bit position - to compensate and the reverse of this procedure on output with the addition of one to the exponent and a right shift of the fraction. This procedure scales all possible input fractions into the range of 0.5 to 2.0 - unnormalized fractions - and by so doing ensures that the most significant bit of the result is always 1. The most obvious advantage of this technique is the removal of any requirement for extensive and expensive multi-position (greater than two) bit shifting arrays for normalization.

Determination of the square-root of the corrected fraction thus becomes the major operational requirement. Various methods based on table or ROM look-up, iterative arrays, binary equivalents of the school taught square-root algorithm, Newton-Raphson techniques etc. can easily be found in a literature survey. However, all these possibilities appear, on closer examination, to require extremely complicated and/or expensive and/or extravagant hardware implementations.

The comparitive recent development of single package, high-speed binary multipliers in various configurations (2x4, 4x4, 8x8, 16x16 etc.) enables an alternate technique based on successive approximations to become competitive with the more traditional methods. This technique of "square-root by recursion" is well documented - see Ghest, "AMD Application Note, October 1972" - and encloses a high-speed multiplier in a feedback network comprising a multi-bit successive approximation register and a digital comparitor as detailed below.



In operation, the successive approximation register is initialized to all ones and thereafter, at every clock pulse, successive bits, from left to right, are set to zero on a trial basis. The resulting word is feed to the multiplier for squaring and a decision on the correctness of a zero in that trial position determined by comparison between the squared trial and the input fraction.

This technique requires, in general, as many trials as there are bits in the square-root - in the case under consideration 24. But, with normalized fractions, the most significant bit is 'a priori' known to be a 1 and thus only 23 trials are actually required. In addition, the most significant bits of the result are determined first and the floating point format used produces the first 16-bit word after only six trials. Thus, partial overlap of computation and readout is possible.

Excess 200_8 exponent halving is a little more complicated than just a simple right shift. Analysis of the required operation shows, however, that the simple transformation shown below suffices.

Squared exponent bits $E_7E_6E_5E_4E_3E_2E_1E_0$

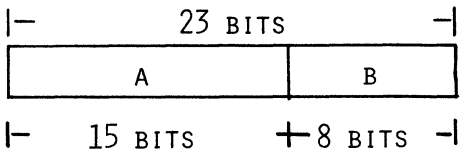
Square-rooted exponent bits $E_7\overline{E}_7E_6E_5E_4E_3E_2E_1$

where the bar signifies complementation.

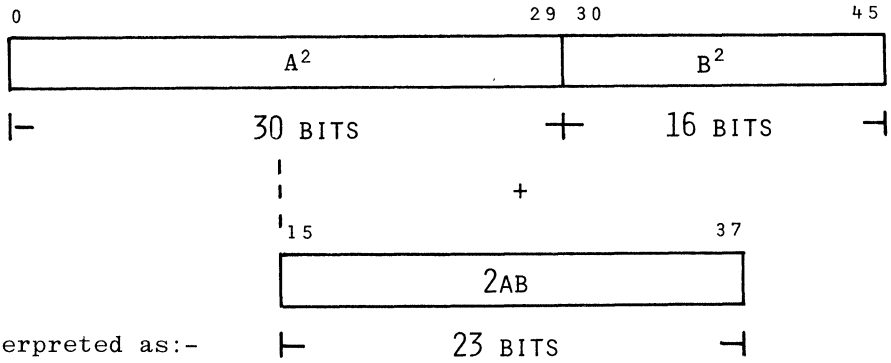
Zero, the only special case, is defined in the floating point format as all those numbers with an exponent of zero. Hence, this condition is easily detected and the appropriate action taken.

Multiplier requirements may be determined by the following analysis:-

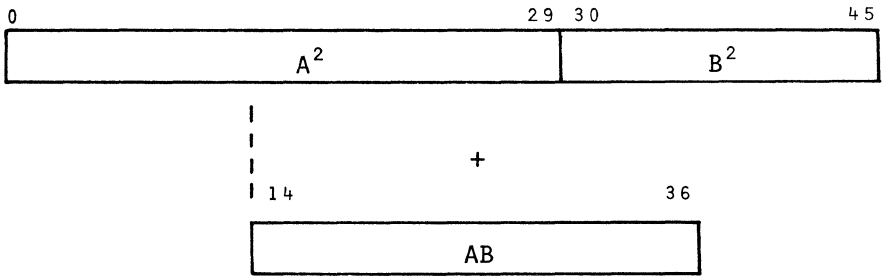
after taking account of the correction needed for the case of an odd exponent, the square-root fraction requires a maximum of 25-bits. On input, the most significant 23-bits of the corrected fraction may be considered in two parts as follows:-



and after squaring:-



this may be re-interpreted as:-



Only the 25 most significant bits are required and thus the B^2 result is redundant, allowing the array to be truncated to two 16x16 multipliers (type TRW MPY 16AJ) with a maximum error of ±1 bit.

INTERFACE TECHNIQUE

The usual means of interfacing a module of this nature to a PDP-11 is via a device such as Digital's DR11-C General Purpose Interface Board. Whilst this technique is satisfactory for most applications, it suffers from the following limitations in the context of this paper - high speed.

a). Two transfer operations are required for single precision floating point

data formatted numbers.

b). The system UNIBUS is utilized for at least two cycles in order to effect this transfer.

and

c). The resulting data transfer rate is relatively slow (15 Mbits/sec).

These limitations can be removed, at least for input operations, by an alternative technique that uses an interface designed such that whenever a number is written into a designated accumulator register of the FP11-B Floating Point Processor that number is also loaded into the input latch of the user's special purpose module. Examination of the FP11-B engineering drawings reveals that the required number is available on the "C" inputs of the accummulator multiplexer and at the wire-wrap pins in slot 3 of the FPP11-B back-plane. In addition, the control logic signals:-

a). Multiplexer Select - S_0 & S_1

b). Accumulator Address - A_0,A_1,A_2 & A_3

c). Function Select - $WE_{1\&2}$, $CS_{1\&2}$

are also avaiable.

For the square-root module, the chosen register is AC0 and the control signals are decoded to produce the two load signals -

$$LOADA = \overline{S_0}.S_1.\overline{A_0}.\overline{A_1}.\overline{A_2}.\overline{A_3}.\overline{WE_1}.\overline{CS_1}$$

and

$$LOADB = \overline{S_0}.S_1.\overline{A_0}.\overline{A_1}.\overline{A_2}.\overline{A_3}.\overline{WE_2}.\overline{CS_2}$$

Note: Two load commands are required since the bytes comprising the number are only available simultaneously if they are shifted from within the Floating Point Processor. If the single precision number is moved into the accumulator register from a location external to the FPP it will be moved as two seperate 16-bit words.

Access to the required data inputs is provided by a user built interface board equipped with low current unified bus receivers and high speed opto-couplers to provide complete electrical isolation inserted into the unused module slots AB15 of the PDP-11/45 together with a small amount of added back-plane wiring - see Figure 1.

The benefits conferred by this interfacing technique can be summerised as:-

a). The UNIBUS is not used for input transfers and thus no interruption to the Central Processing Unit program occurs.

b). Both words of a single precision number may be transferred simultaneously.

c). Complete electrical isolation between the FPP/PDP-11 and the peripheral device may be provided.

and

d). A higher effective data transfer rate is possible (Typically 96Mbits /sec).

There are some limitations to this method:-

a). Data transfer is restricted to the input mode (referenced to the square root module) only, since the output function would require the multiplexing of the inputs to "ACMX" and must be controlled by the FPP or CPU instructions.

b). For other applications where it is not desireable to load the peripheral device with redundant information (see Operational Characteristics), it would be necessary to:-

Either: i). Ensure that the designated register is not loaded until the required number is ready

or

ii). Generate an additional flag or control signal to be used as an enable flag for data transfer.

and

c). The LSB of the single precision mantissa as well as the two low order bytes of the double precision mantissa, if the extra precision was required, are not currently available on the back-plane pins although this facility could be user added to the existing printed circuit cards.

OPERATIONAL CHARACTERISTICS

The prototype square-root module has been connected to the FP11-B FPP and it's performance confirmed. With the square-root calculation initiated by the loading of AC0 (the possible redundant loading referred to above), the clock frequency set to 20MHz and no tailoring of the control signals, the module generates square-roots to an accuracy of ±2 bits in 10.7μsecs. This represents an improvement of a factor of 10 over the 100μsecs required by a typical software routine.

The design is at present far from optimum and it is anticiapted that improvements to:

a). Duration and relative timing of the control signals

b). Increase in clock speed - it has been found that with the existing control signals spurious results do not occur until the clock frequency exceeds 25 MHz.

and

c). Successive approximation register weightings and hard-wiring of the most-significant-bit of the multiplier input to reduce the total number of multiplier iterations

will reduce the computation time to about 8μsecs.

Progress on these improvements has been halted by the arrival and installation of a FP11-C FPP which does not have the required data and control lines readily available.

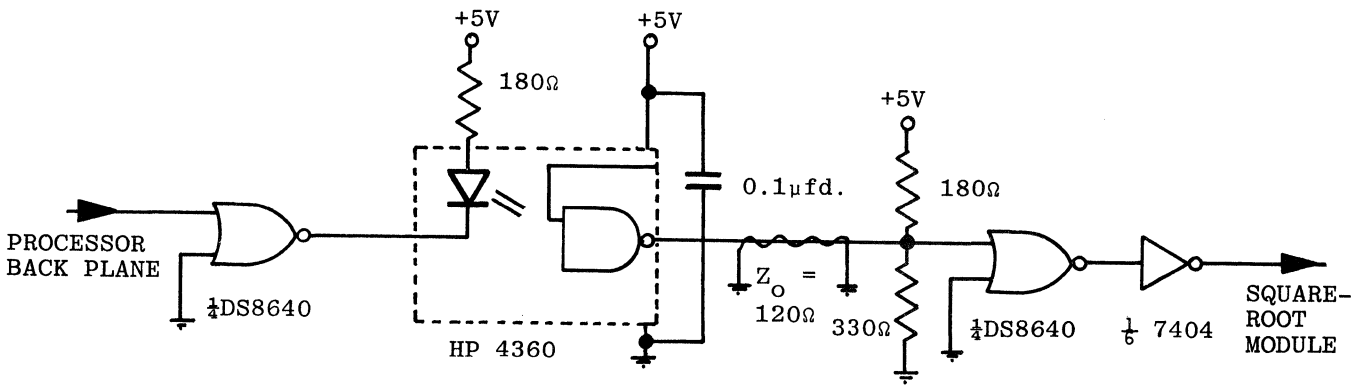
CONCLUSIONS

This paper has described a simple but powerful square-root module and associated interfacing technique that provides for direct access to the accumulator registers of a FP11-B Floating Point Processor. The main advantage of this system is the great reduction in computation time and the ready means whereby an existing machine may have its performance enhanced by in-field modifications. The requirement for minimal additional software management are seen as minor limitations which can be readily overcome in the majority of applications.

The time saved in data transfer and computation can provide for significant reduction in the overall time required in recurrent operations.

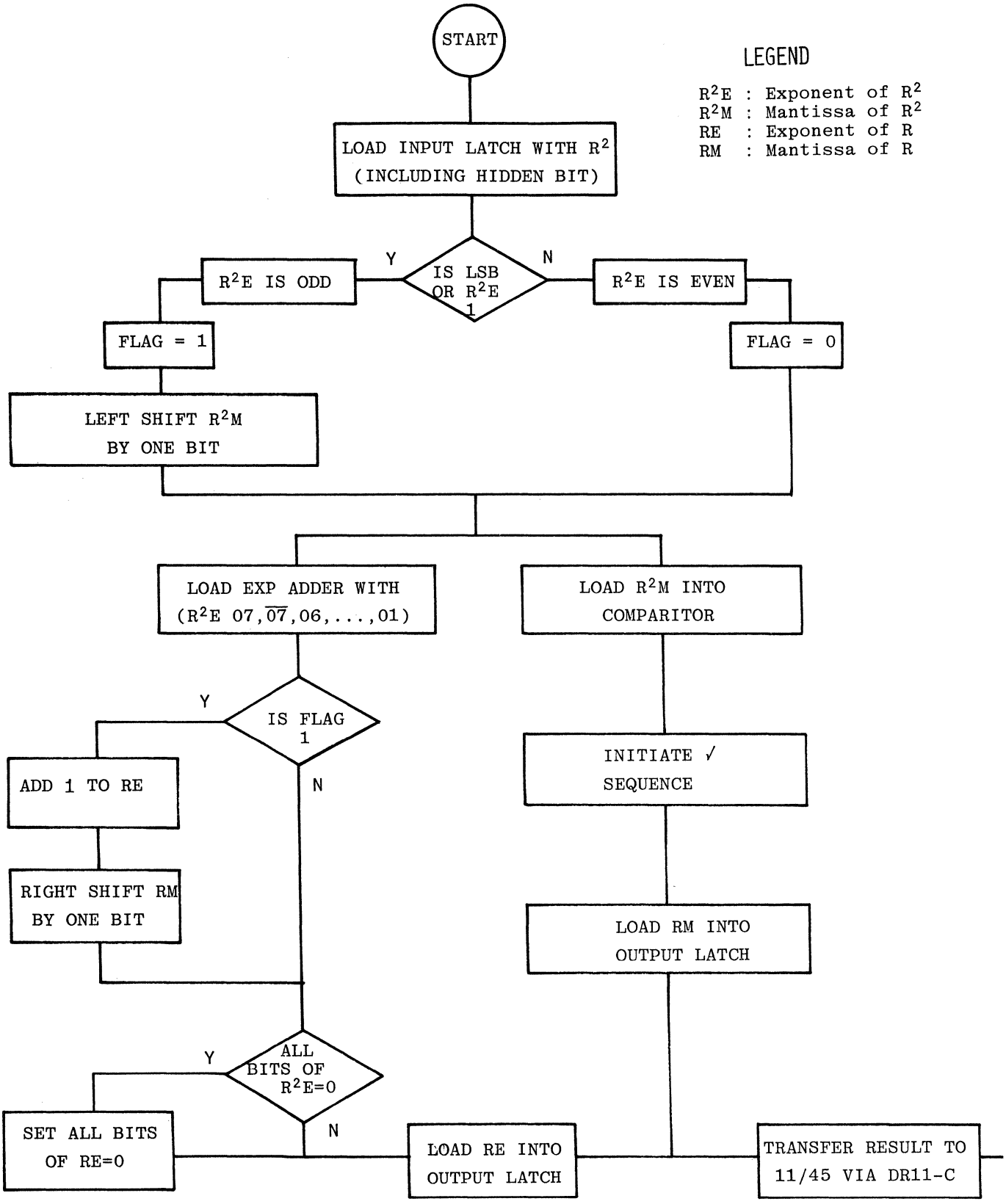
ACKNOWLEDGEMENTS

The authors wish to thank Professor R. J. Bearman of the Faculty of Military Studies for his sponsorship, support and interest and to Dr. D. Jolly of the same department for his assistance during the development period.



LOW LOADING, OPTICALLY ISOLATED
INPUT INTERFACE UNIT

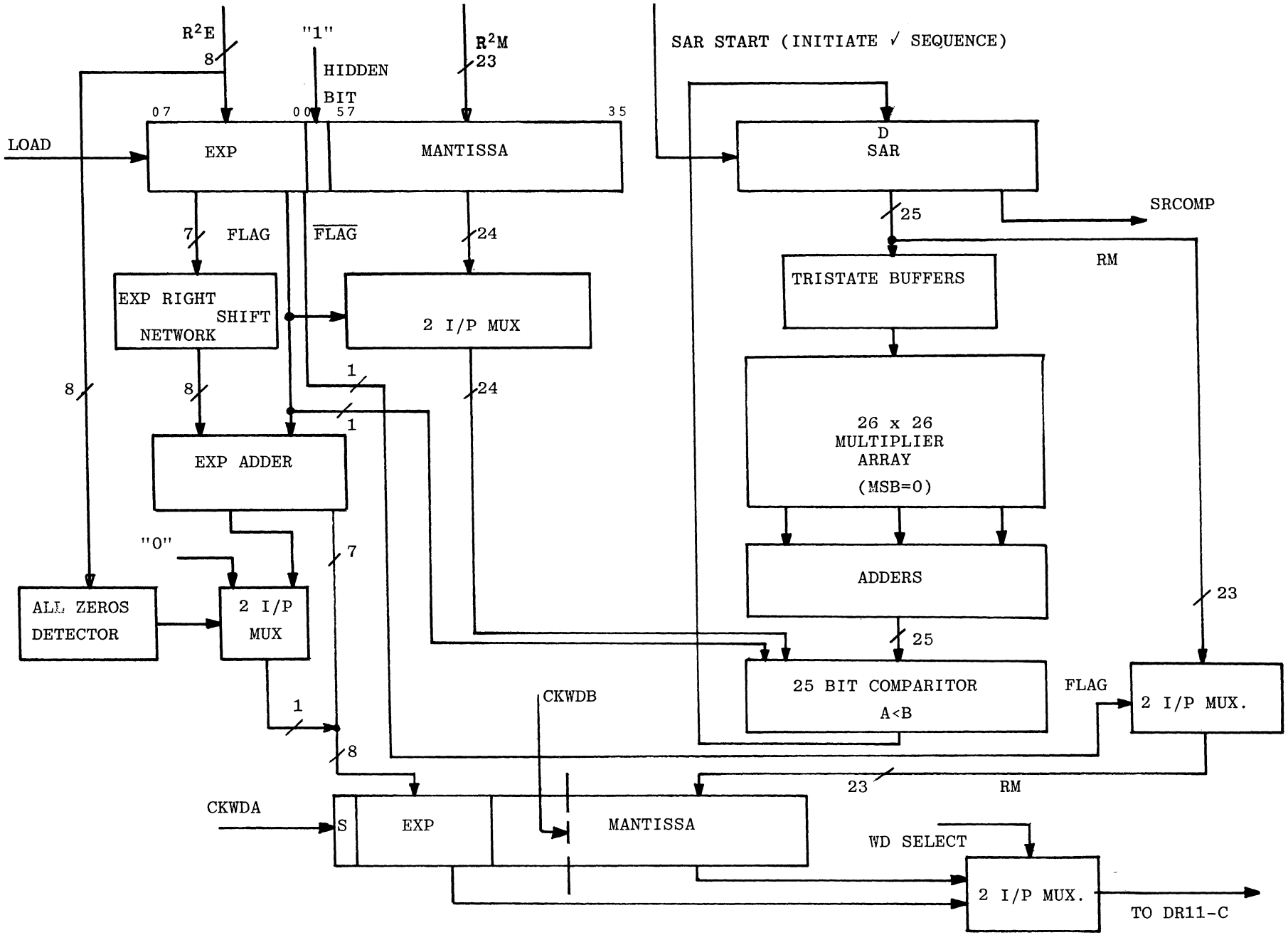
Figure 1



SINGLE PRECISION SQUARE-ROOT MODULE

FLOW DIAGRAM

FIGURE 2



SINGLE PRECISION SQUARE-ROOT MODULE
BLOCK DIAGRAM

FIGURE 3

THE NEUROSCIENCE DISPLAY PROCESSOR MODEL 2

Joseph J. Capowski
Department of Physiology
UNC School of Medicine
Chapel Hill, NC 27514

ABSTRACT

A refreshed vector graphics display processor to generate complex dynamic displays even when driven by a PDP-11/03.

WHAT IS AN NDP2?

From several years of experience, the computer graphics needs of the neuroscience computer user community have been defined.(1) Those needs are to present on a CRT and to copy onto paper those types of displays described below. Because no system, commercially available at a reasonable cost, say less than $10,000, could generate those displays, the Neuroscience Display Processor, NDP, (2) was designed, built, and put into production use as a PDP-11/45 output device in January, 1977. System software was written to allow the NDP to be programmed easily by a FORTRAN programmer. User response to the NDP has been excellent.

In order to increase the speed of the NDP, to simplify it, to allow it to be driven by a processor of power less than that of the PDP-11/45, and to take advantage of new LSI technology, the second version of the NDP, the NDP2, has been designed and built. The NDP2 is a refreshed vector graphics display processor designed to generate complex dynamic displays even when driven by a small processor such as a PDP-11/03.

The salient features of the NDP2 are now described. The NDP2 is driven by a PDP-11 through a 16 bit data cable such as that emanating from a DR11-C or a DRV11 parallel interface. A random access memory stores up to 4096 coordinate triples which define the end points of lines. The triples are multiplied times a rotation and translation matrix. Since the matrix can be updated dynamically, the image can be smoothly rotated. Each triple is checked to see if it has been rotated off-screen, and if so, its entire line is eliminated. The rotated triples are orthographically projected and passed to the line generator. The line generator, a digital differential analyzer design,(3) generates analog deflection ramps for a 17 inch CRT.

TYPES OF NEUROSCIENCE GRAPHICS DISPLAYS

Neuroscience line drawing graphic displays convey information about a neuroanatomical structure or a series of neurophysiological events. This information almost always takes one of three forms: structures, waveforms, and graphs.

A neuroanatomical structure such as that shown in Figure 1, is formed of only short lines. These structures may be two- or three-dimensional. It must be possible to smoothly rotate the structure in order for the scientist to understand its three-dimensional form. The rotation calculation may force some of the parts of the structure off-screen, but since the structure is composed of only short lines, a simple line rejection scheme which rejects an entire line if either end of the line is off-screen can be used without degrading the picture appreciably. An orthographic projection, simply ignoring Z after rotation, may be used to map the three-dimensional structure onto a two-dimensional display surface.

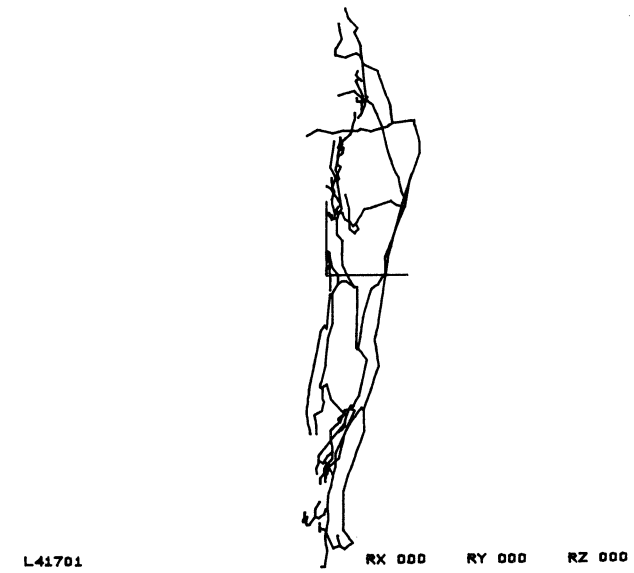


Figure 1. A neuron axon tree from the spinal cord of a cat.

An example of a waveform display is shown in Figure 2. This display is two-dimensional, composed of short lines, and requires no rotation or projection. Depending upon programming techniques, it may be helpful to utilize translation and line rejection facilities to display a window of waveforms from a longer series of them.

Figure 2. A series of physiological electrical waveforms.

An example of a statistical summary is shown in Figure 3. This is a two-dimensional, static display composed of long lines, short lines, and dots. Neither clipping nor projection is involved.

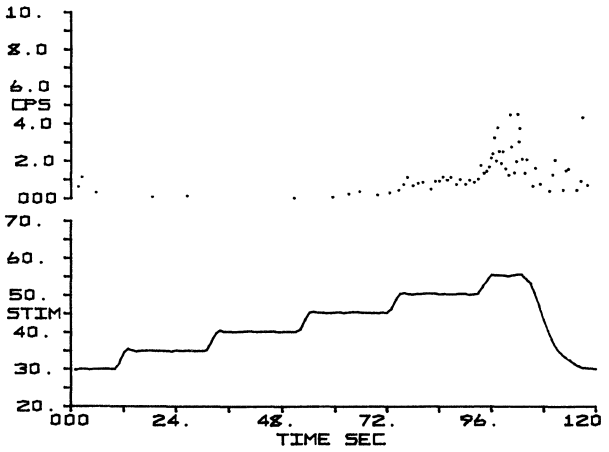


Figure 3. A statistical summary, instantaneous frequency vs. time and stimulus temperature vs. time, of waveforms of the type shown in Figure 2.

NDP2 BUS STRUCTURE

A block diagram of the bus structure of the NDP2 is shown in Figure 4. A 16 bit command is received from the host computer in the upper left corner of the diagram. The most significant 4 bits of the command determine the instruction type. The least significant 12 bits of the command form immediate data, used by the NDP2 during the execution of the command.

A 12 bit count register is used to store the number of iterations required by a drawing instruction. It is decremented, and when it reaches zero, the drawing instruction terminates.

The NDP2 memory, organized as 4096 words of 37 bits each, is used to store up to 4096 coordinates of drawing data. Each coordinate requires 12 bits each for X, Y, and Z, and 1 bit to specify whether a MOVE or DRAW should be performed to that coordinate. A 14 bit memory pointer (MEMP) register keeps track of the current address. Its most significant 12 bits define the current coordinate and its least significant 2 bits define which dimension (X, Y, Z, or MD) is currently under consideration.

The rotation and translation matrix is stored as a 4 x 2 x 12 bit array. Its current address is determined by the 3 bit matrix pointer (MATP) register.

Output from memory passes through a switch (SW) which selects for one input to the multiplier-accumulator either the memory output or the number one. Output from the matrix forms the other input to the multiplier-accumulator. A TRW multiplier-accumulator integrated circuit repeatedly calculates A = A + MEMORY * MATRIX in order to perform matrix multiplication. The rotated and translated coordinate is stored in the 12 bit XDEV, YDEV registers, whose contents are available to the line generator.

If, during the matrix multiplication of a coordinate, overflow occurs, then that coordinate has been rotated or translated off-screen. The MD bit of any coordinate fetched from memory is forced to the value MOVE if overflow has occurred during the matrix multiplication of that coordinate or during the matrix multiplication of the previous coordinate. This modification rejects any line if either of its endpoints is rotated or translated off-screen.

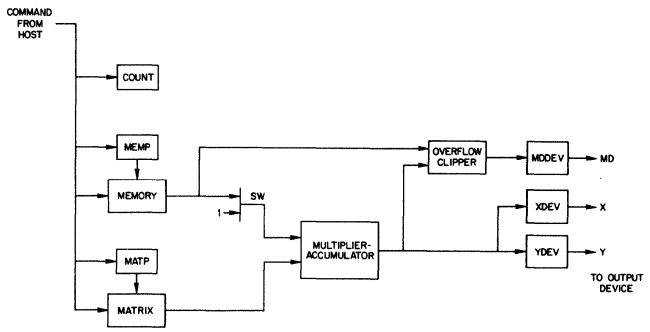


Figure 4. The NDP2 Bus Structure.

NDP2 ALGORITHM

In the block diagram of the NDP2 shown in Figure 5, each rectangle represents one state. Each clock pulse advances the NDP2 from one state to the next.

A push of the master clear button, a clear pulse from the host computer, or the normal termination of a command causes the NDP2 to enter state 0, at which time the NDP2 clock stops, the "done" line to the host computer is turned on, and the NDP2 awaits a command from the host computer. When a command is received, the NDP2 clock starts, and the NDP2 advances into one of five states, depending upon the received command.

If the command is a LMATP, for load matrix pointer, the NDP2 advances to state 5, where the immediate data contained in the command is captured by the MATP register. If the command is a LMEMP, for load memory pointer, the NDP2 advances to state 6, where the immediate data is captured by the MEMP register. If the command is a LMAT, for load

matrix, the NDP2 advances to state 10, where the immediate data is captured into that matrix element currently designated by the MATP register. Then, in state 11, the MATP register is incremented. This scheme facilitates loading of consecutive matrix elements. Similarly, if the command is a LMEM, for load memory, the NDP2 advances to state 12, where the immediate data is captured into that memory word currently designated by the MEMP register. Then, in state 13, the MEMP register is incremented. This scheme facilitates loading of consecutive memory locations.

If the command is a DRAW, first the number of coordinates to be drawn is loaded into the COUNT register. Then, in states 15 through 22, the rotated and translated X coordinate is computed and stored in the XDEV register. In states 24 through 31, the rotated and translated Y coordinate is computed and stored in the YDEV register.

Note that because an orthographic projection, ignoring Z after rotation, is to be performed, no need exists to compute the rotated Z coordinate. Also note that the translation terms are stored as a fourth row of the matrix and that the translation calculation is performed by multiplying (1)(M41) and (1)(M42). Thus the entire matrix manipulation is performed as:

$$[XRT\ YRT] = [X\ Y\ Z\ 1] \begin{bmatrix} M11 & M12 \\ M21 & M22 \\ M31 & M32 \\ M41 & M42 \end{bmatrix}$$

where [XRT YRT] is the rotated, translated, and orthographically projected coordinate, [X Y Z] is the coordinate fetched from memory, and M is the rotation and translation matrix.

If the line generator has not completed the previous line by the time that the NDP2 enters state 32, the NDP2 waits in state 32 for it to finish. When the line generator is ready, the NDP2 advances to state 33 and starts the line generator drawing a line to the current coordinate. Also in state 33, the number of coordinates to be drawn, stored in the COUNT register, is decremented. If another coordinate is to be drawn, its calculation is begun in state 15. If not, the NDP2 returns to state 0.

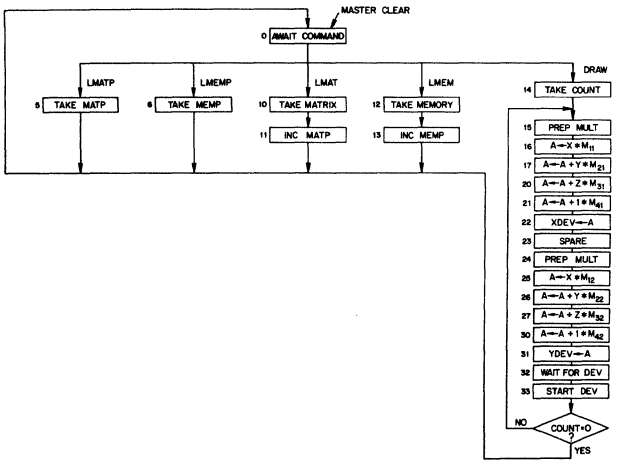


Figure 5. The NDP2 Algorithm.

NDP2 REALIZATION

The NDP2, shown in Figure 6, consists of 135 standard speed TTL integrated circuits mounted on two Augat panels. The NDP2 line generator consists of 100 standard speed TTL integrated circuits plus a pair of deglitched D/A converters mounted on one Augat panel. Standard speed TTL logic was selected to minimize cost, noise, and power distribution problems.

A master clear pushbutton may be used to force the NDP2 into algorithm state 0. A single step switch controls the mode of operation of the NDP2 clock. When the switch is on, each depression of a single step pushbutton generates one clock pulse. This pushbutton may be used to advance the NDP2 through its algorithm one state at a time. Two LED digit displays monitor the current algorithm state. Other LED digits display the current command and the contents of important NDP2 registers. These displays allow the observation of much NDP2 activity and serve in debugging graphics display programs.

The NDP2 control logic is implemented with a register which contains the current algorithm state. This state forms an address into a 32 word by 32 bit ROM, whose outputs control the bus structure and supply the next algorithm state.

The cost of constructing the NDP2 was quite small as it was built primarily from salvaged parts. A conservative estimate for purchasing the parts used in the NDP2 is $2000. The most expensive parts of the NDP2 are three Augat panels, two deglitched D/A converters, a multiplier-accumulator integrated circuit, power supplies, and a mounting rack. The cost estimate should not be compared to the cost of a commercially available graphics display system because it does not include the cost of the labor required to design, construct, debug, and market the NDP2 and its system software.

NDP2 PERFORMANCE

The NDP2 has been running since September, 1978 though not yet in a production environment. The primary application for which the NDP2 will be used, neuron reconstruction, display, and analysis, is presently being developed on the PDP-11/03 to which the NDP2 is attached.

An NDP2 clock period of 0.5 microsecond per algorithm state is currently in use. This period results in a time of 7.5 microseconds to fetch a coordinate from NDP2 memory, multiply it times the rotation matrix, and pass it to the line generator. A line generator period of 0.1 microsecond per iteration is currently in use. This period results in a line generator speed of 43,000 inches per second when the line generator drives a 17 inch CRT. For short lines, less than 75 iterations of the line generator or less than 0.33 inch, the NDP2 runs memory fetch and calculate bound at 7.5 microseconds per line. For longer lines, the NDP2 runs line generator bound. Neuroscience displays have very few lines longer than 0.33 inch. Therefore the NDP2 usually runs fetch and calculate bound and is able to draw 4400 lines at a 30 Hz refresh rate.

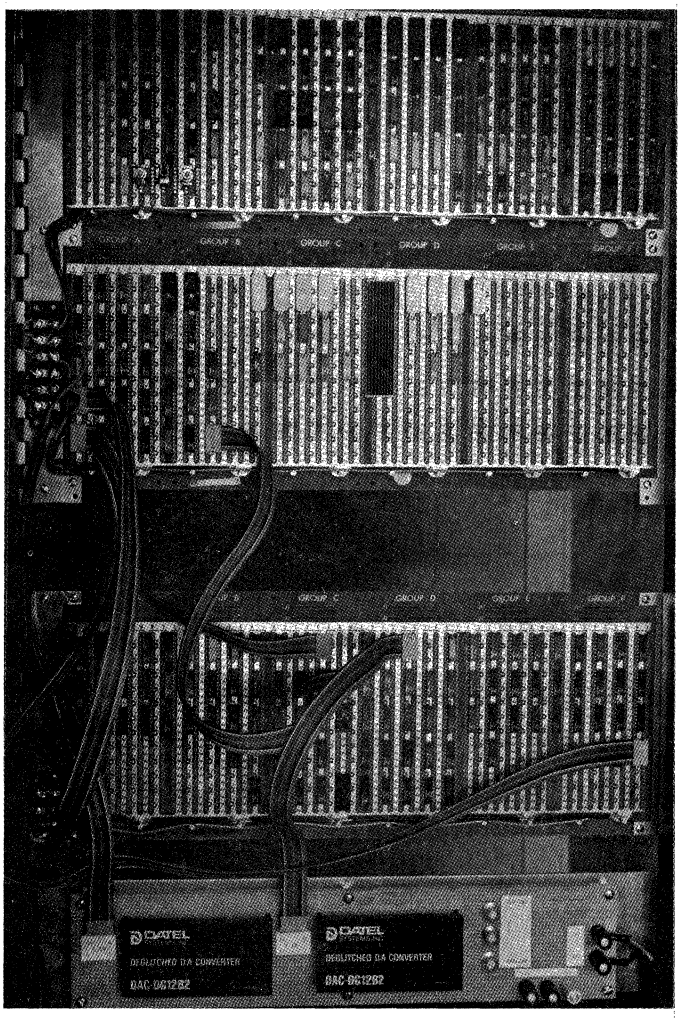


Figure 6. The NDP2.

NDP2 PROGRAMMING

Every attempt has been made to make it as easy as possible for the FORTRAN programmer to use the NDP2 without having to learn much about it. A graphics subroutine package (4) had been written for the NDP and has been updated for the NDP2. The subroutine package is a pyramid or hierarchy of subroutines in which each higher level subroutine calls some of the subroutines of a lower level. A FORTRAN programmer may display a statistical summary such as that shown above with a single call to the highest level subroutine. The highest level subroutine calls building block subroutines which handle individual functions necessary to generate the display. Examples of such functions are the scaling of data into the display file, the drawing and labeling of axes, the display of floating point numbers, and the generation of characters at certain locations in the display. If a programmer wishes to generate some non-standard display, he may call the building block subroutines directly, that is, enter the pyramid on a lower level.

Low level subroutines, called from FORTRAN, allow the programmer to display polygons, dots, or lines from data which he has assembled. 3D structures such as that shown above are drawn by calling these routines. At the lowest level, in assembler language, the programmer may send LOAD REGISTER and DRAW instructions to the DRV11 to command the NDP2. He may check the done bit in the DRV11 status register to see if the NDP2 is ready for another command.

ACKNOWLEDGEMENTS

The author wishes to thank Edward R. Perl, Chairman, Department of Physiology, University of North Carolina, Chapel Hill, for his financial and moral support. Aid in designing and building line generators was received from Roy Propst and Dave Smith of the University of North Carolina and from Fred Rosenberger of Washington University. John McInroy contributed greatly in the programming effort. Neuroscientific material used for examples was contributed by Miklos Rethelyi and Edward R. Perl. This work was supported by the United States Public Health Service through Grant # NS11132 and Grant # RR05406.

REFERENCES

1. J. Capowski (1976) Characteristics of Neuroscience Computer Graphics Displays and a Proposed System to Generate Those Displays. Computer Graphics 10:2, 257-261.

2. J. Capowski (1978) The Neuroscience Display Processor. Computer. November, 1978.

3. W. Newman and R. Sproull (1973) Principles of Interactive Computer Graphics. McGraw-Hill, Inc. New York, N. Y.

4. J. McInroy and J. Capowski (1977) A Graphics Subroutine Package for the Neuroscience Display Processor. Computer Graphics 11:1, 1-12.

THE REAL-TIME CAPABILITY OF THE

EDUCOMP TIMESHARED OPERATING SYSTEM

D. C. BUDDENHAGEN
WESTERN ELECTRIC CO., INC.
OMAHA, NEBRASKA

ABSTRACT

The Educomp Timeshared Operating System (ETOS) is a multi-programming system for the Digital Equipment Corporation (DEC) PDP-8/e, /f, /m and /a Minicomputers. The ETOS allows as many as 16 users shared access to the computer and peripherals. The programming system which the ETOS provides the user is the DEC OS/8 or COS disk based system.

While not comparable to large scale real-time systems the ETOS is designed to handle timeshare and real-time programming. This report discusses the ETOS real-time capability.

INTRODUCTION

The Educomp Timeshared Operating System (ETOS) is a multiprogramming system for the Digital Equipment Corporation (DEC) PDP-8/e, /f, /m, and /a minicomputers. At the Omaha Works, ETOS is installed in a PDP-8/e which forms a part of the Omaha Works Engineering/Manufacturing Timeshare System. The ETOS allows as many as 16 users shared access to the processor (computer) and the peripheral devices. The ETOS monitor provides each user with a virtual computer system, i.e., the ETOS monitor makes it appear to the user that he is running a complete, stand-alone PDP8/e computer system. The programming system which the ETOS provides the user is the DEC stand-alone OS/8 or COS disk based system. (The Omaha Works installation has only OS/8.) Under this programming system the ETOS user has a choice of three high level programming languages; BASIC, FORTRAN II, or FORTRAN IV. PAL8, the assembler language, is also available as well as the normal OS/8 utility programs. For a complete description of the OS/8 programming system, the user should refer to the OS/8 Handbook. Detailed information on the ETOS can be obtained from the ETOS Version 5A System User's Guide.

While not comparable to large scale real-time systems, the ETOS is designed to handle timeshare and real-time programming. The remainder of this report discusses the ETOS real-time capability.

A DESCRIPTIVE ANALOGY

The ETOS can be thought of as a network of DEC PDP-8/e minicomputers sharing common disk storage. Each user job would correspond to one of the computers and the ETOS monitor would be the executive machine which controls the overall network operation and allocates network resources to the individual machines (user jobs) on an as-needed basis. Since the minicomputers are virtual machines which occupy the same real machine, the most important network resource which the executive machine (monitor) allocates is processor time. In this timeshare network, each of the virtual machines, except the monitor, is allocated processor time in turn. During its allocated time, a virtual machine may execute program code or make requests to the monitor for the use of other network resources. Since the monitor controls the network operation, it must operate in real time, i.e., the monitor must respond to the demands of the network as they occur. To enhance the real-time capability of the monitor, a portion of it is permitted to occupy the two lower fields of memory in the real machine at all times. On the other hand, a user job may be temporarily stored on disk while another user job is loaded into real machine memory during its allocated processor time. (This process is referred to as swapping.)

The ETOS allows one user job (virtual machine) to run in real time as well as in the normal timeshare mode. The real-time portion of the user job can be referred to as the foreground job and the timeshare portion as the background job. The user foreground job may use one or two fields of memory. When initiated, the foreground job actually forms an extension of the resident monitor and, as such, is permitted to occupy real machine memory at all times, i.e., the foreground job, once initiated, is not swapped. The background job may be written into the same field(s) of memory as the foreground job, into memory fields not used by the foreground job, or both. Therefore, the background job may be subject to no swapping, partial swapping, or total swapping. In any event, the background job is treated by the monitor as a normal timeshare job. In terms of the multi-computer network analogy, the user job which is running real time can be considered to be a machine which is running two programs, a foreground job and a background job. The foreground job is allowed to respond to external events immediately whereas the background job is queued for use of network resources in the same way as the rest of the network computers. Since they occupy the same machine, the obvious advantage the background job has that the other user jobs do not have, is its ability to

communicate directly with the foreground job and thereby with the real world.

REAL-TIME MULTI-TASKING

Most probably, more than one user job will want the ability to run in real time. Since the ETOS allows only one real-time job there is a conflict which must be resolved. One solution is to simply have the users take turns running their real-time jobs. If the real-time jobs to be run can accommodate this type of scheduling the solution is acceptable and very easy to implement. More than likely, however, the real-time jobs will conflict with each other and a better solution must be found.

Generally speaking, the portion of any user job which must run in real time is quite small and consists of the minimal amount of coding required to successfully handle a peripheral device operating on a program interrupt basis. Even though only one user job is allowed to run in real time, the foreground job can be programmed to handle several real-time tasks. In this configuration, the background job coordinates the transfer of data to and from the other user jobs which are using one or more of the foreground tasks. In effect, the background job becomes a real-time monitor operating within the confines of the ETOS. Unfortunately, communication between user jobs, particularly the first contact, is not easily accomplished. A brief description of ETOS software architecture will permit a more effective discussion of this communication problem.

THE ETOS SOFTWARE STRUCTURE

The ETOS monitor operates on four levels of software priority. These priority levels have nothing to do with interrupt priority, so, to avoid confusion, they will be referred to simply as levels.

Level 0 software is executed immediately following an interrupt. The system interrupt is off, the user flag is cleared, and the instruction field and data field are both set to 0. Level 0 software checks for conditions which may have caused an internal interrupt, i.e., an interrupt resulting from a user executing a virtual machine instruction which requires monitor intervention. In some cases internal interrupts are also processed in level 0 software.

Level 1 software is used to identify and sometimes service external interrupts. External interrupts are caused by peripheral devices. The processor state is the same as for level 0 software except that the data field is set to 1. The foreground task(s) of a real-time user job is considered to be level 1 software and is executed immediately upon entry to level 1.

Level 2 is the software level on which most of the resident monitor is executed. The processor state is the same as for level 1 software except the system interrupt is on. This means that level 0 and level 1 software can be re-entered if an interrupt occurs while level 2 software is being executed. To prevent degradation of system interrupt handling, any user written real-time task(s) should spend a minimum amount of time on level 1. Monitor calls which perform level shifting between levels 1 and 2 are available to the user.

Level 3 is designated for user jobs. While operating on level 3, the system interrupt is on and the user flag is set. In addition, level 3 jobs are subject to swapping. There are two types of level 3 jobs, privileged and non-privileged. Privileged jobs are permitted to change the normal manner in which level 3 software is executed whereas non-privileged jobs are not.

There are three ways in which a privileged job can change normal level 3 operation. The simplest of these is through use of the CUF (clear user flag) instruction. Execution of this instruction causes the user program to run in executive mode. In essence, running in executive mode means that all multi-user functions are inhibited, i.e., the user has complete control of the real machine. (More information about executive mode operation can be found in the PDP8/e, PDP8/m & PDP8/f Small Computer Handbook on page 5-18.) Since the system interrupt is on and the system clock is running, the user is subject to swapping. To return to normal level 3 operation, the SUF (set user flag) instruction is executed.

The second method which a privileged user job can use to change normal level 3 operation is to request the monitor to execute the job on level 2. This is accomplished by executing the RMON instruction. Following the RMON instruction, the user is running in executive mode and, since the machine is running level 2 software, no swapping will occur. The user must not run on level 2 for any extended period since the level 2 queue may overflow, causing the system to crash. A call to a monitor routine is used to return to normal level 3 operation.

The third way of altering normal level 3 operation is for the privileged user to start a real time job by executing a HOOK instruction. Following execution of this instruction, the user foreground job becomes memory resident and is treated as a part of the level 1 monitor but the background job is still executed as level 3 software. The UNHOOK instruction is used to stop a real-time job.

In addition to being able to run real-time or executive mode software, the privileged user has at his disposal several system functions and monitor routines which can be used to access all user and system software. In addition, they can, to some extent, control the execution of user jobs other than his own. It is through the use of these privileged functions and monitor routines that the user job running in real time can set up an effective means of communication between the background job (real-time monitor) and other user jobs which need to use the foreground task(s). More information about the ETOS software structure and the monitor functions and routines available to the privileged user can be obtained from the ETOS Version 5A System Manager's Guide.

HOW TO COMMUNICATE

The user real-time application will determine the type and amount of interaction between the user program and the real-time monitor. In fact, some

applications may require no interaction whatsoever. For example, a data collection application may be implemented by having the foreground task handle the data collection device interrupts while the background task stores the collected data in a file on disk. The user job would then simply access the disk file and process the data as needed.

Other applications may require only infrequent interaction which could be handled in an indirect manner. This method would have the user job and the real-time monitor exchange information by means of a disk file accessible to both. This type of information exchange would require the real-time monitor and the user job to periodically check the disk file for new information. As the frequency of these file checks increases, however, the required disk read/write time increases accordingly and can seriously degrade overall system performance. Therefore, for those applications which require high-speed exchange of instructions or small amounts of data, the utilization of a direct means of communication is desirable. In terms of the multi-computer network analogy, direct communication would be equivalent to providing the individual computers the ability to access the machine resisters and memories of other computers.

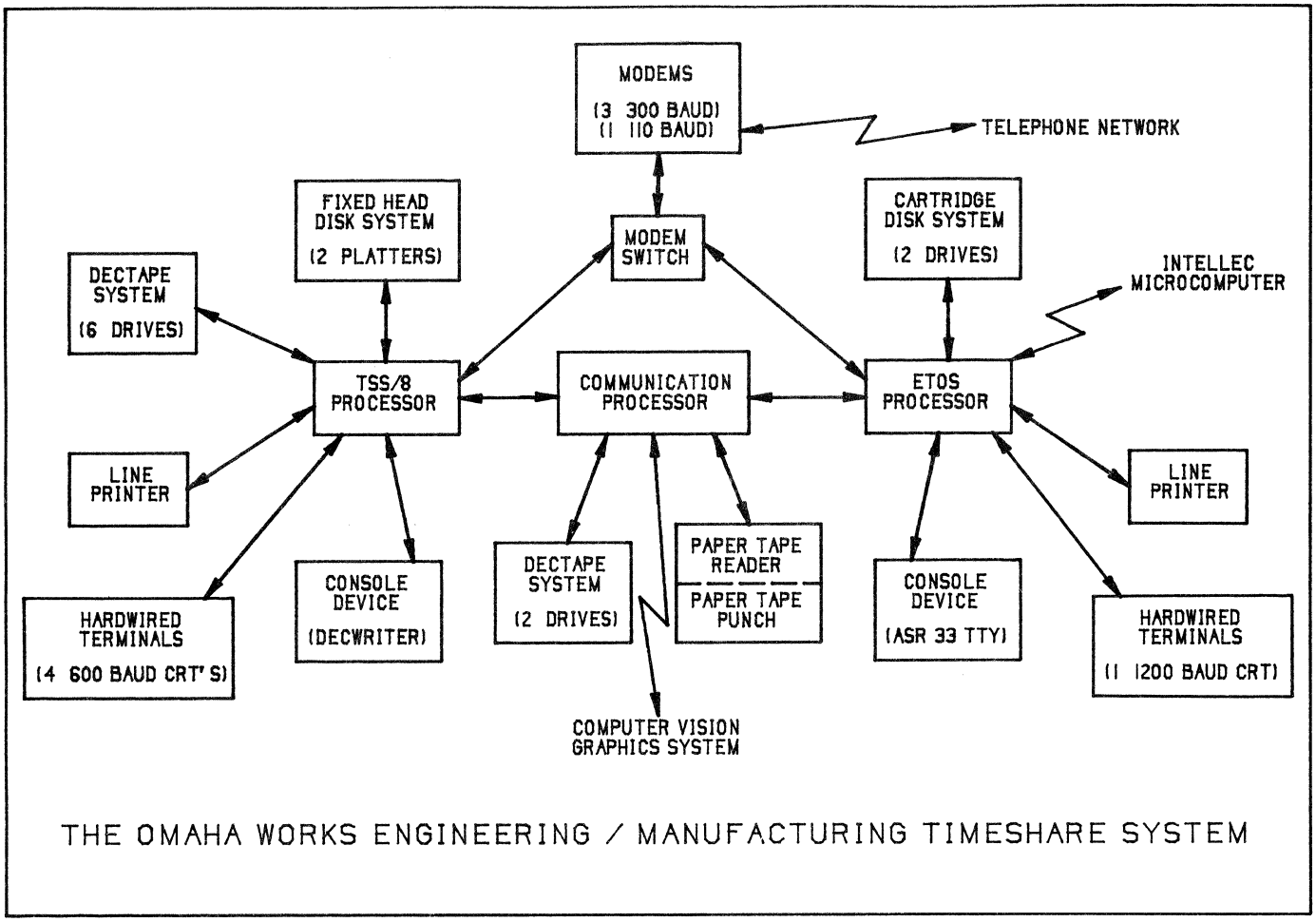
It is expected that most ETOS users would not be privileged. To do otherwise would certainly jeopardize system operation. Unfortunately, a non-privileged user job has absolutely no way of directly communicating with another job. Therefore, the user is forced to use some type of indirect communication to establish contact with the real-time monitor. A disk file could be used for this purpose but a more efficient method is available. Associated with each user job is a 32 word job data block. Non-privileged user jobs are allowed to read any job data block a word at a time but they can only change selected words in their own block. The last five words of the job block forms a small buffer which can be written into by the user job through the use of the TXTSET system function. This buffer area is intended to hold the name of the current program and is only used by the SYSTAT program when listing the system status. The use of this buffer area for other purposes causes no system problems even though SYSTAT may print out some strange program names. Therefore, the user job is free to place a message into this buffer area which will be recognized by the real-time monitor as a request for service. Of course, the real-time monitor must periodically check the user data blocks to determine if real-time service is being requested. Even though this is an indirect method of communication, system operation is not degraded since disk read/write time is not required. The frequency at which this checking is done is determined by the urgency of the user job. Generally, the time lag between the user request for real-time service and the recognition of the request by the real-time monitor is of little consequence. However, once the real-time monitor recognizes the request and initiates the real-time service the rate of interaction between the user and the real-time monitor can be adjusted to fit the job requirements.

Recall that the real-time monitor is a privileged user. (Only privileged users can run in real time.) Being a privileged user, the real-time monitor has complete access to the resident ETOS monitor and can control some aspects of user job operation through the use of privileged monitor functions. Among these control aspects is the ability to change the contents of the user's virtual machine resisters, put the user job to sleep for an indefinite period of time and then wake it up as needed, and even make the user job a privileged job. However, since the user job is subject to swapping, the real-time monitor can not directly affect the contents of the user's virtual machine memory since it is never sure where the user program is in real memory. If it is necessary for the user job to directly interact with the real-time monitor, the real-time monitor must make the user job privileged. This can be done immediately after initial contact by simply having the real-time monitor set the user's privilege word in the job data block to an appropriate value. A privileged user job has an advantage over the real-time monitor in terms of direct communication. Since the foreground job is not subject to swapping, the privileged user job can find it in real memory. Once found, the user job can read from or write into that memory area, thereby communicating directly with either the foreground or background task. This type of operation may be necessary for applications requiring large amounts of information to be quickly passed between the user job and the real-time job. To illustrate how the real-time capability of the ETOS can be used, the first real-time application implemented at the Omaha Works will be discussed. As the application is examined, alternate methods of performing various functions will be indicated.

AN EXAMPLE

On the following page is a block diagram of the Omaha Works Engineering/Manufacturing Timeshare System. The system is comprised of two independent operating systems which are loosely tied together through the Communication Processor (ComPro). One of the operating systems is the DEC TSS/8 time share system which has been in use at the Omaha Works for several years. The other operating system is the ETOS. The ETOS, as configured, has no peripheral devices directly interfaced to it with the exception of a line printer and, of course, the cartridge disk system. However, before installation it was determined that some method of storing and retrieving ETOS and OS/8 files on a media other than magnetic disk was needed. To meet this need it was decided that a parallel bus interface would be used to provide a real-time interface between the ETOS and the ComPro. The ComPro would have interfaced to it a dual DECtape system dedicated to ETOS users and a high speed paper tape reader and punch which would be shared by the TSS/8 and ETOS users. In addition, to facilitate future numerical control support applications, a data link from the ComPro to the Computer Vision graphics system was provided. This data link was also dedicated to ETOS users.

To implement this real-time application it was necessary to write two application programs; the Real Time Executive (RTE) and the File Transfer Program (FTP). The RTE consists of a foreground task and a background task. The foreground task handles the real-time interrupt service for the parallel data bus connecting the ETOS processor to the ComPro. As currently written, the foreground



task is quite short and runs completely on level 1. As more real-time tasks are added, it will be necessary to execute portions of the tasks on level 2 to prevent degradation of interrupt servicing. The background task is the real-time monitor and acts as the interface between the FTP and the foreground task. The FTP is the program which is run by a user whenever he wishes to transfer a data file between his account and one of the peripheral devices controlled by the ComPro. The FTP requests transfer commands from the user and relays them to the real-time monitor. The real-time monitor interprets the commands and, with the help of the foreground task, effects the data transfer. Upon completion of the transfer, the real-time monitor returns control to the FTP and the user can terminate the program or make additional data transfers.

Since the real-time job must be privileged, a special account for real-time use only was established. The RTE program and a special ETOS file named COMLINKDAT were placed in this account and are not available to general users except through the FTP.

To initiate a file transfer, the user simply runs the FTP. The program goes through an initialization routine and then calls the OS/8 command decoder. The user inputs the file transfer commands and then waits for the commands to be executed. To make the file transfer commands available to the real-time monitor, the FTP stores them in the COMLINKDAT file which resides in the same account as the RTE and then initiates the data transfer by contacting the real-time monitor.

In implementing this application it was assumed that all users would be non-privileged. This required the initial contact between the user program and the real-time monitor to be accomplished through an indirect means. Initially, it was intended to use the disk file method of indirect communication. However, when this method was tried it was immediately apparent that the disk read/write time required for the RTE to locate new users would seriously degrade system operation. As a result, it was decided that the file transfer commands would be given to the RTE through the disk file but all other communication would be accomplished through the use of the test area in the user job block.

Normally, the first four words of the test area are used to store the OS/8 file name and extension of the current program; in this case FTP.SV. The fifth word is unused. The FTP stores the number of the COMLINKDAT file block which contains the transfer commands in the fifth word location and requests real-time service by changing the current

program name to FTPX.SV. After the test area is changed (by using the SETTXT system function) the FTP goes into a continuous series of 1 second waits. If at the end of 10 seconds, the real-time monitor has not recognized the FTP request for service, an error message is displayed to the user and the FTP returns control to the OS/8 monitor.

After the foreground task of the RTE is started, the real-time monitor checks all system users every 5 seconds to determine if any users are running a program called FTPX.SV. These checks are made by running in executive mode and reading the appropriate areas of the user job data blocks located in the resident monitor. (The job data blocks start at location 200 in field 1 but the first two jobs are part of the ETOS monitor; KMON and DMON. The first "real" job data block starts at memory location 10300.) The real-time monitor checks the first word of the job data block before reading the text area. If word 1 is 0 the job data block is not being used and no further checks in that block are necessary. If word 1 is not a 0, the real-time monitor checks the job data block test area. When the real-time monitor finds a job requiring service it resets the user job data block test area to FTP.SV. The FTP recognizes this change as a signal that real-time service has begun.

After recognizing a request for service, the real-time monitor establishes its own four word job block for the user. The first word in this block is a user status word. Since several users could request real-time service simultaneously, a queueing system was necessary and the status word is used to indicate to the real-time monitor the user status in the queue. The second word in the block contains a pointer to the last word in the user's job data block. This word location is used to pass control information to the user so a pointer to this location is advantageous. The third word holds the number of the COMLINKDAT file block in which the user's file transfer commands are stored. The fourth word stores a pointer to the user's virtual program counter. Whenever the real-time monitor wants to send control information to the user, it loads the coded information into the last word of the user's job data block and then changes the contents of the user's virtual program counter. In this way the real-time monitor forces the FTP, which is still running continuous 1 second waits, to run a routine which interprets the last word in the users job data block and takes appropriate action.

An alternate approach would have been for the FTP to put itself to sleep indefinitely after determining that the RTE had recognized its request for service. This is done by using the WAIT system function to set one of the user status resister wait bits in the user job data block. Then, whenever control information must be sent to the user, the real-time monitor could set the user's virtual program counter to the desired address and make up the user by calling the monitor wake up routine. This method has the advantage of immediately placing the user job in the run queue whereas the method employed may require an additional 1 second wait before the user is queued. In addition, the control information could be loaded into the user's virtual accumulator, eliminating the need to use the job data block text area.

In those instances where it is necessary for the FTP to return control information to the RTE, the FTP loads the coded information into the last word in the user's data block and returns to the continuous 1 second waits. The next time the real-time monitor checks the user, the returned control information is recognized and acted upon.

Any alternative method for returning control information to the real-time monitor would require the user to be made temporarily privileged. The real-time monitor can do this by setting the user's privilege word in the job data block to an appropriate value. The real-time monitor could then provide the user with the real field location of the memory resident foreground task. This would allow the user to write directly into a specified area of that memory field, eliminating the need for the real-time monitor to periodically scan the user job data blocks. Since the user would be privileged, the user program could also locate the real-time job data block and modify any of the locations including the virtual machine resisters. However, since the real-time monitor is intended to serve several simultaneous users, any meddling in this area must be done carefully to insure that no interference is generated between users.

The actual data transfer between the user account and the peripheral devices is handled by the real-time monitor. The transfer commands are read from the COMLINKDAT file, interpreted, and are normally carried out by the real-time monitor with no additional help from the FTP or input from the user. There are certain error conditions which do require user input to resolve. In these instances, the information is treated as control words and transfered via the user job data block text area. In this application, the disk reads required to obtain the transfer commands are of no consequence in terms of total time required since the data transfer involves a continuous series of disk reads or writes. However, for applications which do not involve reading or writing large amounts of disk data but do require the transfer of small amounts of data between the user program and the real-time monitor, it may be advantageous for the user to be temporarily privileged so that data transfer can be made in memory without resorting to disk file transfers.

SOME LIMITATIONS

The ETOS cannot handle all types of real-time applications. The most obvious limitation is on-line disk storage. The ETOS is limited to four cartridge disk drives and has a maximum on-line data storage capacity of approximately 8.5 million bytes. This eliminates any application which requires a massive data base.

Another less obvious restriction results from the fact that the ETOS is primarily a timeshared multi-programming system. As such, each user in turn is allocated a specific amount of time during which his program is executed. The ability of one job to run in real time does not alter this basic operating principle since the real-time background task is treated as a normal timeshare job. Even though individual jobs can be granted extra

run time, the order in which the jobs are run cannot be altered, i.e., there is no method of assigning task priorities. As more jobs are run, the time interval between run times for any specific job increases. If the real-time application is time dependent to the point where the background task must run within extremely short intervals, it is not a suitable application for the ETOS.

Finally, those applications which require continuous, high speed service cannot be implemented on the ETOS. High speed in this case is defined as an interrupt rate greater than can be successfully handled by the real-time foreground task. Normally, the ETOS monitor will recognize an interrupt and give control to the foreground task within 100 usec.after the occurrence of the interrupt. Allowing 40 to 50 usec. for the foreground task to find and service the interrupting device, a maximum total interrupt rate for all foreground tasks would be approximately 7000 interrupts per second. Servicing interrupts at this rate would leave little time for normal timeshare operation and could be tolerated only for short periods of time. Since the ETOS is primarily a timeshare system, a realistic average real-time interrupt rate would be 1000 to 2000 interrupts per second. This would allow real-time service without seriously degrading timeshare performance.

SUITABLE APPLICATIONS

In spite of the restrictions on ETOS real-time use, there are many applications which can be successfully implemented. One type of application was discussed earlier; the use of a real-time task to handle peripheral devices not supported by the ETOS monitor. An application of this type could be used to provide additional on-line storage for another real-time application which could not otherwise be implemented.

Another type of application which could be implemented is data collection. As an example, assume it is desired to collect manufacturing process data from 20 machines each of which performs a manufacturing operation at a rate of one machine cycle per second. Further assume that the data will be transmitted from each machine to the ETOS in 10 byte bursts over an asynchronous serial line interface at a rate of 240 bytes per second. The ETOS real-time task would have to service a maximum of 200 interrupts per second. Allowing 300 usec. to service an individual interrupt implies the worst case total time for interrupt handling is 60 msec. per second or 6% of total time.

Reversing the above example illustrates another type of application which could be implemented on the ETOS; machine control. Machines which currently receive their control information from paper tape, magnetic tape, punched cards, or other similar types of storage media could be serviced by an ETOS real-time task. Each potential application would have to be examined to determine if the required rate of data transfer exceeds the ETOS real-time capability.

PROGRAMMING

Up to this point nothing has been said about the actual real-time programming. There are three types of programs which must be written in order to put a real-time application into practice; the foreground task(s), the background task previously referred to as the real-time monitor, and the user application program. The foreground and background tasks are coresident in the real-time job and will be discussed first.

In order to successfully operate as an extension of the ETOS monitor, the foreground task(s) must be as short as possible and should do nothing more than locate the interrupting device and service it. Since level 1 routines are re-entered on 100 usec. intervals it may be necessary to do only device recognition on level 1 and queue level 2 service to handle the remainder of the real-time task. Foreground tasks will be written in assembler language. There is no other good alternate choice. Much helpful information on writing real-time tasks is available from the ETOS Version 5A System Manager's Guide.

In writing the background task, the programmer does have a choice of languages. Since it is possible to use assembly language subroutines with all of the higher level languages, the channel input/output functions, system functions, and other special operations which cannot be programmed directly in the higher level languages can be executed in assembler language subroutines which are called from the main program. Assembler language subroutines can most easily be incorporated into FORTRAN II programs. In fact, FORTRAN II allows assembler language statements to be mixed with FORTRAN statements. For that reason, FORTRAN II would be a good language choice for writing the background task. For example, a background task could be written in FORTRAN II, which executes in field 0 and allocates field 1 data storage through the use of the COMMON statement. The foreground task could then be written in field 2 and share the field 1 data storage thereby providing effective communication between the foreground and background tasks. Use of the CHAIN statement would allow the programmer to make the background task as long as necessary. BASIC or FORTRAN IV could also be used although it is more difficult to incorporate assembler language subroutines into programs written in these languages. This is particularly true of FORTRAN IV.

In writing the user application programs, the programmer has the same language choices as when writing the real-time monitor. As was true with the background task, the channel input/output functions, the system functions, and other special operations can be handled with calls to subroutines written in assembler language.

One of the major differences between programming the real-time job and the user job is who might be expected to do the programming. The person programming the real-time task would have to have a basic, system level understanding of the ETOS monitor, experience in programming interrupt driven device handlers, and the ability to program in assembler language. The person programming the background task would require the same system level

knowledge and a working ability in assembler language programming plus the higher level language if he chooses to use one of them.

On the other hand, the person writing a user application program should not be required to have ETOS system level knowledge nor should he be required to have assembler language programming ability. In fact, anyone should be able to use existing real-time tasks by simply writing an application program in any of the ETOS languages. This implies that he should have at his disposal, subroutine calls which he can incorporate into his program in order to perform needed system level functions.

It is assumed that the organization responsible for the ETOS system operation would program the foreground and background tasks. In addition, this organization would also be expected to provide the assembler language subroutines needed to perform system level functions. These are not unreasonable assumptions since the personnel involved with the ETOS would be familiar with the DEC PDP-8/e system hardware, the ETOS system level functions, and would also have assembler language programming expertise.

IN CONCLUSION

After all this discussion, what can be said about the ETOS real-time capability? First of all, the ETOS can be successfully used to put into practice many of the potential real-time applications found in the industrial environment. Included in these potential applications is data collection utilizing either manual or automatic input stations, on demand data retrieval in human readable or machine readable format; and direct control of manufacturing machines, particularly those machines currently controlled by paper tape, magnetic tape, or punched card input. In addition, based on Omaha Works experience, the real-time aspect of the ETOS can be effectively used to incorporate the ETOS into distributed processing computer networks.

This is not to say that the ETOS has unlimited real-time capability. In setting up any real-time job on the ETOS it must be remembered that even though the foreground task will run in real time, the background task and the user application programs are timeshare jobs and as such are limited in the more traditional real-time background task capabilities. This is particularly true in the areas of intertask communication, execution priorities, and scheduling. Therefore, real-time applications which require frequent or time dependent interaction between the user program and the real-time job, or scheduling of user programs for subsequent execution may not be suitable for implementation on the ETOS.

As far as programming is concerned, anyone knowledgeable in BASIC, FORTRAN II, or FORTRAN IV should be able to successfully write a user application program provided he has at his disposal assembler language subroutines which perform the required system level functions. Writing these subroutines as well as the real-time foreground and background tasks would require more system level assembler language programming than would be the case with a conventional real-time system. However, this should be within the capability of the personnel responsible for the ETOS operation.

In conclusion, the ETOS represents a good, low-cost method for putting potential real-time applications into practice and, at the same time, providing general use timesharing.

REFERENCES

1. Digital Equipment Corporation, "OS/8 Handbook", 1974.

2. Digital Equipment Corporation, "PDP8/e, PDP8/m, and PDP8/f Small Computer Handbook", 1973.

3. QUODATA Corporation, "ETOS Version 5A System User's Guide", 1977.

4. QUODATA Corporation, "ETOS Version 5A System Manager's Guide", 1977.

SIMPLE MULTI-OS/8 BACKGROUND SHARING UNDER RTS-8

C.T. Teague, E.W. Yund, and J.W. Brodrick
Veterans Administration Medical Center
Martinez, California

ABSTRACT

A simple background sharing scheme was developed to allow three concurrent OS/8 users on a 32K Lab 8E running RTS-8 v.2 in the foreground. The system allows multiple residents in each of the three OS/8 task partitions. Foreground communication to user real time tasks from each background user is accomplished by trapping a "super IOT". Implementing this system in our environment required modification of the RTS-8 executive to allow dynamic removal of interrupt skip chain entries and to preserve EAE mode and step counter. A separate background task scheduler, currently not a part of the executive, was developed to insure sharing of background time. This scheduler is dynamically controlled by the highest priority OS/8 task in order to allow accurate real time control when required. Thus, both multi-user OS/8 and real time functions are supported by this system.

INTRODUCTION

RTS-8, a fixed priority multiprogamming system created by DEC for PDP-8 users, allows shared use of hardware resources by up to 63 "tasks" (1). Each task is a sequential program that executes under control of the RTS-8 executive. The RTS-8 executive maintains the state of each task, schedules task execution and supervises intertask communication. As a part of the RTS-8 system, DEC supplies tasks that control most DEC I/O devices, a monitor console routine (MCR), a nonresident task swapper (SWAPPER), and an OS/8 support task (OS8SUP). The OS8SUP task controls a virtual OS/8 machine that executes in the background, thus, providing access to full OS/8 facilities (powerful utilities, high level languages, etc.) to a single user. We have modified our RTS-8 system to support up to three OS/8 users who share background time. Unlike the MULTI8 system developed by Cardozo (2) our system retains the RTS-8 structure and can be implemented by any RTS-8 user with sufficient memory to allow at least 8K for each OS/8 user, a system device for each, and 8K memory for RTS-8 system and user tasks.

We are currently running our modified RTS-8 system on a 32K LAB-8/e with 3 RK05 disk drives, parallel LA-30 and 5 KL8 serial interfaces, as well as a custom data break tone generator. This system supports 3 concurrent OS/8 background tasks, one of which is nonresident and shares memory, exclusively, with an essentially identical OS/8 that uses a different console device. In addition, RK8E disk driver, 3 TTY drivers with access to the MCR, system clock, Diablo printer, laboratory peripherals, paper tape data input and OS/8 file listing facilities are supported. All control and utility tasks reside with the RTS-8 executive in fields 0 and 1. Each OS/8 background user executes in a separate 2 field partition above field 1 and is assigned a separate system device.

System memory usage was minimized by heavily overlaying low demand and noncritical I/O bound tasks. For example, the OS/8 file look-up, listing, and paper tape input utility tasks share a single 4 page memory partition and communicate with each other and other foreground activities through a resident 1 page message area. Similarly, all 3 TTY drivers share a 2 page partition; a resident page containing interrupt code is allocated to each. Memory requirements were further reduced by eliminating superfluous (to us) time, date, and task scheduling features of the MCR. Also, foreground access to OS/8 files was restricted to lookup and to reading or writing existing files, thereby eliminating cumbersome background/foreground interlock code. The system was configured to allow up to 32 tasks (23 tasks currently running) and requires 59 memory pages including 6 pages for each OS/8 support task.

MULTIPLE OS/8 IMPLEMENTATION

RTS-8 Executive Modification

RTS-8 provides no mechanism for preserving the EAE mode or step counter in the task state table. Preservation of EAE state is critical when multiple OS/8 or foreground EAE use is desired. EAE state may be simply preserved by extending the RTS-8 task state table and including code to determine EAE mode and store it with the step counter in a single word associated with each task. Rather than extend the task state table we chose to restrict all foreground tasks to using the same EAE mode with individual modes and step counters preserved for each OS/8 support task. This strategy required a separate table having a minimum length of 4 words, one entry for the foreground and one entry for each OS/8 support task. Thus far, this scheme has succeeded and offers the advantage of minimizing RTS-8 table space.

The EAE mode saver code is entered whenever task state is saved and EAE mode restorer code is entered whenever task state is restored (Listing 1). A single bit preserves the mode and the low order 5 bits hold the step counter in the table word. EAE mode is detected by executing a DPSZ instruction which skips if the EAE is in mode B and AC-MQ are 0. Both the AC and MQ registers must be saved before executing this code. The step counter is restored with the EAE in mode B then the mode is corrected to match that which was saved before task exectution was suspended. We placed this code and table in page 0 field 0 because space was available there; however, this code could be relocated elsewhere within the executive.

A further executive modification was required to implement the OS/8 swapping feature of our system. A means to dynamically remove entries in the RTS-8 interrupt skip chain was added so that terminals located in widely separated laboratories could be driven by either an RTS-8 TTY driver or an OS/8 support task. On entry these OS/8 tasks remove the TTY driver skip chain entry, suspend the TTY driver and insert their console handler entry into the skip chain. On exit they remove their own entry, release the TTY driver, send a message to the TTY driver and FREE the partition. The TTY driver checks to see if it is in the skip chain before processing the message and inserts itself if it is not.

The skip removal routine (Listing 2) is called in the same manner as any other executive request with a -SKPIOT argument. It skips down the skip chain looking for a matching I/O opcode. If one is found, the previous entry is updated to point to the next entry after the current one. Then the current entry is zeroed so that it may be reentered. If the end of the skip chain was modified, pointers to the end are also updated. Nothing is done if no corresponding opcode is found. Installation of this feature requires modification of the exective request table to add the entry and modification of SKPINS to place the pointer to the end of the skip chain on page 0 (Listing 3). This routine actually is part of the executive and should be incorporated within the executive. However, since the SWAPPER contains the FREE routine, also a part of the executive, and enough space was available, the skip removal routine was placed following the FREE executive code in the SWAPPER.

RTS-8 v.2 also does not allow nonresident tasks to be started from the interrupt level. This limitation was removed by inserting code in the executive interrupt processor which places a nonresident task in SWAPWT and runs the SWAPPER whenever an event flag is posted by an interrupt which makes a nonresident task runnable. This modification must be made to allow swappable TTY driver tasks; however it is not needed to support multiple OS/8 background tasks.

OS/8 Initialization

The OS/8 support 4ask supplied by DEC loads initialization code into its virtual field 0 which is executed one time to move the OS/8 system areas into virtual memory and establish a correspondence between OS/8 and RTS-8 device handlers. The user mode skip chain entry is also initialized at this time. This initialization code calls the OS/8 user service routine (USR) which performs disk I/O. USR disk usage causes a spurious interrupt to occur after OS/8 initialization is complete and interrupts are restored. This spurious interrupt is a problem if the RTS-8 disk driver has been called before it occurs because the disk driver modifies its interrupt handler so that the spurious interrupt will be interpreted as a fatal error and lock up the system (before it is really going). This problem is accentuated by the multiple initialization required by the 3 OS/8 system. Since the SWAPPER is the only task likely to call the disk driver before OS/8 initializes, we eliminated the spurious interrupt problem by placing the SWAPPER in USERWT initially and releasing it after OS/8 initialization was completed. Otherwise, OS/8 initialization is unmodified except that the highest priority OS/8 support task supervises the initialization of lower priority OS/8 tasks and inserts the user interrupt entry into the skip chain (Listing 4). All lower priority OS/8 tasks initially are placed in USERWT and are released when initialization is complete.

OS/8 Support Task Interrupt Handler

The OS/8 user interrupt handler entry point remains unchanged although code must be inserted to temporarily preserve the AC and Flags from the interrupt and dispatch to the appropriate OS/8 support task. Each task must then recover its AC and Flags (Listing 5).

OS/8 Background Timesharing

Because RTS-8 has a fixed priority preemptive scheduling scheme, the highest priority OS/8 support task (OS81) will prevent execution of lower priority OS/8 support tasks (OS82 & OS83) unless it is waiting for I/O. During largely compute bound operations, such as compilation, OS81 will use all available background time. In order to force OS81 to share time with other background tasks, we implemented a timeshare module as a separate task which could be controlled by OS81 because OS81 sometimes requires all background time to accurately control experiments running in real time.

The timeshare task allocates a time slice to each OS/8 task whenever they are runnable unless OS81 has set a flag indicating that it is not to be suspended. In this case, OS81 is not suspended and remaining background time is equally shared between OS82 and OS83 (Figure 1). In the event that an OS81 program fails to reset the time share flag, the OS81 support task resets the flag automatically whenever its keyboard monitor is loaded. Otherwise, the state of this flag is controlled by OS81 programs through the foreground/background communication module described below.

A time slice of 160 ms is used in our system because this time gives a good balance between OS/8 user response time and the time consumed by the timeshare task. Longer intervals reduce the amount of time consumed by the time share task but also prohibitively increase OS/8 response time; whereas, shorter intervals increase the time consumed by the timeshare task.

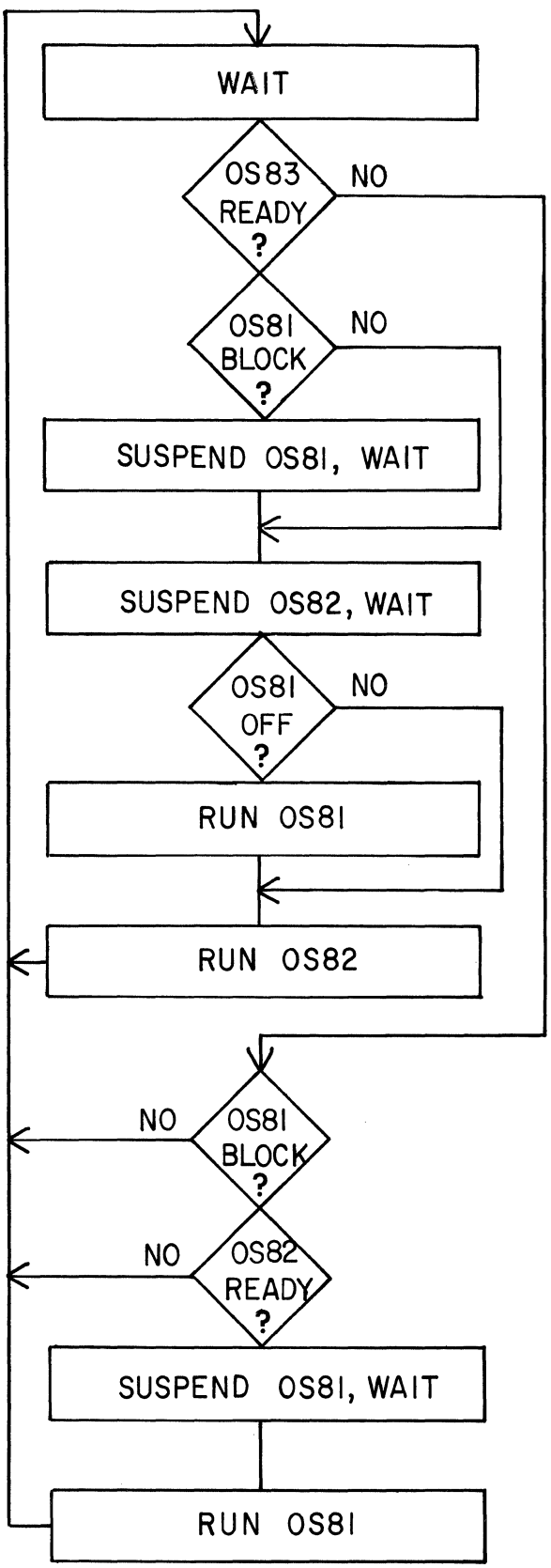


Figure 1. Triple OS/8 timeshare module.

Alteration of OS/8

To facilitate timeshare operation we were forced to make 3 alterations in OS/8 system programs. First, the KL8E TTY handler was modified so that instead of continually executing KRS instructions after receiving a CTRL S the sequence KSF; JMP .-1 is executed instead. This sequence causes the OS/8 support task to wait for I/O rather then needlessly executing. The second alteration was made to the FORTRAN II TTY handler and involved inserting the same sequence of instructions to cause it to wait for input. Finally, FORTRAN IV, without FPP, also was modified to use standard programmed I/O transfer by changing the interrupt driven I/O handlers in the FORTRAN IV run time system (FRTS) page 0. Since other illegal IOT instructions (ION, IOF, etc.) are interpreted as NOP by the OS/8 support task, no other changes need be made.

FOREGROUND/BACKGROUND COMMUNICATION

In order to take full advantage of both RTS-8 and OS/8 facilities an effective means to communicate with the foreground RTS-8 tasks from background OS/8 users is necessary. DEC suggests calling a special task from the background in a manner similar to calling any other OS/8 device handler. Although this scheme should be practical for a single background user, the system overhead in storage and special function decoding this scheme requires would be prohibitive in our multi-OS/8 system. Therefore, we decided to place a foreground communication module within each OS/8 support task. This arrangement allowed us to easily customize the module for the special requirements of each OS/8 user and also allows each OS/8 task to monitor its own foreground requests.

The foreground/background module is activated by executing a 6777 IOT instruction in the background with the offset into a table of callable foreground tasks in the accumulator. The attempt to execute the IOT is trapped by the time share hardware and control is passed to the foreground/background code. This code translates the table offset, forms a message to the foreground task and sends it (Listing 6). In all cases, except when a message is sent to the DEC-supplied clock task, the message consists of the address of the "real" message which resides in an OS/8 background field. Special functions, such as the dynamic time share control required by OS81, are called by passing a negative argument in the accumulator. Currently only one other special function is required; the swapping OS/8 users use this function to FREE their partition. This means of freeing the partition was selected because single character commands were too easy to type inadvertently. For example, when CTRL F was used to free the partition it was often typed instead of CTRL G while using OS/8 EDIT. Furthermore, virtually all control characters are used by our text processing system. Incidentally, access to the Diablo printer from the text format program is provided through this communication module and, thus, may be called by any of the 3 OS/8 users.

COMMENT

Most users are satisfied with the current system; however, a few problems have occurred. Although individual OS/8 system devices are unique, care

must be exercised to restrict indiscriminate file transfers between mass storage devices since all OS/8 background programs have access to all devices. We have implemented a general rule that one may read from but not write upon the mass storage device of another user. This rule should be suffficient in most situations. If it is necessary to transfer files to another user's device, we have found that telephone communication must precede the transfer. We also are among the RTS-8 users who have experienced a problem with the loss of the teleprinter flag in the OS/8 support task TTY handler. Like other users who have experienced this problem we have no solution other than to enter the appropriate wait value in the correct event flag location and post it from an MCR console terminal.

Listing 1. Routine and table to save EAE mode for OS/8's and foreground.

```
     /ROUTINE AND TABLE TO PRESERVE EAE MODE AND
     /STEP COUNTER ACROSS EXECUTIVE CALLS.
             *61             /PAGE 0, THERE IS ROOM
     MODTBL, 4000            /FOREGROUND ENTRY
             4000            /OS8 ENTRIES FOLLOW
             4000            /BIT 0=MODE; 0=B, 1=A
             4000            /BITS 7-11, STEP COUNT
     MOS8N,  -OS8+1          /- ONE LESS THAN OS81
     MDBASE, MODTBL          /TABLE BASE
     MODPTR, MODTBL          /POINTER INTO TABLE
     /SAVE MODE AND SC IN TABLE
     MODSAV, 0
             JMS MODTSK      /POINT INTO TABLE
             CAM             /CLEAR AC-MQ FOR TEST
             SCA             /STEP COUNT IN AC7-11
             DCA I MODPTR    /SAVED
             DPSZ            /SKIPS IN MODE B
             CLA STL RAR     /SET BIT 0 IF MODE A
             TAD I MODPTR    /COMBINE WITH SC
             DCA I MODPTR    /IN TABLE
             JMP I MODSAV    /DONE.
     /RESTORE MODE AND SC ACCORDING TO TABLE
     MODRES, 0
             JMS MODTSK      /POINT INTO TABLE
             SWAB            /MODE B TO RESTORE SC
             TAD I MODPTR
             ASC             /SC RESTORED
             TAD I MODPTR    /BACK TO TEST MODE
             SPA CLA
             SWBA            /MODE A
             JMP I MODRES    /DONE.
     /FORM POINTER INTO TABLE ACCORDING TO TASK
     MODTSK, 0
             TAD TASKX       /CURRENT TASK
             TAD MOS8N       /LESS OS8 OFFSET
             SPA             /+ IF BACKGROUNDER
             CLA             /ALL FOREGROUNDERS
             TAD MDBASE
             DCA MODPTR      /INTO POINTER
             JMP I MODTSK    /DONE.
```

Listing 2. Dynamic skip chain removal code, located in the SWAPPER.

```
     /SWAPPER CODE TO IMPLEMENT SKOUT
             *COMMAND
             INIT            /INITIALIZATION
             XFREE
             IFDEF   SKOUT   <
             JMP I   .+1     /SKOUT ENTRY
     SKLOC,  XSKOUT
             >
     /FOLLOWING AFTER BODY OF SWAPPER AND XFREE
             IFDEF   SKOUT   <
             LOC20=20
             FBSKCH= SKLOC+1
             FXRET=  SKLOC+2
             FCDIF=  SKLOC+3
             INTEND= SKLOC+4
     /TAKE A SKIP OUT OF THE SKIP CHAIN
     /CALL:  CAL
     /       SKOUT
     /       -SKPIOT         /MINUS THE IOT
     /
     XSKOUT, TAD I   LOC20   /GET NEGATIVE IOT
             DCA     T
             ISZ     LOC20
             TAD     FBSKCH  /START OF CHAIN
             DCA     LSTAD
             CDF 0           /IN FIELD 0
             TAD     SCDIF0  /TO START
     SKPLP,  DCA     LCDIF   /FIELD OF PREVIOUS
             IAC
             TAD I   LSTAD   /+ ADDRESS OF NEXT
             DCA     ADPTR   /POINTS AT SKIP
             ISZ     LSTAD   /POINT AT CDF CIF
             TAD I   LSTAD
             TAD     (-SKP   /SKP ENDS CHAIN
             SNA CLA
             JMP     SCDIF0  /SEARCHED IT ALL
             TAD I   LSTAD   /FIELD  INTO LINE
             DCA     NCDIF   /AND SAVED
     NCDIF,  HLT
             CIF 0           /OUR  FIELD *IOF*
             TAD I   ADPTR   /TEST THE INSTRUCTION
             TAD     T
             SZA CLA
             JMP     NXTSKP  /NOT IT
             AC7776          /FOUND
             TAD     ADPTR   /ADDRESS OF NEXT
             DCA     ADPTR
             TAD I   ADPTR
             DCA     ADSAV   /SAVED
             ISZ     ADPTR   /NOW THE FIELD
             TAD I   ADPTR
             DCA     NCDIF
             DCA I   ADPTR   /ZERO THIS ENTRY
             TAD I   INTEND  /CLEARED THE END?
             DCA     ENDFL   /SAVE A FLAG
     LCDIF,  HLT             /FIELD OF LAST
             CIF 0           /OUR FIELD, AGAIN
             TAD     NCDIF   /FIELD OF NEXT
             DCA I   LSTAD   /IN PREVIOUS
             TAD     ADSAV
             STA
             TAD     LSTAD
             DCA     LSTAD   /POINT TO ADDRESS
             TAD     ADSAV
             DCA I   LSTAD   /AND UPDATE
             TAD     ENDFL   /WAS THE END?
             SZA CLA
             JMP     SCDIF0  /IF NOT
             CDF 0           /OTHERWISE, RESET
             TAD     LCDIF   /SKPINS POINTERS
             DCA I   FCDIF   /FIELD
             TAD     LSTAD
             DCA     INTEND  /AND ADDRESS
     SCDIF0, CDF CIF 0
             JMP I   FXRET   /*ION* AND AWAY!
     NXTSKP, AC7776
             TAD     ADPTR
             DCA     LSTAD   /UPDATE ADDRESS
             TAD     NCDIF
             JMP     SKPLP   /TEST THE NEXT
```

```
LSTAD,   0
ADPTR,   0
ADSAV,   0
ENDFL,   0
         >
```

Listing 3. RTS-8 executive modifications to implement dynamic skip chain entry removal.

```
/MODIFICATION OF RTS-8 EXECUTIVE - PAGE 0
T,       0              /TEMPORARY
         IFDEF   SWAPPER <
         IFNZRO  COMMAN-.
         <__ERROR: CHANGE SWAPPER__>
         IFNDEF  SKOUT   <
/SWAPPER LOADS TWO ENTRY POINTS HERE
         *.+2
         >
/SPECIAL CODE FOR SKIP OUT ROUTINE
         IFDEF   SKOUT   <
/SKOUT ALSO LOADS ENTRY POINTS
         *.+4
FBSKCH,  BSKCHN  /BEGINNING OF SKIP CHAIN
FXRET,   EXRET   /NON-INTERRUPT RETURN
FCDIF,   INTCDF  /CDF CIF TO LAST SKIP
INTEND,  BSKCHN  /ADDRESS OF LAST SKIP
         >
         >

/MODIFICATION OF RTS-8 EXECUTIVE DISPATCH
/TABLE - PAGE 3
         XUNBARG
         IFDEF   SKOUT   </SKIP OUT ENTRY
         XSKOUT=COMMAN+2
         XSKOUT
         >

/MODIFICATION OF RTS-8 EXECUTIVE REQUEST
/SKPINS - PAGE 3
         JMP I   (TSTOP
         IFNDEF  SKOUT
INTEND,  BSKCHN          /NOW ON PAGE 0
         >
```

Listing 4. Multiple OS/8 initialization supervised by higest priority OS/8 support task. Actual initialization code is loaded into each OS/8 virtual field 0 by the OS/8 support task.

```
/MULTIPLE OS/8 INITIALIZATION
/RESIDES IN HIGHEST PRIORITY OS8 TASK.
         *57
         IFNZRO  NTASKS-OS8<
TSKCTR,                  / -  OS8 TASKS
ACT,     OS8-NTASKS      /AC TEMP
NTASKX,                  /LAST OS8 +1
LINKT,   NTASKS+1        /INTFLGS TEMP
         >       *166
TSWAP,                   /SWAPPER FOR INIT
AC,      SWAPPER         /OS/8 AC
STKBMX,                  /START AFTER INIT
PC,      STKBMN          /OS/8 PC
OSINIT,                  /INIT CODE ADDRESS
LINK,    INITOS          /OS/8 INTFLGS
/INITIALIZATION CODE
/OVERWRITTEN BY RING BUFFERS
         *4600
START,   CAL
         SKPINS          /LINK IN OS/8
         TTINT           /TELETYPE
OWBASE=  START
```

```
OWLEN=   20
IRBASE=  OWBASE+OWLEN
IRLEN=   10
IREND=   IRBASE+IRLEN
         CDF CIF OS8F0   /INITIALIZE OS8
         JMS I   OSINIT
         IFNZRO NTASKS-OS8<
/*** BEGINING OF SPECIAL INITIALIZATION
         CDF CIF OS82F0  /INITIALIZE OS82
         JMP I   OSINIT
         IFDEF OS83      <
         CDF CIF OS83F0  /INITIALIZE OS83
         JMP I   OSINIT
         >
UNBARL,  TAD     NTASKX  /UNBLOCK LOWER
         TAD     TSKCTR  /PRIORITY OS8
         JMS     UNBLKR  /BLOCKED UNTIL
         ISZ     TSKCTR  /INITIALIZED
         JMP     UNBARL
         >
/ALL OS8 TASKS INITIALIZED ***
/TO PREVENT SYSTEM LOCK-UP WHEN STARTED
         TAD     TSWAP
         JMS     UNBLKR
         JMP I   STKBMX  /LOAD KEYBOARD MONITOR
UNBLKR,  0               /UNBLOCK TASK
         CAL
         UNBARG
         USERWT
         JMP I   UNBLKR
```

Listing 5. Modified interrupt handler for triple OS/8 system. This code is located in the highest priority OS/8 support task.

```
/TIMESHARE INTERRUPT HANDLER
         IFNZRO  NTASKS-OS8<
TASKX=   32              /CONTAINS CURRENT TASK
         DCA     ACT     /AC TEMP
         GTF
         DCA     LINKT   /INTFLAGS TEMP
         TAD     TASKX   /CURRENT TASK
         TAD     (-OS8   /LESS THIS ONE
         IFDEF   OS83    </IF THERE ARE 3
         SNA
         JMP     TSINT1  /THIS IS IT
         CLL RAR         /AC HAD 1 OR MORE
         >
         SZA CLA
         IFDEF   OS83    <
         JMP I   (TSINT3 /MORE THAN 1
         >
         JMP I   (TSINT2
TSINT1,  TAD     ACT     /RECOVER AC
         DCA     AC      /FROM INTERRUPT
         TAD     LINKT   /AND FLAGS
         DCA     LINK
         >
         IFZERO  NTASKS-OS8<
         DCA     AC
         GTF
         DCA     LINK
         >
```

Listing 6. Foreground/background communication module. Similar in each OS/8 support task; this one is from OS81.

```
/COMMUNICATION MODULE FOR VARIOUS DEVICES
/CALL:   TAD     FUNCTION
/        6777            /TRAP IOT
```

```
/       ADDRESS         /MESSAGE ADDRESS
/       .               /RETURN, AC CLEAR
/
SNDMSG, TAD     AC      /FUNCTION
        SPA CLA
        JMP     ALT8SW  /FLOP TIMESHARE FLAG
        JMS     UCIF    /FIELD OF CALL
        ISZ     PC
        TAD I   PC      /MESSAGE ADDRESS
        DCA     SNDADD  /HERE
        TAD     AC      /OFFSET IN TABLE
        SNA
        JMP     CLKCAL  /CALL THE CLOCK
        TAD     SNDLST  /POINT AT TASK
        DCA     SNDDEV
        CDF CUR         /OUR FIELD
        TAD I   SNDDEV  /ACTUAL TASK
        DCA     SNDDEV
        TAD     UCIF+1  /FIELD OF MESSAGE
        DCA     SNDCDF  /FOR TASK
HERSND, CAL
        SENDW           /SEND AND WAIT
SNDDEV, 0
        MHERE           /OUR MESSAGE
ALTOUT, DCA     AC      /ZERO AC BEFORE
        JMP I   SNDRTN  /RETURNING
SNDRTN, XNOP
MHERE,  ZBLOCK  3       /RTS OVERHEAD
SNDCDF, 0
SNDADD, 0
SNDDUR, 0
SNDLST, SNDLST          /TABLE OF TASKS
        TONE            /TONE GENERATOR
        DGIO            /DIGITAL I/O
        VC80            /SCOPE CONTROL
        ATOD            /ANALOG TO DIGITAL
        DIAB            /DIABLO PRINTER
CLKCAL, JMS     UCIF    /HERE FORM CLOCK MSG
        TAD I   SNDADD
        DCA     SNDCDF
        ISZ SNDADD
        TAD I   SNDADD  /DURATION IN
        DCA     SNDDUR  /SYSTEM TICKS
        DCA     SNDADD  /NO HIGHER ORDER
        CDF CUR         / I.E. <= 40 SEC
        IAC
        DCA     SNDDEV  /CLOCK=1
        JMP     HERSND
ALT8SW, TAD     AC      /-1 TIMESHARE OK
        CMA             /OTHERWISE NOT
        DCA     ALTOS   /PAGE 0 FLAG
        JMP     ALTOUT  /RETURN IMMEDIATELY
```

REFERENCES

1. RTS-8 Users Manual. Digital Equipment Corporation. Maynard. Massachussetts. 1975.

2. Cardozo, E.L. : MULTI8 - A real time/timsharing system employing virtual memory techniques. Proceedings of the Digital Equipment Users Society 4:873, 1978.

MICROPROCESSOR BASED OCEAN BOTTOM SEISMOMETER

Robert D. Moore
Chin-Yen Huang
Geological Research Division
Scripps Institution of Oceanography
La Jolla, California

ABSTRACT

The prototype of a new ocean bottom seismometer design has been constructed at Scripps Institution of Oceanography and is operational. The design is based on a microcomputer using a 6100 microprocessor which emulates the DEC PDP8/E*. Instrument design, operating features and some software features are discussed.

Historically, the science of seismology has been the source of most of the data on which our present picture of the large-scale structure of the earth's interior is based. Studies of the propagation of elastic waves produced by earthquakes through the earth's interior has enabled much to be inferred about its structure. Recently, interest in the developing theory of plate tectonics has created a need for data on small-scale structures, such as plate boundaries. In order to obtain such data the seismometer must be sited at or very near the point of interest. Approximately 70% of the earth's surface, including many of the sites at which such small-scale structural data are needed, is covered by the oceans. Consequently, ocean bottom seismometer (OBS) systems capable of recording data at such sites have been developed at several geophysical laboratories in recent years.

The Scripps Institution of Oceanography has had an OBS program for several years using instruments designed here by Prothero[1]. A couple of years ago the need for additional instruments to support an expanding research program became evident. The availability of CMOS microprocessors whose low power consumption makes them usable in this application led to the decision to embark on a new design incorporating a microprocessor. The chief attraction of this approach is the resulting flexibility of the instrument. In order to realize as much of this potential as possible the processor chosen was the 6100 which emulates the DEC PDP8/E. This choice allows the use of a large-scale PDP8/E and the OS/8 operating system for program development.

Figure 1 is a photograph of the OBS system ready for launch. It consists of a pressure case (capsule) which contains the electronics, batteries, tape recorder and the triaxial 1 Hz seismometer. The fourth sensor, a hydrophone to detect pressure waves in the water, and the transducer for the acoustic system used to provide limited two-way communication between the OBS and a surface ship, are mounted outside the capsule and may be seen in Figure 1. The capsule is supported by a triangular anchor frame to which it is rigidly attached by two explosive bolts. The capsule with its attachments is positively buoyant, the weight of the anchor frame being sufficient to make the assembly negatively buoyant. At launch the assembly is lowered from the deck of the ship to a depth of about 200 feet and held there while its operational status is checked out using the acoustics. The line is then severed and the capsule free falls to the bottom. Deployment depths may be as great as 5 km (16,000 feet). Once on the bottom, the instrument automatically goes through a start-up procedure preparatory to taking data. The acoustic system is used to determine whether any problem develops, in which case the capsule is recalled and corrections are made. It is then assembled to a new anchor frame and re-launched. Once the capsule is operating satisfactorily, the ship leaves to launch other capsules or to do other oceanographic work. At the end of the deployment period, typically 10 to 30 days, the ship returns to the launch position, acoustic communication with the capsule is established and a release command is transmitted. This fires one of the two explosive bolts which is the normal release mode. If this fails, but the acoustic commands are being received by the capsule, a second command is sent to fire the other, back-up, bolt. If this also fails to produce a release, a pair of internal timers will attempt to fire the primary bolt first, then the back-up. If all of this fails, a $25,000 instrument is irretrievably lost. If either bolt fires, the capsule separates from the anchor frame and floats to the surface where it is recovered by the ship. During the capsule's ascent, the acoustic system is used to keep the ship positioned above it to ensure that it surfaces close to the ship.

Figure 2 shows the interior of the capsule. The object on top is the tape recorder. It is supported by the card cage which contains the CPU, memory and all peripheral interface cards. In the foreground is the acoustic package. Below these, the can housing the seismometer and its levelling system can be seen. The objects sticking out from the seismometer can are amplifiers, filters, etc.

Figure 3 is a block diagram of the OBS system. The basic design philosophy has been to make the system essentially a PDP8/E computer plus peripherals, all of which communicate via an extended 6100 bus, as

*DEC, PDP8/E and OS/8 are registered trade marks of the Digital Equipment Corporation, Maynard, MA.

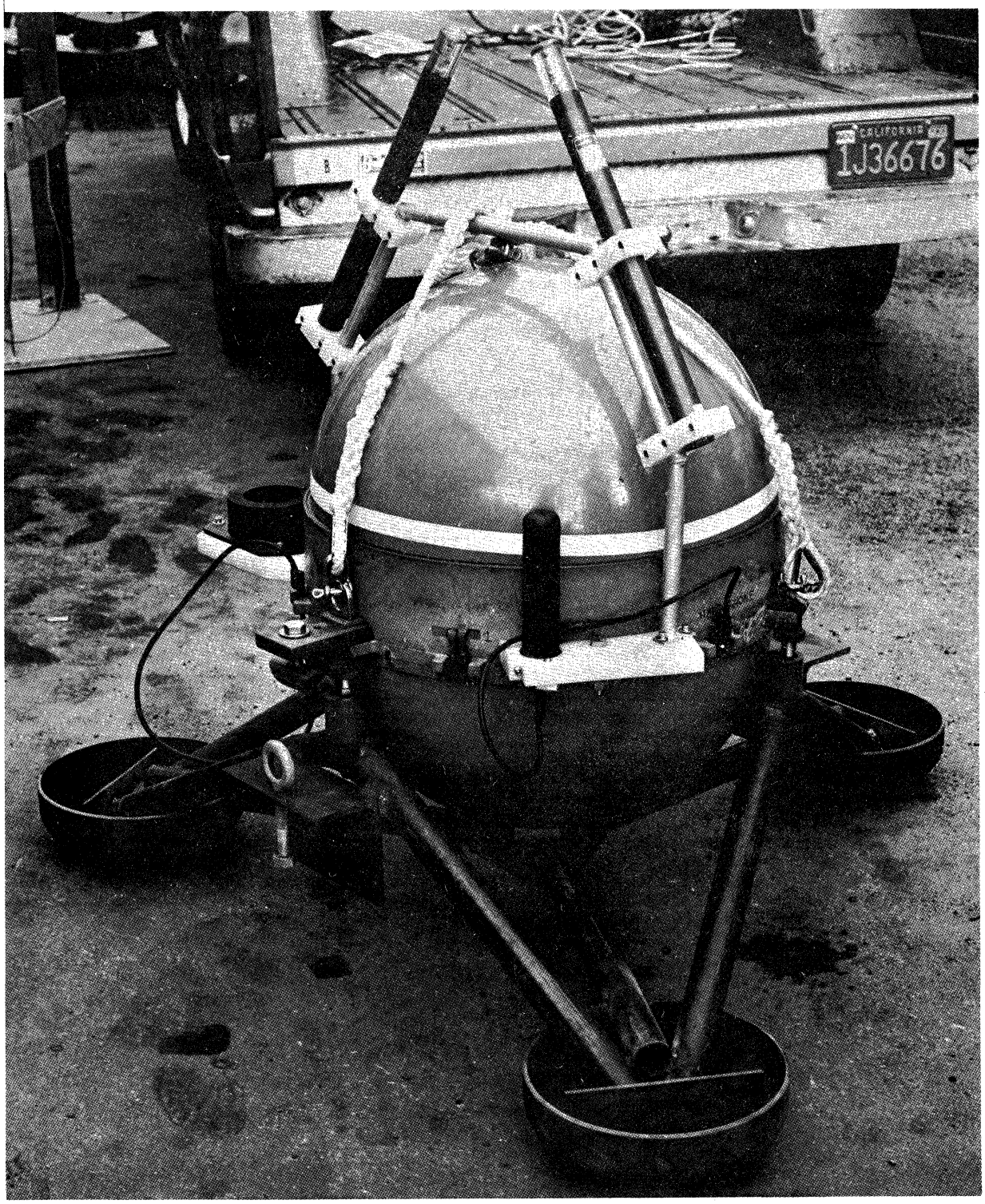


Figure 1. Assembled OBS ready for launch. The cylindrical object in the front is the hydrophone, the annular object on the left the transducer for the acoustic system.

opposed to a hardware-oriented system using the processor as a controller. Apart from timing restrictions, the system has no properties not determined by software. An exception to this is the release system which is autonomous and independent from the computer. This is to prevent either a failure to release, or a premature release due to computer hardware or software failure. Release system status information, i.e. acoustic command(s) received, timer release command(s) generated and whether the associated firing circuits have actuated, are available to the computer for acoustic transmission to the surface. All peripherals except the memory extender are implemented with the 6101 parallel interface element (PIE). All use programmed data transfers and status flags except the analog-to-digital converter which is serviced via a vectored interrupt in order to attain as

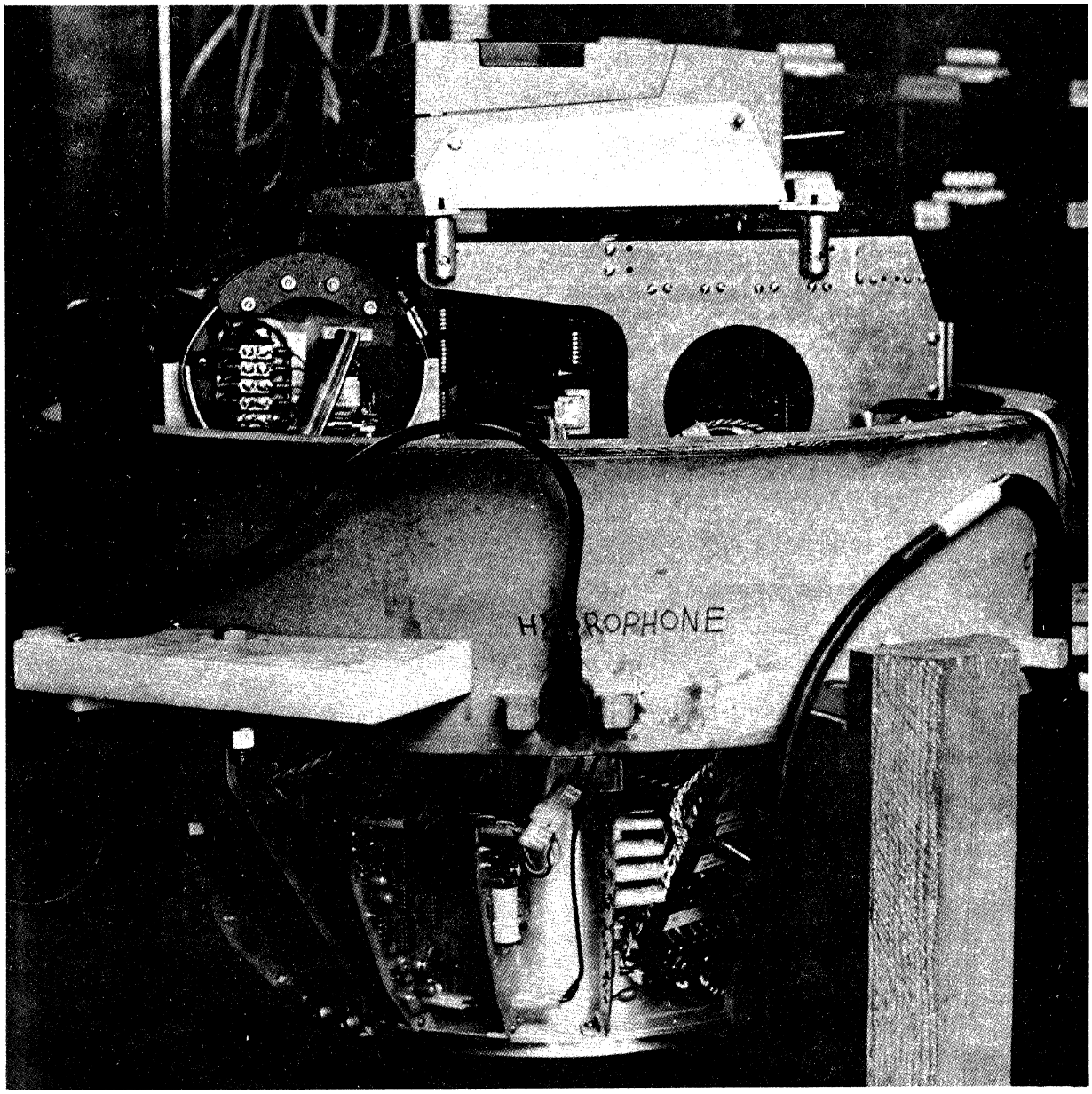


Figure 2. Interior of OBS capsule

precise data sample times as possible. The memory extender is implemented with CMOS logic and is the only departure (apart from the special instruction set pertaining to the peripherals) from PDP8/E software compatability. It only implements the DEC instructions pertaining to data fields, the instruction field being hardwired to zero. The large-scale PDP8/E system used here for program development was built around a 6100 and has sockets to accomodate capsule cards. The PIE device numbers used in the capsule were chosen so as not to conflict with those of the standard peripheral devices on the large system. Thus, all capsule cards may be operated in the large system for hardware and software debugging.

Data are recorded on magnetic tape. A commercial battery-powered 1/4" reel-to-reel tape recorder, modified for digital recording, is used. Four tracks are recorded serially using phase encoding. Each track is recorded independently with one data channel recorded on each track. The four tracks are played back independently, one at a time. The recording system operates at a fixed throughput rate of 1728 bits per second, or 144 12 bit words per second. Since each data channel is sampled at a rate of 128 samples per second, an overhead of 16 words per second is available for time, gain, etc. Four parallel-to-serial shift registers in the recorder formatter provide the four serial bit strings. The recorder is started and stopped under program control. While the recorder is running, the formatter must be supplied with four 12 bit words every 6.94 msec. This is accomplished by loading four consecutive words to the accumulator (AC) and then to the formatter via "load data" commands in response to a "load data" flag. Since the formatter is double buffered, the four words may be loaded any time during the 6.94 msec following the flag. The track on which a given word

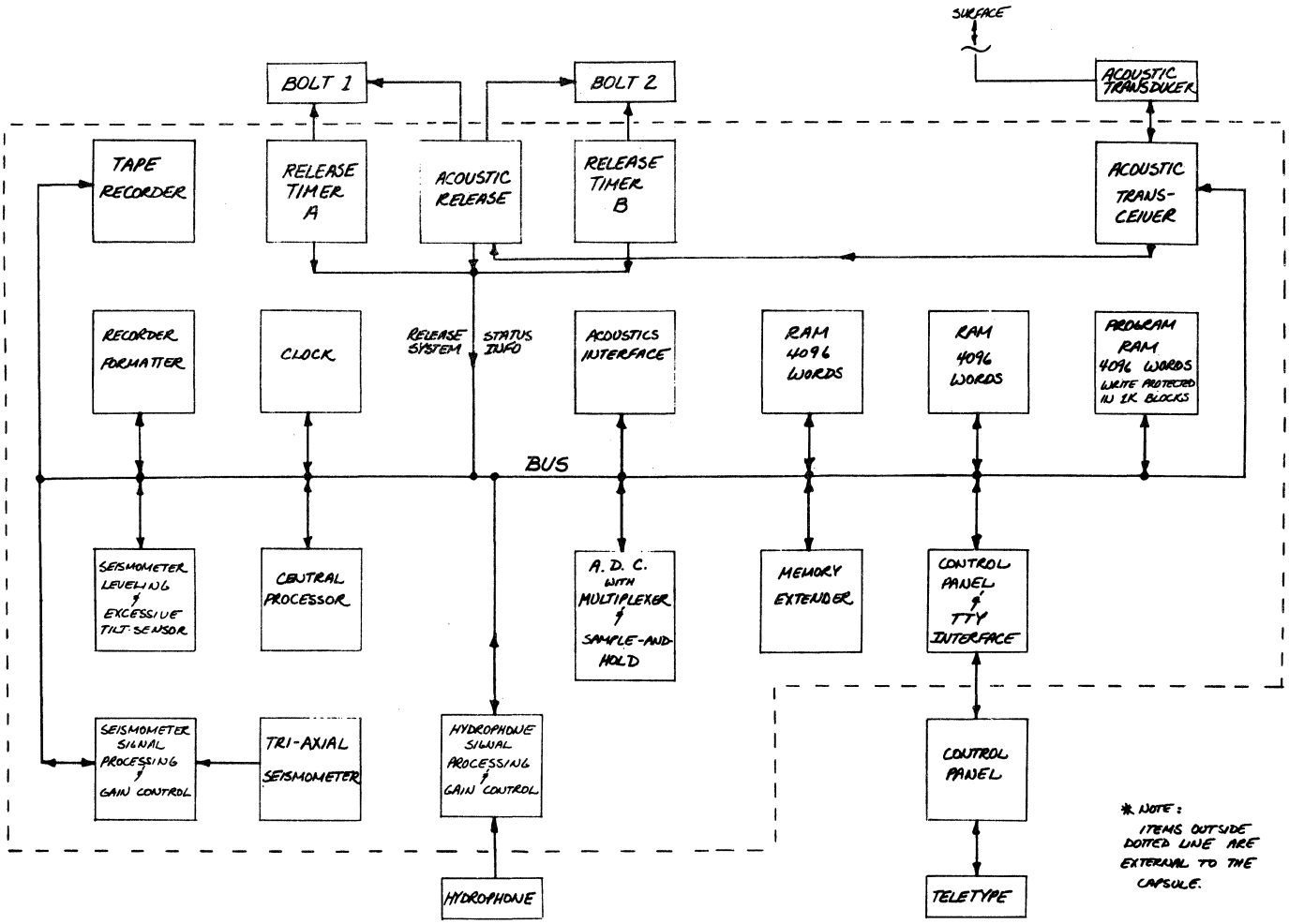


Figure 3. System block diagram.

is written depends on whether it is the first, second, etc. of a given group of four words. Thus CPU time required to service the recorder is minimal. The tape speed used is 15/16 IPS resulting in a recording bit density of about 1800 BPI. Since a five-inch reel holds about 1800 feet of 1/2 mil tape a total recording time of about 6.4 hours is available.

Since typical deployment times greatly exceed the total recording time, only data representing useful seismic events are recorded. Two modes of operation are used. In one mode, earthquakes constitute the signal source. In this case the OBS must by some means determine when incoming data represents a seismic event rather than ambient seismic noise. In the second mode, artificial seismic waves, usually produced by underwater explosions, constitute the signal source. In this mode the "shooting" is done on a schedule and the capsule is required to record on the same schedule. A precision real-time capsule clock is needed both to implement shooting schedules and to generate time words to be recorded on tape along with the data. The recorded time is used to compare seismic wave arrival times between capsules. For the data to be useful these times must be good to $\sim$ 0.1 sec. The real-time clock consists of a 24 bit counter which is incremented at 1 Hz.

The contents of the counter can be read by the CPU. Time is written on the tape once per second. The data are sampled at a rate of 128 Hz in response to interrupts. Since there are 128 samples between one second clock words on the tape "capsule time" can be resolved to better than .01 sec. At launch each capsule's 1 Hz clock is set as closely in phase with WWV as possible. The 1 Hz and 128 Hz clock frequencies are obtained by counting down on a 2.004480 MHz temperature compensated crystal oscillator which is also used to clock the CPU. All of this enables time comparisons between capsules to about 0.2 sec for a 30-day deployment at their present state of development.

The analog-to-digital converter (ADC) assembly consists of the converter, a sixteen channel multiplexer (MUX) and a sample and hold amplifier (SHA), all low-power CMOS devices. The SHA requires 100 µsec to settle after being switched to a new channel; a conversion takes 70 µsec. The assembly is interfaced to the CPU via a PIE. The coding for a single conversion is: load MUX address to AC, load AC to MUX address register, wait 100 µsec, execute

convert instruction, wait for ADC done flag, load ADC output to AC, store AC. Done as fast as possible, a sequence of four conversions takes about 1 msec. This PIE is also used to enable the CPU to transmit capsule status bits to the surface. Data are transmitted serially to the surface, 12 bits at a time, by loading a word from the AC to the interface which starts transmission. A flag is available that signals transmission complete. The computer word is converted by the interface to a series of pulses each of which causes the acoustic transmitter to "ping" once (a ping is a 12 KHz pulse 4.5 msec long). The code used is one ping for a zero, two, spaced 0.5 sec apart, for a one. The time between successive bits is exactly 4 sec as determined by the capsule clock. This enables capsule clock rate to be checked via the acoustic system.

On the surface, communication with the capsule is via a control panel providing all the standard PDP8/E control panel functions and a TTY interface. Data is transferred between the capsule and the control panel/TTY via a two-way serial port. The interconnect requires four wires--data in, data out, UART clock and ground. Since the capsule connector must be a deep submergence type and since highly reliable connectors of this type with large numbers of pins do not exist the small number of wires is very important. The actual control panel functions are performed by the control panel board in the capsule, the external control panel being an input/output device. All clocks used by the capsule board are generated from the UART clock which is inactive when the external control panel is disconnected. When the external control panel is disconnected, just before launch, the capsule control panel board goes into an inactive state so as to not interface with subsequent program execution.

Because of the horizontal seismometers, the triaxial seismometer assembly must be accurately leveled. The seismometer mounting system provides for automatic leveling upon execution of a "level" instruction. The maximum capsule tilt that can be accomodated is ± 15°. If the slope of the surface upon which the capsule comes to rest exceeds this, a "tilt" flag is set resulting in a corresponding acoustic transmission. In such a case the OBS must be recalled and re-launched.

The standard capsule memory compliment is 12K, normally fields 0, 1 and 2. Program memory space is hardware limited to field zero, fields 1 and 2 being used as a data buffer. Whether recording is initiated by a time schedule or by event triggering it is desirable to have data available for a time previous to that at which the recorder is started. At the present system throughput rate the 8K in fields 1 and 2 contains 14 sec of previous data at any time. The memory boards all use 1K RAM (6518) IC's and are designed so that they can be write disabled in 1K blocks by plugging an appropriately wired connector into the board. The present capsule operating software occupies 24 memory pages, using an average of about 80% of the available locations per page. These pages are all in the upper 3K, which are write disabled. This leaves the lower 1K, including page Ø, for program scratchpad.

The use of write disabled RAM rather than ROM or PROM for program storage is mainly to achieve maximum software flexibility. Our experience has shown that write disabled RAM is stable enough to result in reliable program execution over long periods of time. (Battery operation also helps considerably.) The system includes a "watchdog timer" which provides a means of program restart in the event of a problem caused by a soft program error. Hard program errors due to memory or other hardware failures are, in general, irrecoverable.

The useful bandwidth of the system is from about 0.1 Hz to 30 Hz. The upper frequency limit is determined by system throughput rate, the lower by the sensors. Each data channel is sampled at 128 Hz resulting in a Nyquist frequency of 64 Hz. The upper frequency limit for the data is set by the requirement that the low pass anti-alias filters be down at least 60 db at 64 Hz. Data are sampled in response to interrupts caused by a 128 Hz clock derived from the capsule clock. All four channels are sampled sequentially as rapidly as possible after the interrupt so that sample timing will be as accurate as possible. This results in a worst case timing error (last channel sampled) of less than 1 msec, and is negligible compared to the 0.1 sec timing accuracy required of the data.

Considerable processing is carried out on this data within the interrupt service loop. (All processing discussed below is carried out independently for all four data channels.) Short-term (1 sec) averages of the data, and the absolute value of the data (D and $|D|$, respectively), are generated by summing 128 consecutive samples. These averages are updated on every interrupt. Once every 128 interrupts a 128 Hz interrupt coincides with a 1 Hz clock increment. At these times long-term averages of D and $|D|$ are updated (L and $|L|$, respectively). A simple recursive algorithm namely:

$$Y_i = \frac{AY_{i-1} + BX_i}{A + B},$$

is used. Y represents L or $|L|$, X represents D or $|D|$. This algorithm closely approximates the response of a single-section RC low-pass filter with a time constant = (A + B)/B multiplied by the time between successive X_i values. The present software can handle time constants up to A = 1023, B = 1, i.e., 1024 sec, if the algorithm is updated once per second. Calculations are carried out using 36 bit two's compliment arithmetic. This register length is used to accomodate all possible values of Y_i and X_i without overflow.

The data channel gains are automatically adjusted to a value appropriate to the ambient seismic noise at the capsule location. The gains are adjusted during the start-up process and as needed during the entire deployment. In order to maximize dynamic range the gain should be such that ambient noise is just above the ADC threshold. The present scheme maintains $|L|$ at between 4 and 10 bits, full scale being ± 2048 bits.

$|L|$ is also used for event triggering. Once each second the ratio $|D|/|L|$ is calculated. If this ratio exceeds some threshold value T, where T > 1 for N successive trials, an event is said to have occurred and recording is started. T and N are

variable program parameters. This is a very minimal triggering algorithm. Ideally such an algorithm should be capable of triggering reliably on real events in the presence of ambient noise whose amplitude greatly exceeds that of the events. It is hoped that the real-time data processing provided by this system will contribute to the development of trigger algorithms approaching the ideal more closely. Event detection is enabled only if a corresponding program flag is set and is disabled during periods when the system is recording on a time schedule. The software includes the ability to switch back and forth between timed and event recording at preset times during a given deployment.

As discussed above, the automatic gain control program maintains $|L|$ between 4 and 10 bits. Since 2048 bits represents 5 V, this represents an average voltage between 10 mV and 24 mV. It is very possible for the electronics involved to have an offset of this order. This would invalidate the gain adjustment procedure and the trigger algorithm. Since the seismic signals have zero average value the long term average L will simply equal the offset of the channel in question. The data used to compute $|D|$ and hence $|L|$ are offset-corrected by subtracting L before processing them.

All of the operations described above are carried out during the first twelve 128 Hz interrupts following each 1 Hz clock increment. These consist of updating L and $|L|$ and gain adjustment for all four channels. The trigger algorithm is executed during the $|L|$ loop for the channel used to trigger. (At present only one channel, the vertical seismometer, is used for triggering.) The average execution time for the various paths is 3.5 msec. The interrupt service routine also stores the offset corrected data and the additional overhead words into the 8K RAM buffer according to the tape format. The buffer is loaded in a circular fashion, old data being continuously overwritten by new. A pointer register is maintained to locate the moving boundary between old and new data. This pointer is used to determine the buffer addresses read during recording. Since the data are input to and removed from the buffer at equal average rates buffer manipulation is simple.

For most of the time during a deployment the capsule is not recording data. During this time a background program which consists mainly of a long skip chain during which the many capsule status flags are checked is executed. The frequency of the CPU clock can be switched by the program between 2 MHz and 250 KHz. Since computer current consumption at the low frequency is about 50% of that at the high frequency and since more than half the time is spent in the background program, considerable power is saved by running the background program in slow clock. High clock must be used during the interrupt service routine to meet time requirements. The capsule clock is also checked in the background program for times at which the recording mode is to be switched between event and timed, and for scheduled recording times when in the latter mode.

During recording periods a program flag in the interrupt service routine is set. When this flag is set the service routine returns to a different background routine which outputs data to the tape recorder and performs certain related bookkeeping tasks in addition to checking the recorder load data flag. Since the load data flags are 6.9 msec apart and the interrupt service routine only requires 3.5 msec there is time to service both the interrupts and the recorder flag. The CPU clock is maintained at 2 MHz as long as the capsule is recording. When the recorder is stopped, the program flag is cleared and operation reverts to the normal background routine.

The present state of this development is that a prototype capsule is complete and working along with a first-attempt software package. This software is sufficient to duplicate all the features of existing hard-wired systems with some improvements. As time goes on our goal is, through a continuing software development program, to achieve a level of performance beyond that of existing systems. A particular case would be the development of trigger algorithms approaching the ideal more closely than the present one. It is also hoped (perhaps forlornly) that, for a few years at least, instrument performance can be steadily improved by software development, the hardware configuration remaining essentially fixed.

Acknowledgments. During this work, the energetic and dedicated efforts of Dave Berliner, Fred Boatright, Karen Chass and Don Sullivan have been invaluable. In the course of the development of this instrument, many fruitful discussions were held with W. A. Prothero of U.C. Santa Barbara, who is presently engaged in a similar development project, several of the design features of this instrument being originally due to him. This research has been supported principally by the Office of Naval Research under contract USN N00014-75-C-0152 with some assistance from the National Science Foundation under grants OCE76-22680 and EAR76-22493.

References

1. Prothero, W. A., A free fall seismic capsule for seismicity and refraction work, Offshore Technology Conference, Paper #2440, 1976.

CMOS DATA ACQUISITION SYSTEM

FOR OFFSHORE OIL RIGS

by

John M. Kracik
Oceanic Engineering Division
Interstate Electronics Corporation
Anaheim, California

ABSTRACT

This system uses various transducers to monitor strain, temperature, pressure and acceleration on the marine riser pipe through which offshore oil wells are drilled. A data link is used to communicate between the remote site (sea floor) and a minicomputer on the drilling rig. This data link may use hard wire, RF, inductive or acoustical transmission. The 6100 based system is capable of loading software via either the data link or a resident ROM memory. The system is installed in a 5-inch ID pressure housing with power supplied from the surface system.

INTRODUCTION

In offshore drilling operations a marine riser is used to link the wellhead at the sea floor with the drilling rig. Both the drill string and drilling fluid pass through this riser. A structural fault in the riser can result in a catastrophic failure; therefore, to safeguard against such a failure, a remote data acquisition system is employed to measure riser stress and other critical parameters at the wellhead. Operating water depths range from 100 feet to as much as 5,000 feet, and in this instance, the wellhead was at 5,000 feet. (See Figure 1.)

PDP-8 COMPUTER USED IN DRILLING OPERATIONS

The objective of the system in this offshore oil drilling application was to gather large amounts of data at the 5,000-foot deep remote site, process the data through various algorithms, and then transmit the processed data to a central PDP-8 computer on the surface for further processing and storage. Additionally, this CMOS DATA ACQUISITION SYSTEM was required to operate continuously, 24 hours day.

OPERATIONAL DEPTHS LIMIT SIZE AND POWER

The constraints of equipment size and power supply at the remote site made the problem a difficullt one. Because the system was required to operate at a water depth of 5,000 feet, it was necessary to package it into a small pressure vessel capable of withstanding considerable outside pressure. The small size of the package and the 24-hour-operation requirement also effectively ruled out self-contained batteries as the primary power source, therefore power had to be supplied by the PDP-8. And too, because of the great operational depth, the size and weight of the cable had to be minimized.

It was also required that the system be capable of (1) responding to commands from the central PDP-8 computer for synchronizing data gathered at the remote site and at the PDP-8 site, (2) changing system parameters, and (3) executing calibration functions. Additionally, it was desired to change the total operating characteristic of the remote site from the PDP-8 computer (e.g., loading new acquisition programs).

MICROPROCESSOR LOGICAL CHOICE

In meeting the design constraints, the selection of a microprocessor was a logical choice. The microprocessor is capable of controlling the acquisition of sensor data, executing various algorithms associated with the data being acquired for transmission, and responding to various commands from the central PDP-8 computer. A CMOS microprocessor was chosen primarily because it is a low power consumer. Consequently, cable conductor size was also reduced because current required for CMOS operation is minimal. (See Figures 2 and 3.)

To improve reliability of the system over the desired temperature range of 0°C to 85°C, the system supply voltage was increased from 5 Vdc to 6 Vdc and this appreciably reduced the propagation delay due to capacitive loading of various processor and memory signals. To further enhance the system, new state-of-the art CMOS buffers (HD-6495) were implemented.

The 6100 series microprocessor was chosen because it met the above-stated conditions and is also capable of executing the PDP-8 computer instruction set. Being able to compile new programs at the PDP-8 and down-load them to the remote site was an additional benefit.

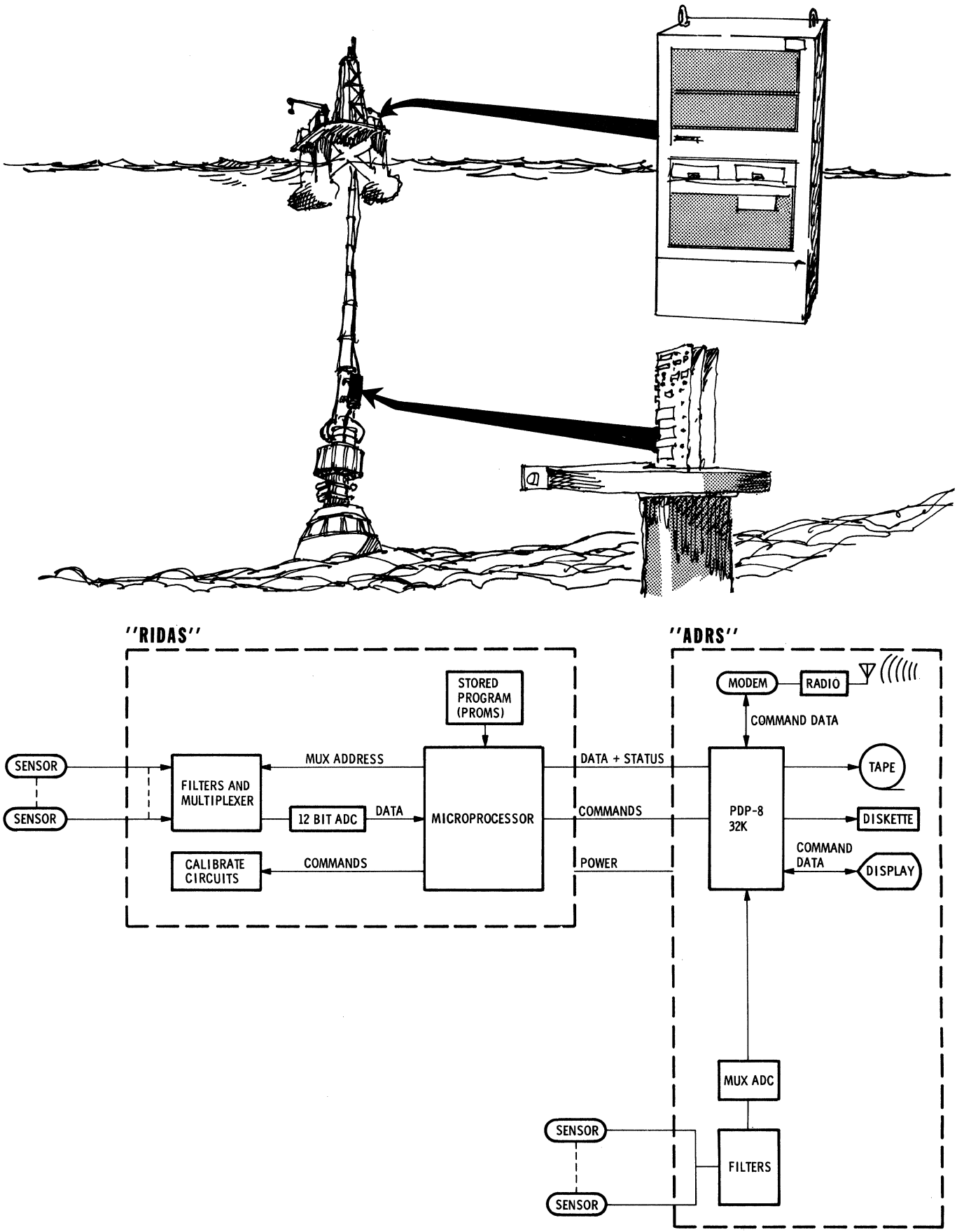


Figure 1. Data Acquisition System

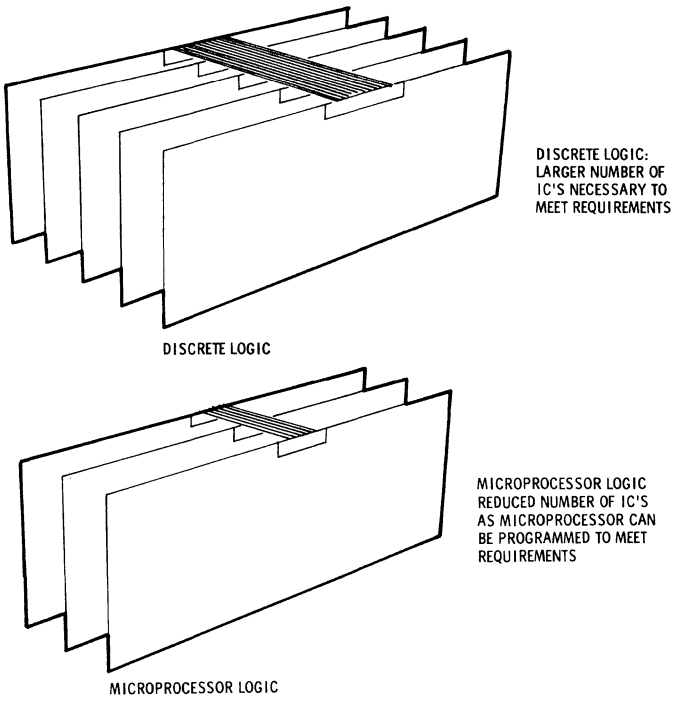
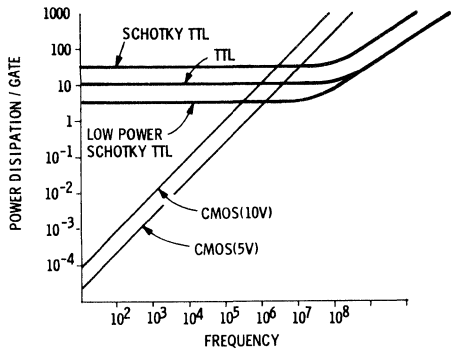


Figure 2. Comparison of Discrete and Microprocessor Logic Boards



	TTL	74L	DTL	74L'S	CMOS(5V)
PROPAGATION DELAY	10ns	33ns	30ns	10ns	35ns
TOGGLE FREQUENCY	35MHz	3MHz	5MHz	40MHz	5MHz
QUIESCENT POWER	10mW	1mW	8.5mW	2mW	10nW
NOISE IMMUNITY	1V	1V	1V	0.8V	2V
FAN OUT	10	10	8	20	50*

*AS DETERMINED BY ALLOWABLE PROPAGATION DELAY

Figure 3. Properties of CMOS Logic Versus other Families

SYSTEM OPERATION

Operationally, the basic function of the data acquisition system was to acquire analog data from various sensors located at the remote site. To acquire data from these sensors, a knowledge of sensor output is necessary to determine critical parameter sample rates and low-pass filter characteristics. Sensor output can be characterized into two subgroupings--low-level differential and high-level single ended--both exhibiting a d-c to 0.25 Hz frequency response. Low-level sensor output can be subgrouped into ±100 mV full scale and ±350 mV full scale. High-level sensor output ranges between ±5 volts and ±10 volts full scale. A sample rate of 2 Hz was necessary to meet accuracy requirements and to prevent aliasing of data over the bandwidth of interest from d-c to about 0.25 Hz. Under these conditions 4-pole low-pass filters with a corner frequency of 0.25 Hz would be necessary, dictating the use of active filters. But physical component size and power consumption prohibited this approach, and it was decided to use passive 2-pole low-pass filters at 8 Hz with a sample rate of 32 Hz. Once the data is acquired at 32 Hz, it is digitally filtered to prevent aliasing and then subsampled at a 2-Hz rate prior to transmission to the PDP-8. The digital filter required maximum processor speed during data acquisition.

The use of read only memory (ROM) for program execution reduces board density; however, this causes the use of indirect memory reference instructions which reduces processor execution speed by as much as 30 percent. Since some RAM is needed for storage of data, an all-RAM system was implemented, but this meant that the program would always have to be loaded upon start-up. This was accomplished by downloading from the surface PDP-8; however, data transmission errors may occur or the downlink may fail entirely. Thus, to ensure proper program loading, a backup method was needed. The solution was to use a Program Injection Module which allows the remote system to be loaded with correct software in the event of a faulty program load from the PDP-8. To minimize processor design, the Program Injection Module is treated as an I/O device, and to minimize power consumption, it is operated in a power-up mode only during program loading, which takes approximately 2 to 3 milliseconds.

MODULARIZED HARDWARE PERMITS EXPANSION

The hardware was modularized for expansion with the minimum configuration consisting of two modules -- a Basic Control Module and a Program Injection Module (optional). Additional Analog Interface Modules can be utilized to expand the system to the required number of channels. The Control Module contains the microprocessor and support circuitry, an interval timer, two serial asynchronous interfaces, debug ROM (1024 x 12), bootstrap loader PROMS (256 x 12), 2K of RAM memory (2048 x 12), and four parallel interface circuits. The Program Injection Module contains 2K words of PROM (2048 x 12) which is power strobed during access. The Low-Level Analog-to-Digital Converter Module contains an 8-channel differential analog multiplexer, a 12-bit analog-to-digital converter (all CMOS), eight low-pass 2-pole filters, and signal exitation and calibration circuitry. The High-Level Analog-to-Digital Converter Module contains a 32-channel, single-ended analog multiplexer, a 12-bit analog-to-digital converter (all CMOS), thirty-two 2-pole filters, and three hardwired voltage levels used for calibration. (See Figure 4.)

The electronics modules were designed so that they would fit into a 5-inch ID, 21-inch-long cylinder. The modules are interconnected by a 40-conductor flat cable.

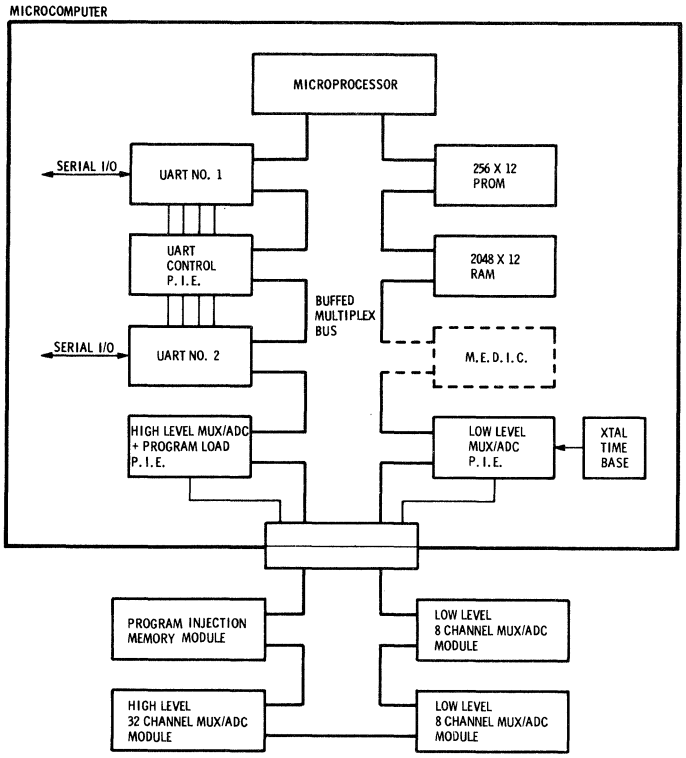


Figure 4. Data Acquisition System Block Diagram

Control Module

The control module contains two separate programs stored in read-only memory (ROM): a 256-word program and 1024-word program. The 1024-word program is stored in a 1024 x 12 masked ROM and contains an octal debugger routine; the 256-word program is stored in three 256 x 4 PROMS and contains a system test program and two loader programs. The selection of ROM programs is accomplished via a switch on the control module. The octal debugger is used as a system debug tool to isolate component failures on the control module and I/O modules. The program allows an operator to enter short test programs and verify the results via a terminal connected to the control module utilizing a serial I/O port. The test/loader program verifies operational status of the system by first checking processor operation, memory operation, crystal clock operation and I/O operation. It notifies the central PDP-8 system of its status by transmitting various control codes via a serial I/O port. Once proper operation has been established, the program then executes a loader routine. The loader routine first establishes if a binary load is to occur by monitoring the serial I/O port for proper binary load sequence. In the event of an improper binary load sequence, it waits for approximately 1 minute at which time it executes a program injection routine. Both binary loader and program injection routines check for valid checksum. If a checksum failure occurs, a status message is sent to the PDP-8 system and the program remains in loader mode.

The control module contains 2048 words of random access read/write memory (RAM) where program execution and temporary data storage occur. The use of RAM memory for program execution was used so that maximum processor speed would occur.

The control module also contains three parallel interface elements (PIE). The PIEs were chosen because of their vector interrupt facility and simplification of I/O circuitry. One PIE is used to control two universal asynchronous receiver-transmitters (UART). In addition, two flag outputs are used to control whether even parity or no parity is implemented on the UARTs. One of the UARTs is used as a primary interface to the PDP-8 computer; the second is used as an auxiliary input.

The second PIE is used to control two low-level Mux/ADC modules. Its two output control signals (WRITE STROBES) are used to load multiplex address information and to start analog-to-digital conversion (ADC). The two input port control signals (READ STROBES) are used to read ADC information, and two sense lines are used to detect end of conversion (EOC) status. Three flag outputs are used to select the low-level calibration function. The fourth flag is used to force the multiplexers into channel zero selection.

The third PIE is used to control the high-level Mux/ADC module and the program injection module. One of the output control signals is used to select multiplexer address information and to initiate conversion. The second output control signal is used to load the program injection module address register. One of the input control signals is used to read ADC information into the processor; and one sense line is used to detect EOC status. The second input control line is used to read data (program instructions) from the program injection module. The remaining sense lines are used to detect power low and interval timing interrupts from the XTAL oscillator.

All control signals are buffered before they leave the module via a 40-conductor cable.

Low-Level Multiplexer ADC Module - The low-level multiplexer analog-to-digital converter contains an 8-channel low-level multiplexer and 12-bit analog-to-digital converter module, eight precision voltage regulators used for sensor excitation, sixteen 2-pole low-pass filters, and fifteen calibration relays. Two sets of low-pass filters are used prior to the connection of sensor output to the differential multiplexer. The differential multiplexer ADC contains an instrumentation amplifier with appropriate gain to amplify sensor output and various circuitry to allow for amplifier settling before an analog-to-digital conversion is initiated.

High-Level Multiplexer ADC Module - The high-level multiplexer ADC module contains a 64-channel multiplexer (51 channels are used), a 12-bit ADC module, and forty-eight 2-pole low-pass filters. Three of the multiplexer channels are hardwired to precision voltage sources used for calibration.

The multiplexer address is loaded from the buffered processor data during a write instruction from the appropriate PIE. Once the address has been latched, the Mux/ADC module waits for its instrumentation amplifier to settle at which time it samples the analog voltage in its sample-and-hold-circuit and then starts a conversion. The end of conversion signal (EOC) is connected to an appropriate PIE sense line. When the EOC signal occurs, the processor executes a service routine which places the A/D data on the processor data lines.

Program Inspection Module

The program injection module contains 2048 words of bipolar PROM memory, a 12-bit CMOS address latch, 12 tri-state CMOS buffers, and control logic used to power PROMS during access. The address latch is loaded via the buffered processor data lines when a write instruction is executed from the appropriate PIE. The write instruction is also used to power the PROM memory. The data (instructions) is loaded on the processor data lines during a read instruction. The falling edge of the read instruction are used to power down the PROMS.

CONCLUSION:

In this offshore drilling application the constraints of physical size, power consumption, and size and weight of the cable connecting the surface equipment and subsurface electronics were met by the use of a CMOS microprocessor. The CMOS microprocessor reduced the amount of logic needed at the remote site and also reduced the amount of power needed for its operation. Typically, the control module consumes only 50 mA of current, hence the size of the conductor cable could be minimized. Utilization of a serial data transmission format reduced the number of conductors between the surface and remote site. Reduction of both conductor size and the number of conductors allowed the use of smaller lighter cables.

The use of a microprocessor reduced the amount of logic necessary at the remote site, hence an increase in reliability could be realized. Furthermore, the remote site now could be reprogrammed to implement any data acquisition function desired.

PDP-8/E DEVELOPMENT SYSTEM FOR BIT-SLICE MICROPROCESSORS

Douglas F. Gluntz
Harris Government Electronics System Division
Melbourne, Florida

ABSTRACT

Microcode generation consumes the largest share of the development dollar in most bit-slice microprocessor based design budgets. Microcode requires many hours to develop, therefore expensive, and is usually left to the skill of the designer. Costs can be significantly reduced by the use of hardware and software design aids. A PDP-8/E is used for source entry, linkage to a functional simulator (CDL) and microcode loading to a memory simulator. The aids discussed in this article were developed and used on a project involving an AMD 2901 bit-slice microprocessor, however, the techniques described can be applied to any microprocessor.

INTRODUCTION

The primary objective of most microprocessor design projects is to provide a reliable device to perform a specific function. Development of the firmware component is an important (sometimes predominant) cost factor, well worth studying.

Firmware engineering is a collection of rapidly evolving concepts which can be applied to firmware development activities. "Firmware engineering" may be defined as the planning designing, construction and management of microprocessor firmware, including programming methodology, reliability, performance and design evaluation, and program development aids. The field is less than ten years old and moving rapidly.

The firmware designer, unlike his hardware counterpart, is pretty much on his own. He usually can't call manufacturers for application information, data sheets, or technical consultation that will help solve his particular problem. Since firmware for bit slice microprocessors is unique for each application, assemblers, compilers, and/or interpreters are non-existent. Cost effective firmware design is currently a hit and miss affair and highly dependent on the skill of the firmware designer. Until now, there was no set format to follow that would ensure cost effective design of firmware.

The method proposed herein, consists of the following steps:

1. Firmware design
2. Computer modeling and simulation
3. Firmware/hardware integration and debug

Good firmware design should obey the same basic rules as good top down structured programming utilized in normal software development. As such, this paper will only briefly dwell on this area.

Computer modeling and simulation provides a means to simulate the hardware and firmware. Although this paper will not concern itself with hardware design, it is an important factor in the timely generation of working microcode. Computer modeling and simulation can provide the hardware designer with essential information to allow for timely development of the hardware to support firmware debug.

At some point in a project, the hardware must be married to the firmware. As most designers have discovered, the honeymoon is of extremely short duration. Divorce is simply not allowed and an amicable co - existence must prevail. This period is known as debug time. It is often faced with great reluctance and despair. At the end is a trail of frayed nerves, bloodshot eyes, lost friendships and a great sigh of relief. This activity can be improved upon significantly with the use of the proper design aids. These aids assist both the hardware and firmware designer and provide meaningful data upon which design change decisions can be based.

FIRMWARE DESIGN

Assuming that you have used top-down design, the complete firmware requirements have now been broken down into the sub-blocks that can be coded easily. What follows next is really an extension of the activity that was already performed in order to get to this point; namely, break the requirements

smaller pieces. However, each piece will have a personality all of its own. The three steps required to get to the end of the design process are as follows:

1. Flow Chart
2. Sequence Chart
3. Microcode Generation

The flow chart for a microprocessor system is really no different from any other flow chart. It is a pictorial representation of the logical sequence of events necessary to perform a required operation. Each sub - block should have its own flow chart. It should not contain detailed step-by-step or instruction-by-instruction detail. Remember it is showing flow, not minute detail. It should be noted, that after debug is complete, this piece of documentation will undergo only minor modification, if any.

Now that you know the overall events necessary to perform a certain task, you are now prepared to proceed to the next level of detail - the sequence chart. The sequence chart describes the sequential operation of the microprocessor. A micro-operation is a single operation and it takes several micro-operations to perform a useful function. Each sequence represents a micro-operation or simply an operation performed by the microprocessor during one clock cycle. Unlike the flow chart, this piece of documentation can - and usually does undergo several modifications before debug is complete.

Now the final, and easy task of generating the microcode. You simply plug in the necessary digits on a coding sheet to perform the micro-operation defined by the sequence chart. It is during this operation that the memory locations will be determined and can be added to the sequence chart.

Before any code generation can begin, a coding format based on the standard eighty column coding sheet is selected. It is noteworthy to point out that the coding format should provide for incorporation of comment statements for each line of code. Full lines of comments should also be allowed. Each column is also assigned a default value to relieve the programmer from inserting a numeric character in each column for all lines of code. These values are also selected such that whenever possible, the default value generates a logic "1" in the PROM if the selected PROMs are received with all locations containing ones and required programming to obtain zeroes.

Since, the final depository of the firmware you have designed is usually stored in solid state memories (ROMS, PROMS, EAROMS, etc.) every attempt should be made to design some multiple of memory word length - typically 1K. During the early design stages when the control word is being developed it is usually a good practice to provide spare bits. Today, the cost is less than $20 for a IKX4 PROM. It is a lot cheaper to provide more bits/word in the early stages of design than to go back and add a IKX4 PROM later on. You will almost invariably need a longer word at the conclusion of a project than you predicted at the beginning.

The size of the firmware can grow in two directions - horizontally and vertically. A horizontal expansion is implemented simply by adding more control bits to each and every word for controlling additional hardware elements. Horizontal expansion allows the microprocessor to perform more parallel operations during each micro-cycle. It suffers from low utilization of the control bits and requires more memory. A vertical expansion requires an increase in the number of words, and can increase the overall capability of the microprocessor. The drawback here is that if the number of words are exceeded for a memory device (e.g. 1K), it will be necessary to double the size of the memory to gain the additional words. Each system has to be analyzed as to the best implementation to meet program requirements.

COMPUTER MODELING AND SIMULATION

There are many hardware simulators commercially available on the market today. Regardless of the simulator selected, it should perform as many of the following functions as possible:

1. Simulation to lowest level of interest (gate, flip-flop, counter, register, etc.)
2. Timing analysis
3. Fault detection
4. Fault isolation
5. Test pattern generation
6. Fanout loading analysis
7. Variable delay element
8. Technology independent
9. Failure analysis
10. Accept microcode as input

The philosophy here is to accurately simulate the hardware by computer modeling to confirm the hardware design and at the appropriate time execute firmware instructions. This allows resolving many problems during the early stages of design instead of waiting until the integration phase when the fix will be more difficult and costly. Effective computer simulation programs are costly to generate, therefore, the tendancy to roll-your-own should be suppressed.

Firmware microassemblers are usually non - existent for bit-slice microprocessors. Although some manufacturers sell firmware prototype and/or development systems, they are costly and may not fit any specific application. Most firmware designers have a computer system available for use which will probably be adequate for the task at hand. The computer system should be able to receive and store the required microcode and comments. Obviously, it is highly desirable that this computer system and the

computer system providing hardware simulation discussed above be one and the same.

An interactive full page editor with CRT display is probably the most prized possession of any firmware designer. No matter how competent the firmware designer or how diligently he aspires to perform error - free, editing will consume a large portion of his time and effort. The ability to see large blocks of microcode combined with the capability to quickly insert and observe a change will significantly reduce edit time and increase firmware development productivity. Something less than an interactive full page editor with CRT display may suffice, but productivity will suffer.

The simulator selected was Computer Design Language (CDL). CDL is a non-procedural language for describing the functional organization, algorithms, and both parallel and sequential operations of a digital system. CDL was initially developed by Dr. Yaohan Chu, Professor of Computer Science, University of Maryland. Since CDL is implemented by a FORTRAN program requiring large core, it was run on a time share system utilizing a Univac 1108 computer. Simulation runs required an average of twenty seconds of CPU time. Since some of the routines contained lengthy nested DO loops it would have been too costly to execute the entire routine. Sometimes short cuts were used or simulation not performed at all. The majority of the firmware problems discovered later were found to be in those areas where the simulation was shortened or omitted. Hardware design release to manufacturing was delayed until sufficient CDL simulations were performed to verify design integrity.

A CDL computer model for the AMD 2901 bit slice microprocessor was developed. Next a CDL computer model was developed for the system and included a 24 bit microprocessor, sequencer, pipeline registers and memories.

CDL requires all input data to be in octal format in order to perform simulation. The coding sheet developed for this project - optimized for ease of programming - was incompatible with required CDL input. Therefore, a CDL preprocessor was developed to accept coding sheet data as input and provide the following printer outputs:

1. Echo of input
2. Binary Memory Map
3. CDL Input

A control card input for the CDL preprocessor allows the operator to select any combination of the printer outputs listed above as well as no listing. The CDL input generated by the preprocessor is stored in a file for use by CDL whenever a simulation run is desired.

After processing by the CDL preprocessor is completed, actual simulation runs are performed to verify the microcode integrity. It usually took several runs to remove all coding errors. As a result of performing CDL simulation, coding errors were quickly identified and corrected. The first few CDL simulation runs also identified hardware design errors which were quickly corrected. Actual hardware fabrication did not commence until after CDL simulation verified design integrity. Other than Manufacturing errors, no corrections were made to the design of the hardware contained in the CDL simulation. As a result of performing CDL simulation, a large amount of debugged microcode was available for integration when the hardware was received from manufacturing.

All of the CDL preprocessor activities were performed on a computer system with the following configuration:

Computer:	PDP8/E
Memory Type:	Magnetic core
Memory Size:	32K bytes (12 bits/byte)
Peripherals:	Card Reader- 300 cards/ minute
	Line Printer- 900 lines/ minute
	TTY - ASR33
	Dual Disc - Two sided single platter 3.2M bytes of 12 bit words
	CRT Screen with keyboard
Operating System:	OS/8-3

The CDL preprocessor accepted as input the punched cards containing the desired microcode with the operator assigning a code name each block of code entered. Comment cards are accepted and are indicated by column 1 being non-blank. A "+" character in column 1 indicated that it was the first card for that file and was used during printout to always start a new page.

The CDL preprocessor examines the non - comment columns of each line of non-comment input. Each column is verified to contain a valid symbol. In the event a blank is encountered, the default value is inserted. If no default value has been specified, an error condition exists. Meaningful error messages are provided and the line of microcode containing the error is also printed out. Further processing of the sub-block is inhibited until all errors are corrected.

FIRMWARE/HARDWARE INTEGRATION AND DEBUG

All efforts expended prior to this point have been directed to removing as many hardware and firmware errors as possible. Obviously, all errors will not have been removed. Design changes in both the hardware and firmware will be required to obtain the desired finished product.

To minimize cost and schedule impact during the integration phase, the firmware should be placed in a PROM simulator for final debug. Since all of the desired microcode

is stored in the previously described computer system needed to support the CDL simulation effort, the PDP-8/E can also be utilized to provide the following functions in support of the PROM simulator:

1. Loader
2. Communicator

The function of the loader software is to control the communicator hardware in order to transmit memory content data to a remote terminal (PROM simulator). The loader program requires the operator to define which sub-blocks are to be transmitted. Upon receipt of this information, no further input from the operator is required. The loader first transmits an inquiry to the PROM simulator to determine that a physical connection exists and that the PROM simulator is powered up. If the inquiry is not successful an appropriate error message is printed out on the teletype and further processing terminated. After confirmation of a valid link has been determined a second inquiry is made to determine if the PROM simulator is ready to accept data. If the PROM simulator is not ready for data, an appropriate error message is printed out on the teletype and further processing inhibited. Once communication has been established and verification of the PROM simulator is ready to accept data, the loader program accesses the sub-blocks binary memory maps.

Data is sent to the communicator in seven-bit bytes. The six LSBs contain memory content data and the MSB is used to indicate whether the current word contains data or a command. Therefore, a total of eight words are necessary to transmit one line of microcode for the Control Memory. After a full line of microcode has been sent, a command is generated to load the transmitted data into a buffer register. The loader program now retrieves the data stored in the buffer register and verifies data integrity. If an error is detected, the entire line of microcode is retransmitted and verification of buffer register content repeated. If after two tries, an error condition still exists, an appropriate error message is printed out on the teletype and further processing inhibited. After the memory has been loaded, the entire contents of the loaded memory is read and transmitted back for verification by the loader program.

Appropriate error messages are generated in the event discrepancies are detected between the transmitted and received memory content data. At the successful completion of the loader program, 72K bits of microcode will have been loaded into the PROM simulator and verified in approximately 5 seconds.

The communicator is the hardware necessary to receive the seven-bit word from the loader program for transmission to the PROM simulator. The communicator also receives data from the PROM simulator for use by the loader programmer. It was designed to plug into the PDP-8/E bus and acts like a peripheral device.

In the transmit mode, the communicator routes the received seven-bit word from the loader program to a UART. The UART generates an even parity bit and attaches it to the received seven-bit word. The UART then performs a parallel to serial conversion and sends the eight-bit word to a differential line driver transmission over twisted pair transmission line to the PROM simulator.

In the receive mode, the data is received by a differential line receiver and routed to the UART. The UART calculates the parity bit based on the received seven bits of data and compares it to the received parity bit. If the received parity bit and calculated parity bit are not identical an error flag is set. The UART also performs serial to parallel conversion of received data. The seven bits of received data and error status are then available in parallel for use by the loader program.

In addition to providing two-way communication with the computer system in the acquisition of microcode, the PROM simulator contains the following operational modes:

1. System
2. Control

In the system mode, the hardware undergoing checkout doesn't know that the PROMs are not installed. In an actual system, the clock frequency was 4 MHz. Therefore, a line of microcode is executed every 250 nanoseconds. It was determined that cable lengths between the PROM simulator and the hardware undergoing debug had to be kept as short as possible, therefore, no cable was longer than three feet. Also, all cables were fabricated using twisted pair ribbon cable. One line of each twisted pair was terminated to ground at both ends as close to the source/destination as possible. All inputs and outputs from the PROM Simulator were buffered using Schottky TTL devices.

There were three operating sub-modes in the System mode. In the run sub-mode, normal system operation was performed. In the single step sub-mode, operation was confined to execution of only one microinstruction. A pushbutton was provided such that one microinstruction was executed each time the pushbutton was depressed. This feature was extremely useful during firmware debug. A logic analyzer was connected to the microprocessor output bus to provide a visual display of data coming from the microprocessor. The combination of the logic analyzer and the single-step feature of the PROM simulator turned out to be a very powerful combination for quickly isolating and correcting hardware/firmware anomolies.

It was noted during the run sub-mode, that the LEDs (a total of 92) used for displays induced significant noise as the result of flashing on or off. On several occassions, the noise reached levels high enough to change memory content. A change was implemented whereby the power for the LEDs came

from a separate power supply which was set at 3.5VDC instead of 5.0VDC. This seemed to alleviate the problem. Subsequent PROM simulators inhibited LED displays during the run sub-mode and eliminated the problem entirely.

In the trap sub-mode, operation would halt whenever a certain memory address was accessed. Switches were provided for each of the ten address lines associated with the memory. Whenever the selected address matched the address state of the switches, normal operation would halt. This allowed the operator to halt at a selected point in the firmware such that manual measurements could be made. It also allowed the operator to halt a program if a wrong branch had been taken as the result of a decision point in the firmware. A switch was provided to enable/inhibit the trap function. Sync signals were also provided whenever the trap address switch settings matched the actual memory address. These sync signals in no way interfered with the PROM simulator operation and were used to trigger external test equipment. Sync signals were generated regardless of the trap enable/inhibit switch setting.

The Control mode provided for manual control over RAM content. Switches allowed the operator to select any memory address. LED displays provided visual indication of memory contents. Switches allowed the operator to set the desired state. Push-button switches when depressed, loaded the switch settings into the selected memory address location. The LED display provided visual confirmation that the desired data content had been loaded. This mode allowed the operator to quickly edit stored microcode during the debug stage.

It must be remembered that any editing done on the PROM simulator in no way affects the microcode stored in the computer systems. On several occassions it proved helpful to reload the PROM simulator with the unedited microcode. Whenever it was determined that the edited microcode was valid then the microcode stored in the computer would be upgraded accordingly. A feature deemed desireable but not implemented is to dump the contents of the PROM simulator to the computer system and after copying the unedited microcode, automatically upgrade the microcode for each of the edited sub-blocks. This would allow the next PROM simulator loading operation to contain the edited microcode but still retain the previous unedited code if needed. Another desireable feature to be added would be the necessary hardware and software to program the selected PROM, once the microcode checkout has been completed.

THE GDP-12 GEOPHYSICAL DATA ACQUISITION SYSTEM

R.B. Staley, R.B. Clark, K.L. Zonge
Zonge Engineering & Research Organization
Tucson, Arizona

ABSTRACT

This paper discusses the design and application of an all CMOS, battery powered, backpackable, MICRO-8 system that is used by the mining and oil industry.

INTRODUCTION

For the past six years Zonge Engineering and Research Organization has been using PDP-8's in mineral and oil exploration obtaining field data on the electrical characteristics of rocks using complex resistivity (CR), conventional induced polarization (IP), and time domain electromagnetic measurements. We have two independent systems, each composed of a PDP8M, TU60 DEC cassette and a teletype, mounted in a truck, which runs off a generator at remote sites. These have given exemplary service in extremely harsh environments from sub-arctic to the deserts of the southwest. Each computer is equipped with 16 K of semiconductor memory and two high speed A to D channels with 12 bit precision. External preamplifiers monitor the received and transmitted signals and after signal conditioning through isoamps, feed them to the A to D's in the 8M's. There, each waveform is digitized and stacked in memory, with up to 2048 waveforms stacked.

Our transmitted signal is a square wave with frequencies of .1 Hz through 10 Hz and is transmitted from a high voltage current source which ranges from one to 20 amps. The stored data is manipulated with digital filtering and then a fast Fourier transform is run on both the transmitted and received waveforms. These are deconvolved and the residual phases and magnitudes of the first eleven harmonics are stored on cassette and printed on the teletype. Later these data are used for further analysis on the main office computer which is a PDP8E with hardware arithmetic, 32 K memory, DEC cassette, DEC tape, RK05 disks, Tektronix terminal and hard copy unit, and a line printer.

After using the system for two years we were approached by a company which wanted a portable battery powered system. Since we were thinking along these lines ourselves it was decided to seriously approach the problem. When the project was conceived, the machine was to be a hardwire design with fixed instruction capabilities. After a year of design implementation and development it was decided that our resources were not up to building such a machine from scratch. At about this time Intersil announced the all CMOS 6100. Since we had been working with 8's for three years it became apparent that with the same assembly language we could use our 8E as a development machine for the MICRO-8.

We began working with the 6100 as soon as it became available and spent about a year trying to get an all CMOS system in operation. Intersil had no experience at this time with anything over a 1 K CMOS system and we were attempting to build 12 to 20 K systems with dual processors and a three bus structure. During this time our 8E was used for program development and debug of system software. The software for the GDP-12 was much different than that used on our truck mounted 8M's as we were now only measuring the fundamental of each transmitted waveform and not doing a Fourier transform. This was due to the speed of the five volt MICRO-8 being about one-third of the large 8's in our application.

OVERVIEW OF THE GDP-12

Analog Section

Going through our system we start with the analog channels. In our current system we have dual analog channels with differential inputs and a computer controlled gain stage ranging from 1 to 32,768. DC offsets can be corrected manually with a DC offset potentiometer. This is used to take care of any static ground potentials present. Each analog channel can be controlled separately and channel two may be turned on and off independently of channel one. There are also two digitally controlled low pass Bessel function filters which can be set from .5 to 5120 Hz. These are used in setting the cutoff frequency for aliasing in the analog to digital conversion. The two A to D converters are all CMOS with 12 bit precision and 65 microsecond conversion times. Outputs are latched on the end of conversion and transferred to the bus upon signal from the MPU. End of conversion is signaled by enabling a skip line which causes the processor to grab the digitized data for each channel.

Microprocessor Section

Going to the MPU board, the first thing to consider is the bus structure. There are three separate busses in the GDP-12, one main bus and two time shared. The main data bus is connected to microprocessor one only. This bus provides access to and from the analog board, main memory and master clock. Gain and filter information is passed to

the analog board and data and skips are passed back. Thirty-two K words of memory may be addressed by the main microprocessor on this bus. The master clock is loaded with the desired frequency and sample rate from the main bus which also enables various flags for different modes of operation.

Sample rate controls the A to D converter directly while A to D end of conversion flags the skip line. There are also flags present for specialized signal processing, i.e., quadrature and duty cycle.

Time share bus A allows input and output from the control panel. Processor A or B can be selected on this bus. This allows the control panel to access either processor but not both at the same time. We use an equivalent of the PDP8 control panel for doing debug operations on programs and hardware. This allows for easy modifications of the programs. By using a tristate bus structure either processor may be selected by the control panel. The control panel is ROM controlled and uses the CP mode of the 6100 chip.

Time Share Bus B

Time share bus B is used for data transfer to and from I/O devices, 4 K of shared memory in field 7 and the front panel displays and switches. The I/O devices designated in the system are a teletype terminal and minicassette. The terminal port is either RS-232 or 20 ma loop with a computer controlled baud rate select of 110-9600 baud. The terminal port is through a UART and PIE with hardware modification to make it look more like a DEC teletype port. Minor software changes were made to facilitate hardware incompatibility.

The minicassette is a Braemar computer devices Model 600 with a hardware interface including a PCI and PIE. Custom software is used to give control of data to and from the computer. It can be accessed by either processor and will be used to load programs via the bootstrap loader program and to store data.

Program loading is currently done through a high speed paper tape reader port with parallel data input and handshake capabilities. The manual paper tape reader is capable of loading data in excess of 5000 baud and is our standard method of loading programs into the GDP-12. It has proven to be very reliable, though somewhat clumsy, and allows for fast reloading of programs. Normally with battery backup on our CMOS memory, a reload is necessary only on program change. Presently we are using a commercially available paper tape reader modified to enable us to be independent of ambient light.

Front Panel

The front panel controls allow the operator to control the sequencing of the programs and allow for changing of modes of operation at the operator's discretion. Most of the input switches are self-explanatory. There are switches to control gain, frequency of incoming signal, number of averages to be taken, program select, and some undedicated switches which can be used to input constants used in the calculations. Data is output to three 3 1/2 digit liquid crystal displays. Display one uses the plus and minus signs to indicate which processor is running and also has a numeric readout of program information. Displays two and three are used to output data from the program calculations such as phase and magnitude. The displays also have limited alphanumeric capabilities and are used to display certain information, i.e., low battery or high temperature indications.

Currently the memory structure is set up for a maximum of 28 K of main memory and 4 K of time share memory. Normally the systems we use have 8 K of main memory and 4 K of time share. Main memory is only accessable by processor A which includes fields 0 to 6 while field 7 is reserved for both processors A and B. Processor B can only access 4 K of memory as it has no EMA control. Processor A normally transfers data to field 7 then processor B can work on it while processor A goes back to gathering data from the A to D converters.

Real Time Clock

The other section of the system is the real time clock used to provide synchronization with the transmitted signal. The oscillator is a 5 MHz proportional oven controlled oscillator with four decade divide by N counters used for setting the maximum sample rate and highest frequency of operation. For example if one wanted a sample rate of 8192 Hz and a maximum frequency of operations of 256 Hz, divide 8192 into 5 MHz, getting the closest approximation which is 610. This is a fixed divisor which sets up the maximum sample rate and the maximum frequency to be used. Sixteen selectable sample rates and frequencies are available between the maximum frequency and the minimum frequency in binary multiples. The normal sample rate used is from 8192 to one sample per second and frequencies from 256 Hz to 1/256 Hz. Most of the frequencies we work with range from .125 Hz to 256 Hz.

The oscillator is coupled to the transmitter controller to reset the divider chain to provide a phase reference between frequencies. This also allows doing a drift check between the oscillators to see if they are oscillating at the same frequency. An electrical trim is provided to adjust the frequency of oscillation, and allows the oscillators to be matched in frequency to one part in 10^{11}. A phase meter is used to lock the two oscillators to approximately the same frequency. This phase setting will hold all day for frequencies up to 10 Hz. For higher frequencies one must use more care in adjusting the oscillator trim and check the drift between transmitter and receiver more often.

Power Supply

The internal power for the receiver is five volts DC for all digital circuits except the timing chain and oscillator. We use a 12 volt to 5 volt switching regulator to supply power to the logic circuits. It is capable of about 75% efficiency and has much greater capacity than needed by our circuitry. This is partly due to the fact that our control panel proms are still bipolar and draw fifty time more power than the rest of the system, along with the LED displays which display the address and data registers. There is also a bipolar 15 volt supply which powers the analog section and A to D converters. There is a separate bipolar 12 volt supply which is used for supplying -12 volts for teletype 20 ma operation or RS232. The power consumption of the analog section is around 100 milliamps. By comparison the digital

section with 12K memory and two processors running simultaneously draws only around 10 mills total on an average power basis.

There are built-in calibraters for both the transmitter controller and receiver controller. They are used to check the internal phase shift of the amplifiers and the phase stability of the transmitter controller. Both the transmitter controller and receiver controller are identical and either one can be used to control a geophysical transmitter. The only difference between the two is that the transmitter controller has no sample rate circuitry and has a manual instead of computer controlled frequency select. The controller operates off its own 12 volt battery and uses the battery's internal regulation to provide voltage to the circuits. The crystal oscillator has a self-contained controller to provide a regulated supply for the crystal. The operating range for the rest of the circuitry is from 10.8 to 14 volts. Total power consumption is about two watts at 0°C and maximum voltage. Battery capacity allows operation for 24 hours.

Bootstrap Loader

Our normal program loader is done through a modified control panel function. It is limited to selecting processor A or B, and bin booting with select for cassette, teletype or high speed paper tape reader. A full function control panel may be plugged into the same port allowing normal implementation for all control panel functions: exam, deposit PC, deposit flags, deposit memory and bin boot.

This concludes the presentation of the GDP-12 geophysical data processor. Our future aims are to condense this unit to one-half of its present size and increase memory size using the 4 K CMOS memories which were not available when we started this design. We would like to increase the operating voltage to 10 volts to increase our throughput in the digital section. We are also waiting for faster CMOS A to D converters. We are limited in sample rate at this time due to the 65 microsecond conversion time of our current A to D's. With all the electronics in the main case there would be room in the lid to put the cassette, and battery powered printer.

At present we have thumbwheel switches for data entry but with a printer system we would be able to go to a keypad, with keyed data displayed on a 32 character alphanumeric LDC display, then loaded into the computer and printer on command for execution. This, in part, is our future goal for a second generation remote site data processor.

An Introduction to PASCAL for BASIC and FORTRAN Programmers

by

James A. Krupp
Middlebury College

ABSTRACT

The principal features of the programming language PASCAL are described. Particular attention is paid to the variety of data types supported. Considerations which might affect the choice of PASCAL in favor of BASIC or FORTRAN as an applications language are also presented.

1.0 Introduction

In the six or seven years since the first PASCAL compilers, PASCAL and the concepts of structured programming have become the basis for the first course in Computer Science at many universities and colleges. The reasons for the rapid growth have been well-discussed in the literature and are most apparent to anyone who has struggled with teaching introductory programming using Fortran or Basic.

The purpose of this paper is not to restate the benefits of PASCAL for teaching, but rather to introduce some of the principal features of the language and to assess the relative advantages and disadvantages of programming various applications in PASCAL. The intended audience is the relatively experienced Basic-Plus or Fortran programmer who has heard much about PASCAL and wonders whether it would be suitable for his/her application. The emphasis of this paper will be on "standard" PASCAL as described in "PASCAL User Manual and Report, 2nd ed." by Jensen and Wirth.

(Note: This paper was designed for presentation in a tutorial session at Fall 78 DECUS. It was anticipated that discussion would follow the presentation. It is the author's intent to summarize this discussion in a future issue of the DECUS PASCAL SIG Newsletter).

2.0 A PASCAL Program.

The following pages contain a relatively short, but complete Pascal program which demonstrates several features of the language. The program comments (in curly braces, { and }) describe what it does. It is followed by sample input and a sample program run. This example was selected because the problem is straight-forward and uses many features of the language. The discussion which follows will refer to this program frequently.

2.1 Program Structure - Overview.

Each PASCAL program consists of a heading and a "block". The heading is the PROGRAM statement which identifies the program name and the standard input and output files used by the program. Everything else is considered the block. It consists of six sections: 1)label declaration, 2)contant declaration, 3)type definition, 4)variable declaration, 5)procedure and function definition, and 6)the statement part. In the example program, all of these except "label declaration" are present.

Pascal makes use of reserved words which have special meanings in the definition of the language. Reserved words are printed in uppercase in the sample program; source files can be written in any combination of cases for most compilers. Source files may be completely free-format; however certain conventions are frequently followed to make the programs more readable. The sample program follows one such convention.

2.1.1 LABEL Section - This section declares all labels used in the program. Labels are used as the targets of "GOTO" statements and are unsigned integer constants. It is the exceptional PASCAL program which has any labels or GOTO statements.

2.1.2 CONST Section - PASCAL permits symbols to be equated to constants to aid in program readability. This is analogous to the PARAMETER statement in DEC's Fortran IV-Plus. The example program declares "numsiz", "numscores", and "groupsize" as constants with the values shown. Note that all identifiers, be they names of constants, variables, etc., can be of any length in Pascal; however, "standard" Pascal is only required to recognize differences in the first eight characters.

```
  PROGRAM example1(input,output);

  {Sample program demonstrating some of the
   features of PASCAL.  This program reads
   a list of student names and grades,
   computes an average grade for each and
   then sorts the list and prints it out.

   Features of this example are chosen to
   make the PASCAL reasonably clear to the
   reader unfamiliar with PASCAL.  Therefore,
   certain options in the language have been
   deliberately omitted as being "too
   obscure".}

CONST
  namsiz = 20;
  numscores = 10;
  groupsize = 50;

TYPE
  groupindex = 1..groupsize;
  scoreindex = 1..numscores;
  student = RECORD
              name : ARRAY [1..namsiz] OF char;
              score: ARRAY [scoreindex] OF integer;
              average: real
            END;
  group = ARRAY[groupindex] OF student;

VAR
  i,nstu : groupindex;
  j,ngrades: scoreindex;
  sum    : real;
  infile : text;
  class  : group;
  ch     :char;

  PROCEDURE open(VAR fil:text);

    {this procedure is used to "hide"
     implementation dependent version of
     reset() procedure.}

  VAR
    filnam: ARRAY[1..20] OF char;
    i : 1..20;

   BEGIN
     readln;  {implementation quirk for tt: input}
     write('Input filespec >  ');
     i:=0;
     WHILE NOT eoln DO
      BEGIN
        i:=i+1 ; read(filnam[i]);
      END;
     readln;
     reset(fil,filnam);
   END;
```

```
PROCEDURE classort(VAR class:group; n:groupindex);

  {Sorts class on average using a recursive
   quicksort algorithm from Wirth's
   "Algorithms + Data Structures = Programs"}

  PROCEDURE sort(l,r:groupindex);

  VAR
    i,j: groupindex;
    x,w  : student;

   BEGIN
     i:=l; j:=r;
     x:=class[(l+r) DIV 2];
      REPEAT
       WHILE class[i].average < x.average DO i:=i+1;
       WHILE x.average < class[j].average DO j:=j-1;
       IF i<=j
        THEN
         BEGIN
           w:=class[i];
           class[i]:=class[j];
           class[j]:=w;
           i:=i+1;  j:=j-1
         END
      UNTIL i>j;
     IF l<j
      THEN sort(l,j);
     IF i<r
      THEN sort(i,r);
   END    {sort procedure};

 BEGIN
   sort(1,n)
 END  {classort procedure};

PROCEDURE listclass(VAR class:group;
                    nstu:groupindex; nscore:scoreindex);

  {This procedure lists the students in order by
   average score.}

VAR
  i,j: integer;

 BEGIN
   writeln;
   write('   Student          ');
   FOR i:=1 TO nscore DO write(i:4);
   writeln('  Average');
   writeln;
   FOR i:=1 TO nstu DO
   WITH class[i] DO
    BEGIN
      FOR j:=1 TO namsiz DO write(name[j]);
      FOR j:=1 TO nscore DO write(score[j]:4);
      writeln(average:8:2)
    END
 END  {listclass procedure};
```

```
BEGIN  {start of main program}
  open(infile);
  readln(infile,nstu,ngrades);
  FOR i:=1 TO nstu DO
   BEGIN
     class[i].name:='
     j:=0;
     read(infile,ch);
     WHILE (ch<>',') AND (j<namsiz) DO
      BEGIN
        j:=j+1;
        class[i].name[j]:=ch;
        read(infile,ch)
      END;
     IF ch<>','
      THEN
       REPEAT
        read(infile,ch)
       UNTIL ch=',';
     sum:=0.0;
     FOR j:=1 TO ngrades DO
      BEGIN
        read(infile,class[i].score[j]);
        sum:=sum+class[i].score[j];
      END;
     class[i].average:=sum/ngrades;
     readln(infile)
   END   {for loop};

  classort(class,nstu);
  listclass(class,nstu,ngrades);
END.
```

Sample input file:

```
7,5
dkjfjl,56,78,96,43,24
abc,23,34,35,46,67
defghi,34,45,63,74,97
dskflskjflk,12,23,24,35,45
absurdlylongnamefortesting,1,23,34,45,67
dkdkdlsls,33,22,44,55,66
dkjflsk,45,34,67,98,34
```

Sample program run:

```
RUN EXAMPL
Input filespec >  TEST.DAT

   Student              1    2    3    4    5  Average

dskflskjflk            12   23   24   35   45    27.80
absurdlylongnamefort    1   23   34   45   67    34.00
abc                    23   34   35   46   67    41.00
dkdkdlsls              33   22   44   55   66    44.00
dkjflsk                45   34   67   98   34    55.60
dkjfjl                 56   78   96   43   24    59.40
defghi                 34   45   63   74   97    62.60

Ready
```

2.1.3 TYPE Section - This section is used to extend the basic data types provided in the language. Section 2.2 below on "Data Structure" will discuss data types further.

2.1.4 VAR Section - All variables in a PASCAL program must be declared. This requires some adjustment in programming style for BASIC and FORTRAN programmers. However, this adjustment comes quickly (PASCAL compilers flag any statement containing undeclared variables) and has many benefits. One benefit is compiler type checking, which allows the compiler to detect incorrect expressions and assignments before such errors have a chance to produce obscure errors in executable code. For large programs, the requirement that all variables be declared helps immeasurably in program maintenance.

2.1.5 PROCEDURE Section - Declaration of procedures (i.e., subroutines) and functions is like writing a program within a program. Namely, the procedure or function statement plays the role of the program statement in the header. The rest of the procedure definition is a block consisting of exactly the same six parts which make up a program. This leads to "nested" variable and procedure definitions and the idea of the "scope" of a definition. Variables defined within a procedure are local to that procedure. Variables defined in procedures outside the current procedure are considered global to the current procedure. Variables defined in the main program are global to all procedures. If the same variable name is declared twice, the "closest" definition is used.

In the sample program the variables i,nstu,j,... declared in the main program are accessible within all of the procedures, unless the same identifier is redefined within the procedure. In "open", the variable "i" is redefined and is thus different than the "i" defined in the main routine. The variable "j" could be accessed within "open". The array "filnam" cannot be accessed from the main program. The scope rules for variable names apply equally to all identifiers in a PASCAL program.

Parameters may be passed by value or by reference; the procedure or function statement identifies the type of argument transmission for each parameter. VAR preceeding a parameter name means it is passed by reference; otherwise it is assumed to be passed by value. Functions and procedures may also be passed as arguments. All procedures and functions in PASCAL may be used recursively. The various procedures in the sample program show these features.

2.1.6 Statement Section - This is the "program"; it starts with the keyword BEGIN and continues through the last END. The preceeding 5 parts have defined all the variables, constants and procedures available to the program. For large applications developed in a "top-down" way, the main program is usually little more than a series of procedure invocations within a simple loop. The elements of the language which make up the statement part are described below. In the sample program, a comment identifies the start of the statement section.

2.2 Data Structure.

The wide range of data types within PASCAL is potentially the most confusing aspect of the language. The language provides four basic scalar types: integer, real, boolean, and character. However, this simple beginning is augmented by methods for defining additional scalar types and for structuring these types, that is, grouping them together into complex units. The structured types are: arrays, records (i.e., the PL/I "structure"), sets (similar to PL/I "bit strings"), and (sequential) files. In addition the type "pointer" exists for use with dynamically allocated data structures.

The discussion which follows will be somewhat limited. However, enough will be covered to allow the experienced programmer to see the potential for clear and efficient programming of his/her own applications in PASCAL. The word "type" will appear frequently in this discussion. By "type" we mean the characteristics associated with a constant or the range of values which a variable may take on. For example, integer type means whole numbers between -32768 and 32767 (for most PDP-11 implementations). The concept of data type is fundamental to PASCAL. The TYPE section of a program is where a programmer defines the characteristics of a new data type and gives this type a name.

2.2.1 Scalar Types - Intrinsic. The types real, integer, and boolean correspond to real, integer, and logical from Fortran. The type char (for character) is used to represent the printing and non-printing characters; the byte type from Fortran is similar (it stores one character), but PASCAL does not permit arithmetic on variables of type char (as Fortran does on byte data). These four types are referred to as scalars.

2.2.2 Scalar Types - User-defined. Other scalar types may be defined by the programmer. Scalars are distinct from structured types such as arrays. Scalars possess order (e.g., integers), but they lack any structure such as a 2-dimensional array of integers might have (e.g., "rows" and "columns"). To define a new scalar type, one includes a definition in the TYPE section of the program. For example,

```
TYPE
  figure=(circle,triangle,square,star);
```

Subsequently, variables may be declared of type "figure", e.g.,

```
VAR
  obj1,obj2: figure;
```

The result of scalar definition is to define a set of programmer specified symbols (here, circle, triangle, etc.) as constants of a new type (figure). Then variables declared of this type may only take on values from this collection; e.g., obj1 may be assigned the value of circle, but not the value 1. Types so declared impose an ordering on the objects which make up the type; namely, the first item is less than the second is less than the third, etc. Special functions succ() and pred() are supplied for obtaining the successor value or predecessor value, resp; for example, succ(circle)=triangle. The obvious application of programmer defined scalar types is to remove obscure integer values from a program and to replace them by a suitable identifier.

Another way to define additional scalar types is to specify one type as a subrange of another type. For example,

```
TYPE
  counter = 1..60;
```

defines a type counter to be limited to the subrange 1 to 60 of the type integer. Variables declared of type counter could be used in expressions involving integers, however, any attempt to assign a value larger than 60 or less than 1 to such a variable would produce a runtime error (provided such run-time checking were in effect). As experience with PASCAL grows, a programmer will most likely find him/herself using subrange types extensively. Use of subranges provides excellent documentation on how variables are supposed to behave, documentation which substantially aids in the maintenance of large programs. Several subrange types are used in the example program.

2.2.3 Structured Types - The Array. The simplest structured type is the array. This is defined exactly as in other languages. A variable definition involving an array would appear as

```
VAR
  a,b,c : ARRAY [1..10,1950..1958] OF REAL;
```

Here, each of a(), b() and c() is defined as a two-dimensional array. The range of each subscript may be any scalar or subrange type, except real. Arrays may be of any dimension. The storage of multi-dimensional arrays is by row. Reference to array elements are of the form a[i,j], where the i and j may be any expression which yields a value of the appropriate index type used in the definition of the array. Array bounds must be constant; there is no dynamic array allocation in PASCAL.

2.2.4 Structured Types - The Set. A variable declared as a set of objects may be viewed as a "bit-string", i.e., the presence or absence of an object from the set is determined by its bit being set or cleared. For example,

```
TYPE
  collection = SET OF figure;
VAR
  p,q,r: collection;
```

Here figure is the scalar type declared above; it could be any scalar or subrange type. The maximum size of a set is implementation dependent; typical PDP-11 implementations allow 64 elements. Variables p, q, and r can be assigned values, e.g.,

```
p:=[circle,triangle];
```

assigns the set consisting of the two elements circle and triangle to the variable p. This is equivalent to setting bits in p to show the presence of these items. Set operations include union and intersection (and-ing and or-ing of bit strings) and relational operations of set inclusion (bit testing). An operation for determining the presence of a particular element in a set is also provided. This is particularly convenient for range searches. For example, the following could be used to determine if ch is one of the characters 'A', 'G', 'P', 'T', or 'Z'.

```
VAR
 ch : char;
 letters : SET OF char;

....
 {in the statement part}
letters:=['A','G','P','T','Z'];
....
 IF ch IN letters THEN ...
```

2.2.5 Structured Types - The Record. The record is the most powerful data structuring technique in PASCAL. It allows the grouping together of many different types, including type record itself, into a single data type. This type is most easily demon-

strated by some examples.

```
TYPE
 complex = RECORD re:real;
                  im:real
           END  {declare type complex as
                 an ordered pair of reals}

 date = RECORD month: 1..12;
               day  : 1..31;
               year : 1900..1999
        END;  {date as an ordered triple}

 employee = RECORD name: ARRAY [1..40]
                                   OF char;
                    birth: date;
                    start: date;
                    code : (aa,bb,cc,dd)
            END;  {an employee record}
```

The type student in the sample program is another example. The individual declarations within a record are called fields. Record types also permit "variants", in which a single type is defined with one or more fields whose definition varies according to the value of another "tag" field.

Records are frequently used for dynamic data structures. This is done by including one or more fields of type "pointer" which can be assigned values "to point" at another record, producing arbitrary linked data structures. PASCAL provides procedures for allocating records and setting a pointer variables to point at them. This feature together with procedures that may be used recursively means that most algorithms on graphs can be simply and efficiently implemented in PASCAL.

2.2.6 Structured Types - The File. Only sequential files are supported in standard PASCAL. Files are thought of as sequences of elements of some type on which certain operations are permitted. For example, appending an element to the end of a sequence corresponds to writing onto the file. If fil is declared to be a file of some type, and t is a variable of the same type, then the statements

```
fil^:=t;  put(fil);
```

assign the value of t to the buffer variable (fil^) and output the buffer. Since data can be accessed directly in the buffer, many file operations can be performed quite efficiently in PASCAL.

Files may be of any valid type. Files of characters, textfiles, are treated specially; namely, the line organization of textfiles is marked with special 'end-of-line' characters which can be detected when a file is read, or inserted when the file is written. Textfiles are very similar to stream-files in PL/I. The sample program shows the use of textfiles and I/O to the user's terminal.

2.2.7 Some Final Comments on Data Structures - It is not possible to provide a feeling for the richness of PASCAL's data types in so short a presentation. In particular the use of pointers with records for dynamic data allocation makes implementation of many algorithms very clean and efficient. Likewise, the true strength of such tools as variant records and sets is hard to appreciate without looking at large and complex programs which use them.

There is of course one data type which is conspicuous by its absence, namely string. Strings are a major weakness of PASCAL. They are defined as arrays of characters, and, therefore, must be of fixed length. This together with explicit type checking makes the string manipulation facilities very primitive when compared with even the simplest of BASIC interpreters. In the area of strings, PASCAL is about on par with a Fortran compiler supporting CHARACTER data type.

2.3 Control Structures.

Compared to the complexity of data types, the control structures within PASCAL are simple to describe and understand. The element which simplifies it the most is the concept of a compound statement. If we let the letter S stand for any simple statement in PASCAL, then the group of statements

```
BEGIN S; S; ... S; S END
```

is a compound statement and may be placed anywhere a simple statement is permitted. In fact, the statement part of a PASCAL program may be considered a single compound statement. Note that the semi-colon (;) serves as a statement separator in PASCAL, unlike PL/I where it serves as a statement terminator.

Simple statements consist of assignment statements, procedure invocations, conditional or selective statement execution (the IF and CASE statements), and looping statements (WHILE, REPEAT, and FOR). There is also the unconditional transfer statement (GOTO), but this is rarely used in well-written PASCAL programs. Since these control structures are relatively familiar from other programming languages, only a few brief comments on each will follow.

2.3.1 Assignment Statement - The assignment operator is ':=', to distinguish it from the relational operator for equality. Assignment is only allowed between expressions and variables of identical type. The exception is that subranges of the same scalar type may be mixed and integers will be automatically coerced to type real. The conversion from real to integer must be explicitly performed using the functions trunc() or round().

2.3.2 Procedure Invocation - Procedures are "called" simply by writing their name; see the last two lines of the sample program. Functions are invoked by using them in an expression.

2.3.3 IF...THEN...ELSE and CASE...OF Statements - Conditional execution is via the IF statement. A boolean expression follows the IF; a true value causes the THEN "clause" to be executed; a false value the ELSE "clause". The ELSE "clause" is optional. "Clause" is a simple or compound statement.

The CASE...OF statement is similar to BASIC's ON...GOTO or Fortran's arithmetic GOTO; it can be used with any scalar type except real. For example,

```
CASE obj1 OF
 circle: <S1>;
 triangle: <S2>;
 square,star: <S3>
END
```

will execute exactly one of the compound statements <S1>, <S2>, or <S3> according to the value of obj1.

2.3.4 WHILE, REPEAT, and FOR Looping Statements - Each of these statements controls the execution of a loop. The formats are

```
WHILE <boolean exp.> DO S;

REPEAT  S  UNTIL <boolean exp.>;

FOR <scalar>:=<scalar exp.>
                    TO <scalar exp.> DO S;
```

Each of the statements S in the WHILE and REPEAT statements is executed until <boolean exp.> is false or true, resp. The major difference between WHILE and REPEAT is when the test is performed; WHILE tests before S is executed, and REPEAT tests after S is executed. FOR loops have a step of +1 or -1; TO is used for +1 and DOWNTO is used for -1. Any scalars, except real, may be used as a FOR loop index, e.g.,

```
FOR obj1:=circle TO star DO ....
```

3.0 Some Implementation Features.

There are several implementations of PASCAL available for the PDP-11. Most of these provide essentially all of the features of standard PASCAL together with various "enhancements" to the language. Some of these enhancements improve the ease of programming in PASCAL; others do violence to the basic precepts of the language's design. Some of the more helpful and common enhancements follow.

1. An EXIT statement to cause premature exiting from WHILE and REPEAT loops. Others provide a separate LOOP construct which is similar to WHILE, but makes explicit provision for premature loop exit.

2. Inclusion of an ELSE statement in the CASE statement. The statement following the ELSE would be executed if the scalar value didn't match any of the explicitly listed case labels.

3. Boolean operations (AND, OR, NOT) on integer data types.

4. Special mechanisms to allow easy I/O to the user's terminal for interactive programs.

5. Support of structured constants. For example, if z were declared as complex, something like

   ```
   z:=complex(3.0,5.0);
   ```

   is permitted. Standard PASCAL would require

   ```
   z.re:=3.0;  z.im:=5.0;
   ```

   for this assignment. Note that if z and w are both of type complex, "z:=w" is supported in standard PASCAL.

6. Functions which may return a value of structured type. Standard PASCAL supports only scalar valued functions.

7. Random access files and procedures for associating a file name and device with a file variable.

8. External procedures and separately compiled procedures This feature is necessary for large applications requiring overlaid code. Calling conventions are usually documented well enough to permit calls to assembly language routines. Some implementations provide a direct mechanism for calling Fortran subroutines.

9. At least one implementation provides a debugger which makes debugging PASCAL almost as easy as debugging Basic-Plus.

This list is not exhaustive. However, even as it stands, it clearly demonstrates why many people are very concerned about developing standards for the language before implementations become so fragmented that PASCAL becomes as non-transportable as Basic.

4.0 PASCAL vs. BASIC-PLUS or FORTRAN

Having reviewed the basic features of the language, the question arises, "For which applications should PASCAL be selected?" If the decision is between Fortran and PASCAL,

PASCAL will beat Fortran in areas of ease of implementation and program efficiency in almost every application. The exceptions are applications requiring double precision or complex arithmetic; PASCAL has neither. If random access files are used, then "standard" PASCAL would have to be by-passed. However, most implementations provide some sort of random access file capability. If portability is an issue, PASCAL and Fortran rank about the same, if you use only "standard" PASCAL or "standard" Fortran.

When the choice is between Basic-Plus and PASCAL, the decision will be more difficult. The following is a partial list of considerations.

1. If extensive use of Basic strings and string functions seems likely, programming in PASCAL may require major revisions to the approach being used. For example, the Basic functions LEFT(), RIGHT(), INSTR(), have no analog in PASCAL. If you are planning significant amounts of command parsing using these functions, this portion of your application would require complete re-design if written in PASCAL.

2. If your application is a "quick and dirty, one-shot" program, use Basic. It wouldn't be worth your energy to use PASCAL. If your application is a large program, or series of programs, the inherent efficiency of program execution with PASCAL may be worth whatever additional energy is required for programming it.

3. Well-written PASCAL is easy to read and maintain, once you become familiar with the language. Poorly written PASCAL is at least as bad as poorly-written anything else. If you can develop your application using the methods of "top-down" design, then writing it in PASCAL will be quite easy. The larger the application, the greater the relative advantages of PASCAL. If you develop programs by continuous "hacking" at a crude first attempt, PASCAL will probably not help you.

4. If you require access to monitor tables and special features in your application, this will require external procedures written in assembly language. However, once this is done, programs written using these procedures have the potential of being infinitely easier to understand than their counterparts in Basic-Plus. The principal reason for this is that all the "ugly stuff" can be hidden in a procedure; it won't intrude upon your program the way PEEK sequences and SYS() calls usually do.

5. The extensive error facilities of Basic-Plus are simply unavailable in PASCAL. Most implementations will allow recovery from common file system errors, but this is not "standard" PASCAL and does require significantly more work than is required by Basic.

6. Applications involving complex in-core data structures are more easily written in PASCAL than in Basic-Plus. However, if these applications require fancy file manipulation, care must be taken. This is particularly true if files of structured types are created. PASCAL says nothing about how such files appear on external devices. If access to these files is required by non-PASCAL programs, significant problems could arise.

7. Since PASCAL is not currently supported on PDP-11's by DEC, you must be somewhat of an adventurer to select PASCAL as your implementation language. The available compilers are generally solid, but not without their "kinks" that can cause a great deal of lost time. However, most of the problems the author has experienced while learning PASCAL are no worse than problems which arise from trying to be too clever with Basic-Plus or Fortran.

The most important consideration is to realize that PASCAL cannot be the best solution to all of your problems. There are clearly applications where it will excel; there are others for which it would be a very poor choice.

5.0 References

For persons wishing to know more about PASCAL the following short list is provided. Jensen and Wirth contains the definition of "standard" PASCAL. The other book by Wirth has some excellent algorithms written in a slightly modified PASCAL. The book by Grogono is the clearest text on PASCAL at this time; it also has a very complete bibliography on other books and articles on PASCAL.

Grogono, P. -- Programming in PASCAL, Addison-Wesley, 1978.

Jensen, K. and Wirth, N. -- PASCAL User Manual and Report, 2nd Edition, revised. Springer-Verlag, 1978 (corrected printing).

Wirth, N. -- Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

LSI-11 WRITABLE CONTROL STORE ENHANCEMENTS TO U.C.S.D. PASCAL

Gordon Smith and Roger Anderson
Lawrence Livermore Laboratory
University of California
P.O. Box 5507
Biomedical Sciences Division
Livermore, California 94550

ABSTRACT

The DEC KUV11-AA Writable Control Store was used to implement selected portions of the U.C.S.D. Pascal P-machine in firmware. The frequency and execution speed of P-machine instructions were measured in a battery of test programs to guide the selection process. A 32 to 46% reduction in execution time has been obtained for these test programs.

INTRODUCTION

The recent release of the KUV11-AA Writable Control Store (WCS)[1] has permitted user access to a level of the LSI-11 previously restricted to a select few. A number of interesting projects are possible with this product such as the application described here, the microprogramming of portions of Ken Bowles' U.C.S.D. Pascal P-machine.[2]

The LSI-11 Microprocessor and Writable Control Store

The LSI-11 is a microprocessor microprogrammed to emulate the PDP-11 instruction set. The microprocessor has 26 eight-bit registers that are addressed by a combination of direct and indirect means. Its instruction set is highly vertical, i.e., it is not unlike a conventional, albeit primitive, machine language. Control is exercised by a Translation Array, consisting of four programmed logic arrays, that examines the fetched PDP-11 level machine instruction and determines where microprogram execution is to begin. The Translation Array may continue to exert control by generating new inputs to the location counter as a function of the current value of the location counter, interrupt signals, and other control inputs.

The memory address space is 2K words. It is divided into four 512 word pages. Half of this address space, or two pages, is used to emulate the PDP-11. The EIS/FIS chip is optional and adds a third page of microcode that emulates an extended PDP-11 instruction set. These additional instructions include integer multiply and divide plus a battery of floating point instructions. The fourth page is left unused.

The WCS contains a 1K, or two page, random access memory that is primarily intended to be used as the third and fourth page of the microprocessor memory. Use of the WCS as the third page of memory is restricted by the Translation Array. Normally the EIS/FIS code resides on this page. To facilitate its execution the Translation Array is programmed to perform various control functions when execution reaches specific memory locations on the page. User microprograms must avoid these locations. Complications can be avoided entirely by loading the EIS/FIS code into one page of the WCS and restricting new microprograms to the other.

DEC has allocated opcodes 76700-76777 for user microprogramming. When the Translation Array encounters an opcode in this range, it directs control to a single address in the fourth page of memory. The user is then responsible for decoding the individual opcodes. Use of the Translation Array to facilitate execution of user microprograms is not supported.

The U.C.S.D. Pascal P-machine

The U.C.S.D. Pascal system is a complete stand-alone system designed to run on micro- and minicomputers. One of its most impressive features is its use of an underlying P-machine. The P-machine is a stack-oriented pseudocomputer that exists as an interpreter written in the assembly language of the host computer. Pascal source code is compiled to an intermediate P-code that is, in effect, the assembly language of the P-machine. This design makes the system highly portable. The operating system itself is written in Pascal. Only the relatively small native code interpreter must be written to transfer it to a new host. To date there have been successful implementations on the PDP-11 series and the 8080 family of microprocessors, including the 8080A and 8085. Other advantages of this type of implementation include the efficient use of small memories and fast compilation speed.

PROJECT DEFINITION

As described above, the execution of the U.C.S.D. Pascal P-machine is a two-level process. The LSI-11 microprocessor emulates a PDP-11 computer, which in turn simulates the P-machine. We are exploring the possibility of having the LSI-11 microprocessor emulate the P-machine directly. The primary advantage will be an increase in program execution speed.

One of our initial observations was that there is not enough room in the WCS to implement the entire

P-machine. On the average, it requires more microcode than macrocode to implement a P-machine instruction. One measure put the ratio at 1.7 words of microcode for every word of macrocode. The current LSI-11 macro-level interpreter requires 2.4K words. A microprogrammed P-machine, we estimate, will require 3 to 4K words of microcode without the use of the Translation Array.

For this project we used one of the pages of the WCS to implement portions of the P-machine and the other page to contain the EIS/FIS code. Our task was to select the best portions to microcode.

SELECTION OF PORTIONS OF THE P-MACHINE TO MICROCODE

Control Structure

To execute a P-machine instruction, an opcode must be read from macro-level memory, control transferred to the appropriate routine for execution, and control returned for the next instruction fetch. This process will be called the interpreter fetch sequence. In the macro-level P-machine interpreter, this takes approximately 25 μs for most instructions and 30 to 45% of the total program execution time. This data makes the interpreter fetch sequence an excellent candidate for microprogramming.

The interpreter fetch sequence in our micro/macro interpreter uses a variation of the scheme used in the macro-level interpreter. In this scheme the P-machine opcode is used to index a table of macro-addresses for the respective routines. This same table is used by the mcro/macro interpreter with the difference that micro-addresses are also stored in the table. The addresses are differentiated by having the high order four bits of the words containing the 11-bit micro-addresses set to ones. Macro-addresses never have these bits set because the high end of memory is normally reserved for I/O devices.

The microcoded interpreter fetch sequence has two entry points. First, its execution can be initiated from macrocode by the opcode 76704. Second, after a microcoded P-machine instruction is executed, a jump is made directly into the interpreter fetch routine. The direct entry from microcode is faster than from macrocode.

The speed of the microcoded interpreter fetch sequence averages approximately 14.5 μs for most instructions. This is somewhat disappointing, but nevertheless is our most successful single microcoded routine. It alone can reduce program execution time by 12 to 19%.

The micro/macro control structure can handle microcode instructions in addition to 76704. Useful instructions include general purpose instructions such as a block move. Other useful ones are specialized instructions that execute the frequently used parts of a P-machine instruction, leaving the logic for handling special cases or error conditions in macrocode. The scheme for supporting the non-76704 instructions centers around the use of the microstruction, Modify Instruction (MI). This technique is discribed in detail in the WCS Users Guide.[1] Briefly, the MI instruction uses the fetched macro-level instruction to index a table of jump instructions in the microprocessor memory.

Another key element of the control structure is the handling of interrupts. Periodically, during the execution of long microcode routines a check is made to see if interrupts are pending. If an interrupt has occurred, execution is aborted and control returns to macrocode to service the interrupt. After the interrupt has been serviced, the micro-routine is restarted from the beginning. Microcoded routines that may be aborted must be careful to postpone making permanent changes until after the last interrupt check.

P-machine Instructions

A series of tests were conducted to determine the best P-machine instructions to microcode. For these tests a DEC KWV11-A programmable real-time clock was used to maintain a running count of the number of microseconds spent executing each P-machine instruction. Also, each instruction was counted as it was executed. From this data the average execution speed, percent of program execution time, and percent of instruction execution frequency were calculated for each instruction. Instructions with a high percentage of program execution time and/or a high frequency of execution are prime targets for microprogramming. Frequency of execution is important because of the faster direct entry from microcode to the interpreter fetch sequence.

Unfortunately, there does not exist a typical program that can be tested. Instead a battery of test programs was assembled to gain insight into commonly used programs. These test programs are as follows:

Compilations - Six programs totalling 3341 lines of source code were compiled. The programs were selected at random. Two were written by one of the authors of this paper, one was WHETSTONE (see below), and three (XREF, CALC, & RT11TOEDIT) were selected from the software distributed by U.C.S.D..

WHETSTONE - This program is a synthetic benchmark developed by H. Curnow and B. Wichmann[3]. It exercises a computer in a manner considered typical of scientific programs. Specifically, it includes array manipulation, conditional jumps, procedure calls, integer arithmetic, and trigonometric and other standard functions using real numbers.

Sorts - Three programs, Quicksort (recursive), Quicksort (nonrecursive), and Heapsort, were used to sort an identical array of 3,000 reasonably random

integers. The algorithms used were based on those given by N. Wirth[4].

Miscellaneous exploratory programs

Two programs were run in the hopes of gaining special insights into the behavior of the P-machine. The first creates a cross-reference of a Pascal source program. This XREF program was written at Sperry-Univac. The second (BTSI) builds a balanced tree in the heap and conducts searches of that tree. Again, the algorithm was based on one by N. Wirth[4].

An example of these test results is presented in Table 1.

With this test data, we are able to select portions of the P-machine to microprogram, code those portions, and then evaluate the resulting performance. The ultimate basis for the evaluation is the percent reduction in program execution time derived per word of WCS used and the consistency of the improvement across the spectrum of test programs.

RESULTS

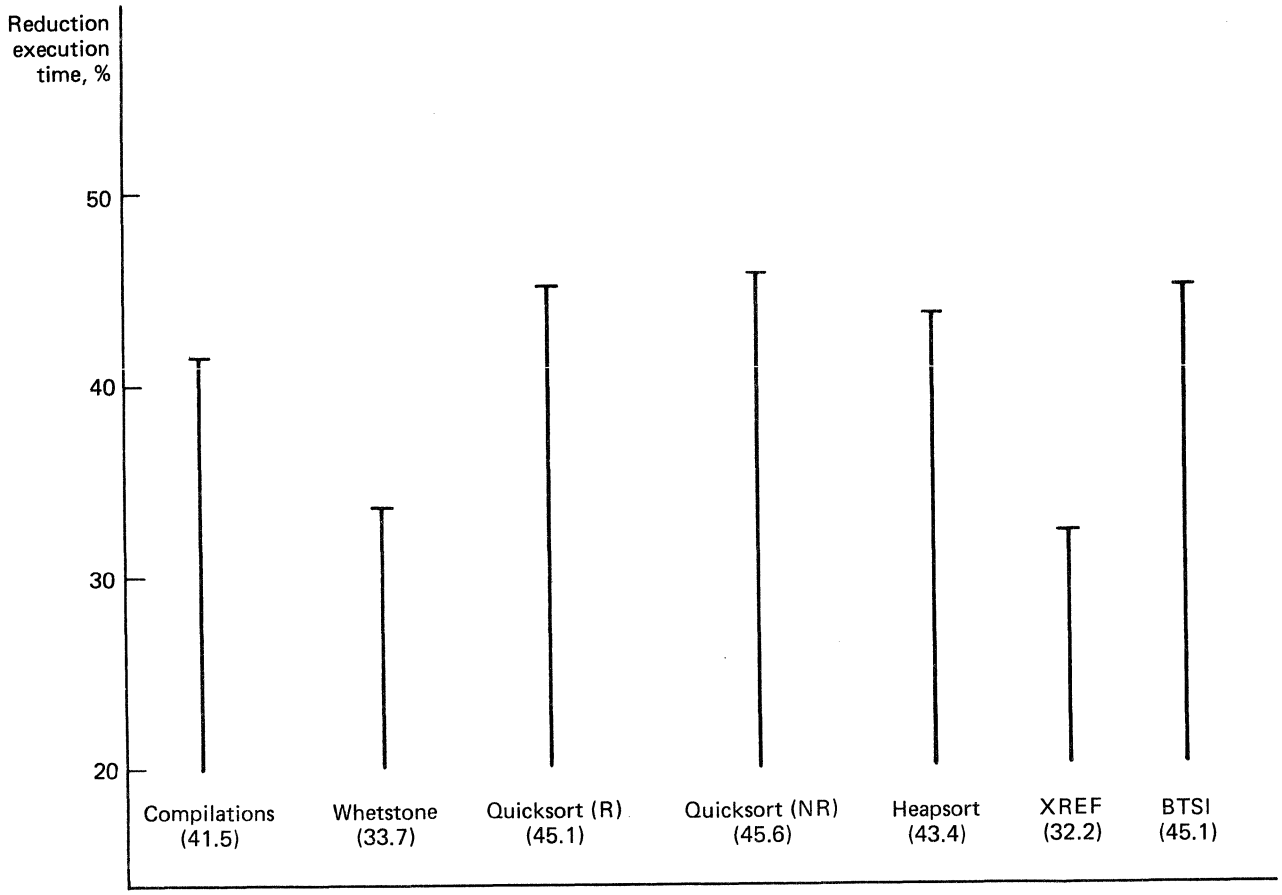
To date a page of microcode has been coded. All of this code was written before the test series described above was completed, although preliminary test results were available. Tests using a line time clock have measured a 32 to 46% reduction in execution time for the test programs when compared to the macro-level LSI-11 interpreter. (Note, both interpreters use the EIS/FIS code). These results are shown in Fig. 1. The page of microcode contains 19 P-machine instructions and four general purpose instructions. The speed improvements

Table 1. Execution speed, percent of execution time, and execution frequency of individual P-machine instructions coded in LSI-11 assembler language. These results were obtained from the compilation of 3341 lines of source code. The average execution speeds are adjusted to eliminate the time for the interpreter fetch sequence. The percent of program execution time includes the interpreter fetch sequence time in the calculation. The interpreter fetch sequence and the single instruction SLDC are in microcode.

Mnemonic	Instruction	Average execution speed in μs	Percentage of program execution time	Percentage of execution frequency
CIP	Call Intermediate Procedure	632	21.6	2.0
CSP	Call Standard Procedure	1186	17.5	0.9
FJP	False Jump	25	3.7	8.7
RNP	Return From Non-base Procedure	75	3.1	2.5
SRO	Store Global Word	31	2.5	4.7
SLDO	Short Load Global Word, total	12	2.5	12.7
INN	Set Inclusion	114	2.1	1.1
SLDL	Short Load Local Word, total	12	1.7	8.4
LDO	Load Global Word	32	1.6	3.0
CLP	Call Local Procedure	205	1.6	0.5
EQUI	Integer Comparison, =	20	1.6	4.5
UJP	Unconditional Jump	22	1.5	3.9
LDM	Load Multiple Words	68	1.4	1.2
STL	Store Local Word	31	1.3	2.5
CXP	Call External Procedure	487	1.1	0.1
XJP	Case Statement	63	1.1	1.0
ADI	Add Integer	10	1.0	6.0
UNI	Set Union	92	0.8	0.5
LDB	Load Byte	21	0.8	2.3
LAO	Load Global Address	31	0.7	1.4
SIND	Short Index and Load Word, total	17	0.7	2.4
LLA	Load Local Address	31	0.7	1.3
SLDO12	Short Load Global Word, offset 12	12	0.7	3.4
SLDC	Short Load Word Constant, total	3	0.6	15.3
IXA	Index Array	75	0.5	0.4
SLDO3	Short Load Global Word, offset 3	12	0.5	2.6

The remaining Instructions have percents of program execution time of less than 0.5% and percents of total frequency of execution of less than 2.3%.

Fig. 1. Percent reductions in program execution time obtained by the micro/macro interpreter when compared to the macro-level interpreter.



obtained for the P-machine instructions and the number of words of WCS required to code them are given in Table 2. The general purpose instruction are:

An instruction to retrieve "BIG" operands. These operands may be either one or two bytes long, depending on whether the sign bit of the first byte is set.

An instruction to traverse down the static links of the P-machine stack n levels.

A block move instruction that increments the source and destination addresses as each word is moved.

A block move instruction that decrements the source and destination addresses as each word is moved.

The microcode produced to date and the detailed test results and procedures are available from the authors on request. Test were run using the LSI-11/2 (KD11-HA) with the MSV11-DD 32K memory.

WHAT CAN BE DONE

Work is still in progress. Both the selection of routines to microcode and the density of the microcode can be improved. The current reduction in execution time, as previously mentioned, is 32 to 46%. We expect that this figure can be improved by several percentage points. A final figure of roughly 45 to 55% seems to be possible. When it is completed, this code could be programmed into read-only memory and made available at a price considerably lower than the WCS board. Although such a product may not be of widespread interest, some installations may find it worthwhile.

An exciting possibility is the complete conversion of the microprocessor to a P-machine. In particular, if the Translation Array could be used for the interpreter fetch sequence and other control functions, speed improvements should far outstrip anything that can be accomplished with a single page of WCS. Space will still be a major problem even with the Translation Array. If this problem can be overcome speed improvements by a factor of four or more do not seem unrealistic.

ACKNOWLEDGEMENT

Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract number W-7405-ENG-48.

Table 2. Microcoded P-machine instruction execution times and the number of WCS words used. The timing results were obtained from a test consisting of two compilations, the Whetstone and Quicksort (nonrecursive). Execution speeds are adjusted to eliminate the time for the interpreter fetch sequence. Similarly, microcode for the interpreter fetch sequence is not counted in the "words of WCS" figures. The number of "Words of WCS shared" with other routines are included in the number of "Total words of WCS".

*Figure is an estimate.

Mnemonic	Instruction	Micro time, μs	Macro time, μs	Total words of WCS	Words of WCS shared
AND	Logical And	5	17	6	0
CHK	Check Subrange Bounds	22	26	19	0
CIP	Call Intermediate Procedure	188	542	29	0
CLP	Call Local Procedure	75	189	113	0
FJP	False Jump	4	24	20	13
GRTI	Integer Comparison, >	10	23	17	9
LAO	Load Global Address	10	38	22	13
LDCI	Load Constant Word	5	22	8	0
LDM	Load Multiple Word	22	56	24	0
LDO	Load Global Word	10*	32	24	13
LEQI	Integer Comparison, ≤	11	21	17	9
LLA	Load Local Address	9*	31	22	13
NEQI	Integer Comparison, ≠	8	21	12	9
RNP	Return From Non-base Procedure	24	74	49	0
SLDC	Short Load Word Constant	3	6*	4	0
SRO	Store Global Word	10*	31	23	13
STL	Store Local Word	9	31	23	13
UJP	Unconditional Jump	6	24	14	13
XJP	Case Statement	16	63	28	0

*Figure is an estimate.

REFERENCES

(1) LSI-11 WCS Users Guide, EK-KUV11-TM-001, Digital Equipment Corporation, Maynard, Mass. (1978).

(2) UCSD (Mini-Micro Computer) Pascal, Release Version I.4, January 1978, Institute for Information Systems, University of California at San Diego, La Jolla, Ca., Ken Bowles director.

(3) H.J. Curnow and B.A. Wichmann, "A Synthetic Benchmark", The Computer Journal, Vol 19, No 1, February 1976, pp 43-49.

(4) N. Wirth, Algorithms + Data Structures = Programs, Prentice Hall, New Jersey, 1976.

PASCAL/P-CODE CROSS COMPILER FOR THE LSI-11 *

Bruce L. Hitson
Stanford Linear Accelerator Center
Stanford, California

ABSTRACT

This paper describes the implementation of a cross compiler for Pascal that produces code that can be executed on an LSI-11 minicomputer. The approach taken is to first compile the source Pascal program (using an existing compiler) into an intermediate form known as P-Code. The P-Code is then cross compiled to LSI-11 assembly language. Once this has been achieved, the assembly language programs can be assembled using existing assemblers (such as MACRO-11) to produce relocatable load modules. These are linked together into an absolute load module and reformatted for transmission via serial line to the LSI-11. The details of the implementation are described. A comparison is also made between the approach taken in this implementation (cross compiling to the host machine's assembly code) and the approach where P-Code is interpreted directly.

INTRODUCTION

This paper describes the implementation of a cross compiler for the LSI-11 that converts Pascal pseudo-code (P-Code)[1] into assembly code that is suitable for processing by existing LSI-11 software. Unlike many other implementations of Pascal for minicomputers, this approach does not interpret P-Code, but instead produces LSI-11 assembly code [2] by cross compiling the P-Code statements that are output by the Pascal compiler. One of the goals of this approach is to generate code that will execute significantly faster than existing interpretive implementations without a severe increase in the total program size (program code plus runtime support routines). We would also like to allow programs written in Pascal and other languages such as Fortran, PL-11[3] and assembly language to be compiled separately and linked together into software packages that make use of the best features of each language. We are able to protect our large investment in existing software and yet be able to write new programs in a high-level language that should be easier to debug and maintain.

HARDWARE

The system is designed to be used on LSI-11 systems that have a minimal hardware configuration. A minimal system might consist of the LSI-11 processor, an EIA RS232 serial line interface for connection to the host computer, and a ROM kernel that can communicate with the host computer and download LSI-11 core images via the serial line from the host computer. In our case, the host computer is referred to as the TRIPLEX. It consists of two IBM 370/168s running OS/VS2 and a single IBM 360/91 running OS/MVT. All three processors operate under the ASP job management system. An alternative configuration could be based solely on an LSI-11 or other PDP-11 family computer with floppy drives and a large enough memory to execute the Pascal compiler. We have chosen to implement the first configuration for our current applications.

The systems in use at Stanford Linear Accelerator Center (SLAC) typically consist of an LSI-11 with serial line interface, 4K of ROM kernel routines, and 24K words of RAM. This allows execution of reasonably large software packages without having to be overly concerned about the efficient use of memory. Typical programs use only a fraction of the available memory for code storage, leaving the remainder for runtime stack and heap.

SOFTWARE

The software used to produce code that can be executed on the LSI-11 is, for the most part, written in Pascal. The only exceptions to this are the Pascal runtime routines which are written in assembly language for the sake of efficiency. The main programs used in the process of making an LSI-11 absolute load module are described briefly below.

1) Stanford Pascal compiler [4] - This is a highly modified version of the Zurich P2 compiler[5]. Modifications to produce P-Code that is also cross compiled into efficient IBM 370 code were done by Sassan Hazeghi of SLAC. This is the same P-Code that is cross compiled to LSI-11 code that eventually runs on the LSI-11.

* Work supported by the Department of Energy under contract number EY-76-C-03-0515.

2) P-Code Cross Compiler (PCC) - This is a 1500 line Pascal program that takes as its input P-Code produced by the Stanford Pascal compiler, and produces assembly code suitable for processing by standard LSI-11 assemblers such as MACRO-11. The detailed implementation of this program is the topic of the following sections.

3) Pascal runtime support - This collection of routines provides the standard procedures of the Pascal language (e.g., PUT, GET, EOF, etc.). It is currently written in assembly code for the sake of efficiency, and is in the process of being coded in Pascal.

4) SLAC LSI-11 Software [6] - This consists of implementations of programs such as MACRO-11 that run on the TRIPLEX and are used for assembling, linking, and loading LSI-11 code. This also includes routines for downloading programs via the serial line interface to remote LSI-11 systems.

All program development, compiling and linking is currently done on the TRIPLEX. The LSI-11 is simply downloaded from the TRIPLEX via the serial line and started executing at the beginning of the program that was loaded. Note that complex program systems may be loaded which may themselves consist of compilers, interpreters, etc.

IMPLEMENTATION DETAILS

Memory Organization

Memory is conceptually divided into three areas: Pascal monitor, program code, and runtime stack/heap. These are shown in Figure 1. The Pascal monitor performs the necessary initialization before entering the main Pascal program. It also does clean-up operations when the LSI-11 has finished executing and before control is returned to the TRIPLEX. The program code is the actual code for the routines of the Pascal program that is to be executed. The rest of the memory space is allocated to runtime stack and heap. The heap starts at the end of the program code and grows towards higher memory locations. The stack starts at the highest memory location and grows towards lower memory locations.

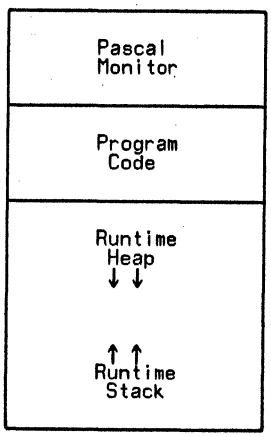


Figure 1 - Memory Organization

Data Types

Before discussing P-Code, it is useful to know the structure of the data that it will be referencing. There are six basic types of data: addresses (A), boolean (B), character (C), integer (I), real (R), and set (S). Boolean and character variables occupy one byte of storage each. Addresses and integer variables occupy one word (2 bytes) of storage each. Reals are represented in standard DEC floating point format, and occupy two words (4 bytes). Sets occupy four words (8 bytes), and can have up to 64 members. Alignment for each data type is provided for by the compiler according to the number of bytes it occupies. Thus, reals are aligned on 8 byte boundaries, while characters and booleans are aligned on single byte boundaries.

P-Code

P-Code is a pseudo-assembly code designed for a mythical stack computer (the P-machine[5]). There are two basic types of instructions: instructions that manipulate the top few items of the stack, and instructions that move data to and from "memory". The "memory" is actually part of the stack, and is accessed by specifying a pointer into the stack. A general P-Code instruction consists of four fields: OP, T, P, and Q.

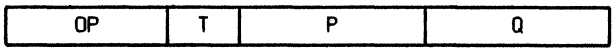


Figure 2 - P-Code Format

OP is a string of characters that specifies the operation to be performed. T is a single character that specifies the type of the operand to the instruction (e.g., I=integer, R=real). P and Q are used for a variety of purposes. They are most commonly used to specify a level and offset for instructions that load or store variables. The P-Code instruction set is described in the paper by Gilbert and Wall[1].

Referencing Variables

The P-Code produced by the Stanford Pascal compiler references variables by specifying two numbers: a level number and an offset. The level number specifies the lexical level of the variable being referenced. The scoping rules of the Pascal language require this to be interpreted as the lexical level of the most recently invoked procedure at the level specified. The offset specified is the number of bytes from the base of the specified lexical level where the variable being referenced is stored.

In order to make references to variables as quickly as possible, we would like to use the indexing capabilities of the LSI-11's general purpose registers. A number of registers, referred to as DISPLAY[1]..DISPLAY[*n*] are used to hold pointers to the base of the most recent activation of the lexical level (i.e., procedure or function) associated with *n*. To access a particular variable at lexical level *n*, we can use the indexed addressing mode of the LSI-11. Thus, to implement the P-Code instruction

LOD I *<level>*, *<offset>*

which loads an integer onto the stack, we can say

MOV -*<offset>*(DISPLAY[*<level>*]), -(SP).

A problem with this scheme is that we may want to access

variables in more lexical levels than there are registers to hold their base pointers. A solution to this problem involves keeping only the most commonly used DISPLAY registers in actual registers of the LSI-11. The remaining display registers are stored in memory, and loaded into registers only as they are needed. The concept of display registers has been discussed by Gries[7] and others.

As it turns out, the structure of many Pascal programs is such that most variables accessed are either local to the currently invoked procedure or are global variables (i.e., declared in the body of the program and *not* in a procedure or function). Taking advantage of this fact, only two registers are dedicated to holding DISPLAY register pointers. DISPLAY[1] is referred to symbolically as "GMP" (Global Memory Pointer), and DISPLAY[n] (where n is the level of the currently executing procedure) is referred to symbolically as "CMP" (Current Memory Pointer). References to variables in lexical levels other than those specified by GMP and CMP require that the value of DISPLAY[n] first be loaded into a temporary register which is then used for indexing.

In order to allow recursive procedures and functions, the value of CMP must be saved at each invocation. This process is described in the section on the Runtime Stack. The value of GMP need not be saved and is, in fact, fixed for the duration of a programs execution since a Pascal *program* (as opposed to a Pascal procedure or function) is not allowed to call another program at lexical level 1.

Runtime Stack

The format of the runtime stack is shown in Figure 3. Starting at the high end of memory, we have the stack frame for the main program. This consists of the return address to the Pascal monitor. Following this are five words of system variables, and three words of I/O buffer addresses. The I/O buffer locations contain pointers to buffers for up to six different devices. The default I/O device is the tty. The global variables are stored after the I/O buffer addresses.

When a procedure or function is invoked, a new stack frame is allocated. The first word of this stack frame is the return address to the procedure that invoked it. (In the case of the first procedure call, this will be the main procedure). The value of the CMP register must also be updated. This consists of 1) save the old value of DISPLAY[*<level>*] in the next stack location 2) load DISPLAY[*<level>*] with the current value of CMP 3) load CMP with a pointer to the return address that was pushed onto the stack in step 1...this is the base of the new stack frame.

The next four words on the stack are used to store the result of calls to routines that are functions. These four words are unused if the routine is a procedure. Local variables (variables declared in the level that we are now entering) appear next on the stack. The code for the routine whose stack frame was just created is now executed. At some random point in this routine, another procedure or function call may occur. If the call is to a function that is embedded in a calculation, some intermediate results of the calculation being done may be stored on the top of the stack. These are referred to in the diagram as temporary variables since they represent intermediate results. At this point, a new stack frame for the function being called is created, and execution proceeds as described above.

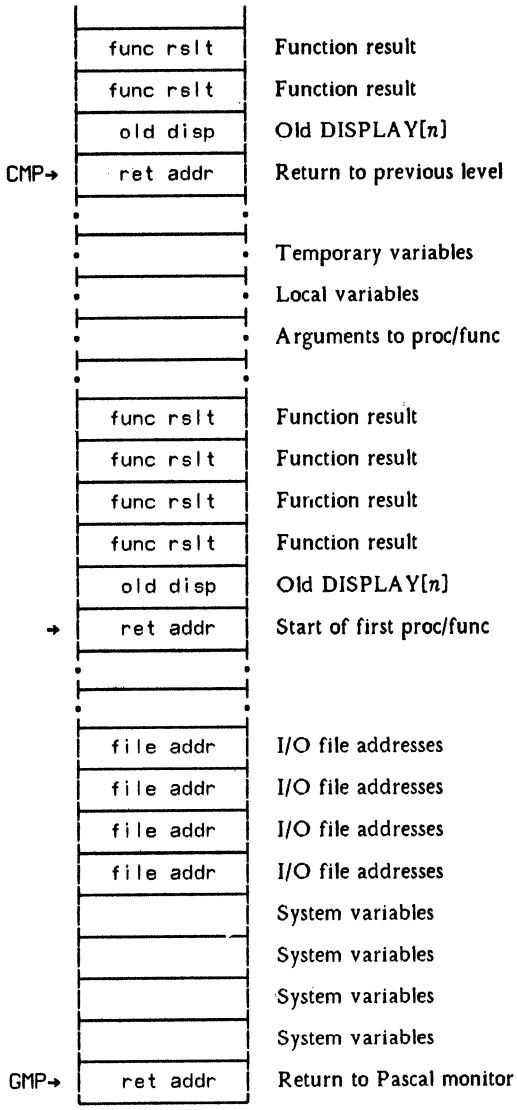


Figure 3 - Runtime Stack Format

OPTIMIZATION

At the present time, only a slight degree of optimization has been implemented. This manifests itself as not executing the standard system calls to do the initial reset/rewrite of the tty (since the tty is already initialized by the Pascal monitor). Also, routines to "start I/O (SIO)" and to "end I/O (EIO)" to the tty have been optimized out for the same reasons.

There are many places where the LSI-11 translation of a sequence of P-Code statements are relatively inefficient. Consider the Pascal statement "i:=i+1". If the variable "i" is an integer located at an offset of 14 in level 1 (the main program), the following P-Code might be produced:

```
LOD I 1,14      ;Get variable "i" onto stack
LDC I 1         ;Load the constant "1" onto stack
ADD I           ;(Top-1)←Top+(Top-1); Top:=Top-1 ;
STO I 1,14      ;Store the value back in "i"
```

The LSI-11 assembly code produced would be:

```
MOV -14(GMP),-(SP)    ;SRC=> LOD I 1,14
MOV #1,-(SP)          ;SRC=> LDC I 1
ADD (SP)+,(SP)        ;SRC=> ADD I
MOV (SP)+, -14(GMP)   ;SRC=> STO I 1,14
```

Obviously, this is not the optimum solution since the single LSI-11 assembly statement

```
INC -14(GMP)
```

would have had the desired result. The increment instruction takes two words of memory, whereas the sequence produced above takes seven words!! This inefficiency is due largely to the differing architectures of the register oriented LSI-11 and the stack oriented P-machine. Efforts are currently under way by various people to produce optimized P-Code[8] that would eliminate some of the more obvious inefficiencies. We are working on a much less extensive optimization of replacing the above *<load><load><operation><store>* operations with a more nearly optimal solution. This should have a fairly significant effect in reducing program size.

INTERFACE TO OTHER LANGUAGES

One of the main objectives in cross compiling Pascal to LSI-11 assembly code was to allow Pascal routines to be linked together with routines from other languages. In this way, our existing software library of Fortran, PL-11, and assembler routines can be used along with Pascal routines to produce significant software systems. Also, this allows the use of a language that is most appropriate for the problem at hand. As an example, it is fairly inefficient to deal with low-level concepts such as bit masking or trap handling through Pascal routines (although not impossible). These routines can be implemented in PL-11 or in assembly language and linked in with the Pascal routines to produce a usable software package.

PERFORMANCE EVALUATION

As was mentioned in the introduction, one of the main motivating factors for the cross compiling implementation of Pascal as opposed to the interpretive approach is the speed with which the code executes. Although extensive testing has not yet been completed, a preliminary comparison of the Pascal system produced using PCC (PASLSI) and two other systems has been made. The performance was also compared to DEC Fortran running under RT-11 on the LSI-11. A simple integer bubble sort was used as a benchmark. It is a fairly good example since it tests a combination of performance characteristics such as loop efficiency and array indexing efficiency. Execution time was tested for the sorting of 500 integers (arranged in reverse order so that every integer must be moved). Three Pascal systems were compared: PASLSI, UCSD Pascal[9], and Stanford Pascal. The results shown below are typical of the three systems from measurements made so far.

Execution time of bubble sort of 500 integers
(measured in seconds)

Stanford Pascal (IBM 370) - 0.6
DEC Fortran (LSI-11) - 22.0
PASLSI (LSI-11) - 84.0
UCSD Pascal (LSI-11) - 463.0

Based on the preliminary results above, we can make a couple of comments. PASLSI seems to have a significant speed advantage over UCSD Pascal (which uses an interpretive approach). Compared to Fortran, PASLSI runs about four times slower at the present. It should be kept in mind, however, that the DEC Fortran has been optimized to a significant extent, whereas the PASLSI system still has considerable room for improvement. With some of the optimizations mentioned under "Optimizations", it seems reasonable to assume that PASLSI in its final form will probably execute within a factor of two of DEC Fortran.

CONCLUSIONS

The cross compiling approach to making Pascal available on a minicomputer such as the LSI-11 is a useful addition to our existing software package of Fortran, PL-11, and assembly language routines. It allows us to use programs written in Pascal together with existing software written in other languages with only minor changes to the existing software. High level programs can be written quickly and cleanly in the block structured environment of Pascal. Low level routines can be written that perform critically time dependent tasks or that can more easily access the lower level constructs of the LSI-11 than Pascal. Thus, a particular task can be written in the language that is most nearly suited to the task (be it execution-time critical, memory usage critical, or software development and debugging time critical). The system is in a continual state of improvement and extension, and will no doubt have many new features added by the time this paper is printed.

ACKNOWLEDGEMENTS

I would like to thank Sassan Hazeghi of SLAC for his help with Stanford Pascal and for the many Pascal programs he has graciously shared with me. I especially want to thank Les Cottrell, whose initial encouragement got me started on this project, and whose continued support, help, and advice keeps me going.

REFERENCES

[1] Erik J. Gilbert and David W. Wall "P-Code Intermediate Assembler Language (PAIL-3)", Stanford Artificial Intelligence Laboratory, on-line documentation, March 1978.

[2] Digital Equipment Corporation, "Microcomputer Handbook 1976-1977", Maynard Massachusetts.

[3] Russell, Robert D. "PL-11: A Programming Language for the DEC PDP-11 Computer" European Organization for Nuclear Research (CERN), Geneva, 1974.

[4] Sassan Hazeghi, "Stanford Pascal Compiler" online documentation at SLAC.

[5] K.V. Nori, U. Ammann, K. Jensen, H. H. Nagel, "The PASCAL <P> Compiler: Implementation Notes", Berichte des Institutis fur Informatik, Zurich.

[6] R.L.A. Cottrell and C.A. Logg, "An IBM 370/360 Software Package for Developing Stand Alone LSI-11 Systems", Proceedings of the Digital Equipment Users Society, vol. 4, no. 4, pp 985/991, April, 1978.

[7] Gries, David, "Compiler Construction for Digital Computers", John Wiley & Sons, N.Y. 1971.

[8] Sites, Richard L. "Progress Report, June 1978: Machine-Independent Code Optimization", Department of Applied Physics and Information Science, University of California, San Diego.

[9] UCSD (Mini-Micro Computer) PASCAL Documentation, Release Version 1.4, January 1978.

BLISS COMPILER OPTIMIZATION TECHNIQUES

Alan P. Lehotsky
Digital Equipment Corporation
Maynard, Massachusetts

ABSTRACT

The family of BLISS compilers developed by DEC produces highly optimized object code. The formal and heuristic techniques used by these compilers are reviewed, with specific examples drawn from each compiler: Bliss-16, Bliss-32, and Bliss-36.

INTRODUCTION

The language family categorized as Common Bliss has been under development by DEC for three years. Figure 1 is a brief history of that development.

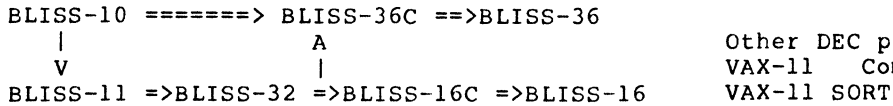


Figure 1 - BLISS Family History

The current languages and their implementations are direct descendants of Bliss-11. [1]

Digital has a commitment to Bliss as the preferred implementation language for all new software developed within Central Engineering. This commitment is justified by:

- Reduced life-cycle costs. (Fewer bugs, faster implementation, easier maintenance)
- Code quality approaching that produced by a talented assembly language programmer.
- The opportunity to transport software and software engineers among architectures. [2]

The Common Bliss family presently consists of three members: Bliss-32, Bliss-36 and Bliss-16. Approximately 50% of each compiler is common source code. All three are written in Bliss and share a common design. The TOPS-10 versions of Bliss-36 and Bliss-16 have the only non-Bliss code in the full compiler set. It is a six page MACRO-10 routine used to interface to SCAN, and was written as part of the original bootstrap process. Bliss-36 has taken about 1 man-year to bootstrap from the Bliss-32 sources and reach a usable level of functionality. It is expected that Bliss-16 will require a similar effort. Currently, only Bliss-32 is available for purchase by customers. Bliss-36 and Bliss-16 are not available at this time.

Other DEC products written in Bliss include VAX-11 Common Runtime Library, Linker, VAX-11 SORT, DEBUG and RMS-32 ISAM. Ongoing (and unannounced) development includes DECNET utilities and at least four high-level languages.

Our optimization strategy generally favors space reduction over speed reduction. The justification is expressed in part by the following:

"Our philosophy is that code size is a more important consideration since we cannot predict which portions of a program will contribute significantly to the execution time, but that space costs something whether or not the code occupying it is ever executed."[1]

In addition, smaller code is faster due to fewer instruction fetches. On paging machines, smaller code reduces the working set and provides better paging behavior.

COMPILER OVERVIEW

This paper divides the compiler into 3 primary phases, based on their optimization strategies and goals. These phases are:

Common Front End - Constant folding, data-flow analysis and common sub-expression recognition. There is a fairly rigorous theory behind the strategies and algorithms used. Most optimizations are essentially transformations of the source-program after macro and structure expansion.

Target Specific Transformations - Knowledge of the target-machine architecture is used to transform the internal representation of the program into a more optimal program. "Optimal" is usually measured in the number of instructions and data-references required to implement the program's semantics.

Final Optimization - More traditional optimizations, including peepholes, redundant instruction elimination, jump-branch resolution and adjacent instruction merging.

The compiler works on a single routine at a time. The internal representation of the routine is a tree, almost identical to a parse-tree (but without any declarations). Figure 2 shows a single routine declaration for our old friend "N!".

```
ROUTINE factorial(n)=
    BEGIN
    IF .n EQL 0
    THEN
        1
    ELSE
        .n * factorial(.n-1)
    END;
```

Figure 2 - Sample routine

Figure 3 shows the tree built for the routine FACTORIAL.

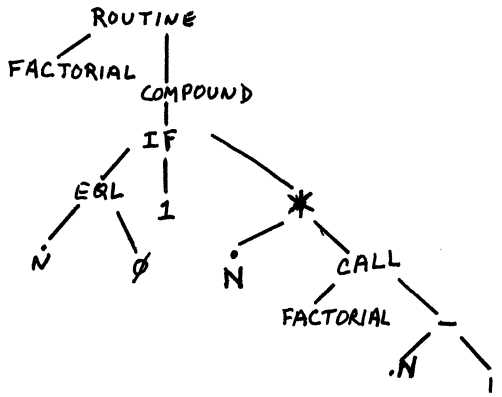


Figure 3 - Internal tree

Each phase does a top-down, left-to-right treewalk. As each node in the tree is encountered, node-specific routines are invoked.

COMMON FRONT END

Optimizations performed here are primarily target machine independent.

Constant Folding (KFOLD)

Constant folding is an exception to target independence in that Bliss-16 runs only as a cross-compiler under VAX/VMS, TOPS-10 or TOPS-20. Host architecture differences with regard to arithmetic processing can be significant unless care is taken. Therefore, constant folding is carefully tailored to the target-machine's arithmetic precision.

Constant folding is based on the axioms of arithmetic. Besides folding expressions like

```
1^6     =>     64
```

the folding sub-phase performs more esoteric optimizations. For example, Figure 4 shows a LITERAL declaration and a SELECTONE expression. The SELECTONE expression compares its SELECTOR with each of the SELECT-LABELS, when the SELECTOR and SELECT-LABEL are equal the SELECT-ACTION is executed.

```
LITERAL
    white=0,
    red=1,
    blue=2,
    green=3,
    purple=4;
....
....
SELECTONE .color OF
    SET
    [red,white,blue]: us_flag;
    [green]:          grass;
    [purple]:         cows
    TES;
```

Figure 4 - SELECTONE Expression

The range of expressions "[red,white,blue]" in the SELECT-LABEL is transformed by KFOLD into a single expression "[white TO blue]", which only requires a single unsigned comparison equivalent to

```
(.color LEQU blue)
```

as it takes advantage of the fact that 0 is the smallest unsigned number.

Flow Analysis (FLOWAN)

Flow analysis has the largest body of formal theory behind it.

Common Sub-Expressions - Two expressions E1 (the creation) and E2 (the use) are Common Sub-Expressions (CSE's) if

1. The values returned by E1 and E2 are always identical.

2. Program control flow is such that whenever E2 is evaluated, E1 has been evaluated.

The FLOWAN subphase detects potential CSEs. Because of the target architecture, some expressions may be easier to recompute than to save in temporaries. This can be better determined in target-dependent phases, so the

identification of 'feasible' CSE's is done later.

There are essentially two types of CSE's recognized, REAL and BOGUS. REAL CSEs have a single creation, followed by one or more uses. Figure 5 shows a set of expressions involving ".x+1", some of which are CSEs.

```
BEGIN
...
...
IF .x+1 GTR 0        ! E1
THEN
    BEGIN
    ...
    a=.b[.x+1];      ! E2
    ....
    END;
...
c = .x+1;    ! E3
...
x = .d;
...
f = .x+1;    ! E4
...
...
END
```

Figure 5 - CSE Candidates

E1 and E2 are CSE's, E1 and E3 are CSE's. But, E2 and E3 are not CSE's because E2's evaluation is not guaranteed to precede E3's evaluation. E1 and E4 are not CSE's because the values returned are not always identical.

BOGUS CSEs have creations on every branch of conditional-execution flow and one or more uses. Figure 6 is an example of a BOGUS CSE "(2*.a)".

```
IF .x
THEN
    BEGIN
    a = .x;
    factor = (2*.a)
    END
ELSE
    BEGIN
    a = -.x;
    root = (2*.a)
    END;
sum = .factor + (2*.a);
```

Figure 6 - BOGUS CSE Candidate

The equivalent transformed expression is shown in Figure 7.

```
IF .x
THEN
    BEGIN
    a = .x;
    T1 = factor = (2*.a)
    END
ELSE
    BEGIN
    a = -.x;
    T1 = factor = (2*.a)
    END;
sum = .factor + .T1;
```

Figure 7 - BOGUS Equivalent

Reference [1] provides a good discussion of the algorithms and techniques used in CSE recognition.

Code Motion - Another major optimization in FLOWAN is the recognition of code-motion opportunities. Expressions which are independent of surrounding expressions may be moved to a place where they are either easier to compute, are outside of loops, or are evaluated once rather than in all branches of a conditional expression such as IF or CASE. There are four fundamental types of code motion: ALPHA, OMEGA, RHO and CHI.

ALPHA and OMEGA apply to IF-THEN-ELSE and CASE expressions. Constituent expressions are moved 'backwards' (ALPHA) or 'forwards' (OMEGA) when their evaluation is independent. Figure 8 shows an expression amenable to ALPHA and OMEGA motion, and Figure 9 shows the equivalent transformed expressions.

```
IF .x
THEN
    (y = .a + .b; x = 5)
ELSE
    (z = .a + .b; x = 5);
```

Figure 8 - ALPHA-OMEGA Candidate

```
T1 = .a + .b;
IF .x
THEN
    y = .T1
ELSE
    z = .T1;
x = 5;
```

Figure 9 - ALPHA-OMEGA Equivalent

Notice that the BOGUS CSE in Figures 6-7 couldn't be ALPHA-motioned because the evaluation of "(2*.a)" wasn't independent of the preceding assignments "a = ...".

CHI-motion removes loop-invariant expressions from the body of a loop, to avoid re-

computing them during each loop-traversal. Figure 10 shows a CHI motion candidate in the initialization of an MxN matrix.

```
INCR i from 0 to N-1 DO
INCR j from 0 to M-1 DO
    (a + (.i*N) + .j) = 0;
```

Figure 10 - CHI Candidate

The expression ".i*N" is invariant to the inner loop, and will be transformed by FLOWAN into Figure 11:

```
INCR i from 0 to N-1 DO
    BEGIN
    t1 = .i*N;
    INCR j from 0 to M-1 DO
    (a + .t1 + .j) = 0
    END;
```

Figure 11 - CHI Equivalent

which eliminates M(N-1) multiply instructions at execution time.

RHO motion is more complex to describe. Figure 12 shows a typical RHO-situation. The expression ".x+.y" is created inside the loop, invalidated by a store into one of its constituent expressions ("x = .c") and recomputed again. RHO motion optimizes by moving the first creation outside the loop and recomputing the second creation into the same temporary.

```
WHILE .c DO
  BEGIN
  a = (.x+.y) * 5;
  x = .c;
  ....
  ....
  c = (.x+.y) * 5
  END;
```

Figure 12 - RHO candidate

Figure 13 shows that an add and a multiply have been moved out of the loop by RHO motion.

```
T1 = (.x+.y) * 5;
WHILE .c DO
  BEGIN
  a = .T1;
  x = .c;
  ....
  ....
  T1 = (.x+.y) * 5;
  c = .T1
  END;
```

Figure 13 - RHO equivalent

The code motions shown here seem trivial, but data-structure references and macro-expansions can produce code which profits from these optimizations.

TARGET SPECIFIC TRANSFORMATIONS

Target-specific transformations continue to use the internal tree to detect and apply optimizations. There are several subphases of interest, including:

DELAY - Determines the general 'shape' of the code based on the target architecture. The set of feasible CSEs is selected from the potential CSEs identified by FLOWAN. The name 'DELAY' refers to the strategy of delaying the computation of a low-level node in the tree until an upper level in the tree.

TNBIND - Measures the lifetimes of user-declared dynamic storage and compiler temporaries. These 'variables' are assigned to stack and register locations according to their needs.

CODE - Generates the instruction sequences required to implement a single node in the tree.

DELAY Subphase

The DELAY subphase propagates information down the tree describing how the sub-tree will be used in an expression. The kind of use information includes

RESULT - Either REAL or FLOW, indicating whether this sub-expression will be needed as a real value, or only to determine conditional flow. For example, the expression "(.x EQL 0)" is REAL when used in

```
y = (.x EQL 0);
```

and FLOW when used in

```
y = (IF .x EQL 0 THEN 1 ELSE 0);
```

Additionally, it is possible for an expression to be both REAL and FLOW. If ".x EQL 0" is a CSE, it will be both.

CONTEXT - OPERAND context indicates that the result of an expression is needed for itself, while ADDRESS context implies that the expression is used only to address a memory location. Thus the expression ".A+4" is ADDRESS when used in the expression

```
(.a+4) = 0   ! CLRL 4(a)
```

which allows the compiler to use addressing modes to evaluate the expression, assuming that "a" is in a register. But it is OPERAND when it appears in the expression

```
y = .a+4   ! ADDL3 #4,a,y
```

and an add instruction is needed to obtain a physical result.

MUST_BE_REG - Result must be in register. This is used by Bliss-16 when compiling for a PDP-11 with EIS. One of the operands of the multiply must be in a register.

COND_CODE - Result will only be used for its condition code setting. This is used on VAX by the CASE expression when its selector is the SIGN builtin-function or the CH$COMPARE string comparison function. Figure 14-16 show the CASE expression, the resulting code without and with this optimization. This saves 21 bytes in the example shown!

```
CASE SIGN(.x) FROM -1 TO 1 OF
    SET
    [-1]:     .....
    [0]:      .....
    [1]:      .....
    TES;
```

Figure 14 - CASE Expression

```
    TSTL   X
    MOVPSL R0
    EXTV   #2,#2,R0,R0
    SUBL3  R0,#1,R0
    CASEL  R0,#-1,#1
1$: .WORD  2$-1$
    .WORD  3$-1$
    .WORD  4$-1$
```

Figure 15 - Non-optimized VAX Code

```
    TSTL  X
    BLSS  2$
    BEQL  3$
4$: ...
```

Figure 16 - Optimized VAX Code

TNBIND Subphase

This subphase is primarily target independent. The few differences for each compiler are mainly a result of quirks in the target hardware. This is especially true of the character manipulation instructions. VAX potentially uses R0-R5 in string instructions, the KL-10 requires any 5 contiguous accumulators, and the PDP-11 normally uses software simulation via routine calls.

The primary activities are

1. Temporary assignment
2. Lifetime determination
3. Priority ranking
4. Packing temporaries into available stack and register resources.

The primary goal of TNBIND is to minimize the dynamic space requirements. A single register may be used for declared local storage and compiler temporaries having disjoint lifetimes.

I choose to gloss over this subphase, because it is very complex and its payoff is subtle.

CODE Subphase

CODE generates the locally optimal sequence of machine-instructions based on information provided by DELAY and TNBIND. No consideration of the surrounding instruction environment is made at this time. Thus garbage such as

```
    BRB   10$
10$: ....

    MOVL  R0, RSLT
    MOVL  RSLT, R0
```

can be generated by adjacent nodes in the tree. However, within a single node, the generated code will be nearly optimal. For example each Bliss compiler performs considerable analysis on field move generation. Expressions such as

```
DST = .SRC<p,s>;
```

where 'p' and 's' are literals are very common. Bliss-32 and Bliss-36 have an easier job, as they never need more than a single instruction (EXTV or LDB) to fetch a field. Bliss-16 however may require 4 or more instructions to mask and shift the field into position.

On the PDP-10 full use is made of immediate mode when generating literal values. All forms of MOVxI and HxyzI are used.

FINAL PHASE

The final phase is ad-hoc, but has a high payoff. A number of optimizations which would be impossible in earlier phases are done here, once instruction sequences and operand locations are known. Some techniques are common to all implementations.

Cross-Jumping

Cross-jumping is somewhat similar to OMEGA-motion in FLOWAN. Identical code sequences which branch to the same merge point are adjusted to become part of the merge point. For example:

```
IF (.x eql 0) THEN x=.y ELSE x=.w;
```

where x,y and w are memory locations would

generate the DECsystem-10 sequence

```
      SKIPN   T1,X
      JRST    L$1
      MOVE    T1,Y
      MOVEM   T1,X
      JRST    L$2
L$1:  MOVE    T1,W
      MOVEM   T1,X
L$2:  ...
```

which would be cross-jumped to result in

```
      SKIPN   T1,X
      JRST    L$1
      MOVE    T1,Y
      JRST    L$2
L$1:  MOVE    T1,W
L$2:  MOVEM   T1,X
```

saving a single word.

Peepholes

The cross-jumping example shows another optimization possibility. The DECsystem-10 SKIPx can replace a MOVE JUMP sequence in some situations. Thus FINAL will reduce the previous code sequence to:

```
      SKIPE   T1,X
      SKIPA   T1,W
      MOVE    T1,Y
      MOVEM   T1,X
```

Many of FINAL's optimizations are synergistic. Eliminating a single instruction can result in many additional optimizations.

Literal generation is a very powerful peephole optimization. The VAX-11 has an extensive and complex set of instructions and addressing modes which can be used to materialize literal values. For example, the hardware allows literal operands between 0 and 63 to be represented by a single byte. Thus MOVL and MNEGL allow -63 to 63 to be easily generated. The Bliss-32 compiler also knows that

```
MCOML #63,dst
```

can be used to generate -64!

In Bliss-32, a major optimization is the merging of adjacent field references. For example

```
MOVL R0,R2
MOVL R1,R3
```

will end up as

```
MOVQ R0,R2
```

Similar optimizations are performed for field insertions, such as

```
BICW2  #^XFF01, A
BICB2    #2, A+3
```

will be merged to

```
BICL2  #^X200FF01, A
```

Jump-Branch Resolution

One of the few formally definable parts of FINAL is the jump-branch resolution done for Bliss-32 and Bliss-16. Both machines support variable-length branching - the PDP-11 having 2 or 4 byte instructions, while the VAX has 2, 3 or 6 byte instructions. On both machines, the conditional branches are only 2 bytes long. The compiler can always produce the 'best' sequence of JMP/BR instructions required to reach the target label.

This is implemented by building a flow-graph of the machine-language and reducing that graph by collapsing sub-graphs of known size.

CONCLUSION

The Bliss compiler family uses a variety of algorithms, strategies, heuristics and 'tricks' to produce optimized object code. When measured against other optimizing compilers on suitable problems, Bliss generates code that is typically 30% smaller.

Table 1 compares the Bliss family against FORTRAN on several architectures. The test case is an iterative solution to the Towers of HANOI originally written in FORTRAN. No special effort was made to redesign the algorithm in order to optimize for a particular machine or compiler. In fact, the Bliss, FORTRAN and PASCAL versions were compiled from single sources for each language. Appendix A and B show the sources for Bliss and FORTRAN respectively. Appendix C shows a VAX-11 Macro version which was tweaked as much as possible.

Language	Instructions
Bliss-32	85 bytes
VAX Fortran	126 bytes
VAX-11 Macro	81 bytes
Bliss-36	27 words
FORTRAN-10	44 words
PASCAL(note 1)	78 words
MACRO-10	25 words
SAIL	64 words
Bliss-16	100 bytes
Fortran-4 Plus	194 bytes
C	142 bytes
RSX-11 FORTRAN	248 bytes
PASCAL(note 2)	342 bytes

Table 1 - Code Quality Comparisons

Note 1 - Hedrick Souped-Up PASCAL for DECsystem-20. No run time checking.

Note 2 - DECUS PASCAL V5. No run-time checking.

In order to achieve the optimization goals, Bliss uses tricks which are "flogging offenses" for a programmer to even contemplate!

BIBLIOGRAPHY

[1] Wulf, Johnsson, Weinstock, Hobbs and Geschke, The Design of an Optimizing Compiler, American Elsevier, New York, 1975.

[2] Marks, Maurice, "Transportable Programming in Bliss", Proceedings of the Digital Equipment Computer Users Society, Vol. 4, No. 4 1978.

[3] Brender, "A Survey of Bliss-16, Bliss-32 and Bliss-36", Proceedings of the Digital Equipment Computer Users Society, Vol. 4, No. 4

APPENDIX A - BLISS Example

```
MODULE TOWER(MAIN=HANOI)=
BEGIN
OWN
  NO : VECTOR[20],
  NF : VECTOR[20],
  NT : VECTOR[20];

GLOBAL ROUTINE HANOI : NOVALUE =
  BEGIN
  LOCAL
    N,I,K,JP;

  I=1;
  K=2;
  JP=0;
  N=19;
  WHILE 1 DO
    BEGIN
    IF .N EQL 0
    THEN
      BEGIN
      DO
    (IF (JP = .JP-1) LSS 0 THEN RETURN)
      WHILE
    (N = .NO[.JP]) LEQ 0;
      I = .NF[.JP];
      K = .NT[.JP];
      NO[.JP] = -.N;
      I = 6 - .I - .K;
      END
    ELSE
      BEGIN
      NO[.JP] = .N;
      NF[.JP] = .I;
      NT[.JP] = .K;
      K = 6 - .I - .K;
      END;
    JP = .JP + 1;
    N = .N - 1;
    END;
  END;

END ELUDOM
```

APPENDIX B - FORTRAN Example

```
C   FORTRAN BENCHMARK - HANOI
C
    DIMENSION NO(20),NF(20),NT(20)
    I=1
    K=2
    N=20
C
    JP=1
3   IF (N-1) 5,7,5
5   NO(JP)=N
    NF(JP)=I
    NT(JP)=K
    K=6-I-K
6   N=N-1
    JP=JP+1
    GO TO 3
7   JP=JP-1
    IF(JP) 8,777,8
8   N=NO(JP)
    IF (N) 7,7,9
9   I=NF(JP)
    K=NT(JP)
    NO(JP) = -N
    I=6-I-K
    GO TO 6
777 CONTINUE
    STOP
    END
```

APPENDIX C - VAX MACRO Example

```
    .TITLE  Tower of Hanoi

    .PSECT  $own$,noexe,2

NT: .BLKL   20
NO: .BLKL   20
NF: .BLKL   20

    .PSECT  $code$,nowrt,2
    .ENTRY  HANOI, ^M<R2,R3,R4>
    MOVAB   W^NO, R4
    MOVL    #1, R1  ;I
    MOVL    #2, R2  ;K
    CLRL    R0      ;JP
    MOVL    #19, R3 ;N
1$: BNEQ    3$
2$: DECL    R0
    BEQL    5$
    MOVL    (R4)[R0], R3
    BLEQ    2$
    MOVL    80(R4)[R0], R1
    MOVL    160(R4)[R0], R2
    MNEGL   R3, (R4)[R0]
    PUSHAB  -6(R2)[R1]
    MNEGL   (SP)+,R1
    BRB     4$
3$: MOVL    R3, (R4)[R0]
    MOVL    R1, 80(R4)[R0]
    MOVL    R2, -80(R4)[R0]
    PUSHAB  -6(R2)[R1]
    MNEGL   (SP)+, R2
4$: INCL    R0
    DECL    R3
    BRB     1$
5$: RET
    .END
```

ACCURATE DESCRIPTION OF SYSTEM STRUCTURE - A NEW STANDARD FOR LANGUAGE QUALITY

Edward S. Lowry
Digital Equipment Corporation
Maynard Massachusetts

ABSTRACT

DAWN is a generally extensible programming language wich provides adaptable, declarative application programs. Programs and system descriptions written in DAWN accurately describe not only the function of information systems but also their structure. Current languages (particularly ones like APL and LISP) describe function but extensively omit, distort, and bury information about system structure. Deficient language has seriously impeded progress in information science, which (like any science) is completely dependent on accurate, detailed descriptions. Progress in software technology requires the following improvements, which depend on accurate structural description:

- elimination of representational irrelevancies,
- elimination of irrelevant distinctions between data objects and system objects,
- reduction of need for special purpose languages,
- natural ways of referencing sets of objects,
- non-procedural programming style,
- assistance to the user through automated analysis of programs for greater informality, adaptability, reliability, and performance.

Avoiding representational irrelevancies while keeping the language small requires a careful choice of primitive operand types. It is argued that the nodes and arcs of directed graphs are the best choice. Additional technical dependencies among the above steps, and the ways that DAWN facilitates taking them, are discussed.

This paper proposes principles of a general approach to making improvements in ease of use of computers, and briefly describes a specific design for a language called DAWN based on those principles. DAWN applications are very free from computer oriented irrelevancies and are very declarative and English-like in style, especially for business applications. The most immediate usefulness of DAWN is expected to result from providing business data processing customers with application packages which they can easily understand and adapt to their needs. The capabitities of Dawn will also be helpful in coping with the diversity of data structures and system objects which will exist in computer networks.

DAWN EXAMPLE

An illustration of the style of Dawn is given in Figure 1. The program accepts orders, groups them by customer, and produces an invoice for each such customer. It calculates extensions, totals, taxable totals, and discounts for each invoice. It is invoked as orders are created. The example is taken from one used to illustrate the BDL language [1].

An important feature is the lack of procedural statements. The relationship between the input orders and the invoices produced is expressed declaratively.

The first statement creates a context for the program and data to reside in and gives it a name: billing.

The next statement defines the structure of a set of objects of type customer each of which is uniquely identified by an integer. The "key" attribute asserts its uniqueness. Each customer has a name and address which are character strings. Customers are also categorized as classA, classB, or classC.

The next statement defines the structure of a set of items in a sales catalog. Each item is identified by an integer. Each item is defined to have a unit price in dollars and an indication of whether it is taxable.

The next statement defines the structure of a set of orders which are created in the context. Each order is related to a customer, and contains a list of item_lines indicating the kind and number of items ordered. The word "holds" indicates that

DAWN BILLING EXAMPLE

```
declare billing context;

declare customer set  /* customer file
  has integer key
  has name in string
  has address in string
  is one of (classA classB classC);

declare item set      /* sales catalog
  has integer key
  has price in dollar
  maybe taxable;

declare order set          /* input file
  has customer
  holds set item_line
  has invoice
      := findorcreate invoice(its customer);
   /* creates invoice for customer if needed

declare item_line sets
      /* contained in orders and invoices
  has item
  has count
  has amount :=the count*price of the item;

declare invoice set        /* output file
  has customer key
  has set order converse
        /* the orders having this invoice
  holds  set  item_line  :=  copy  every
            item_line of every order of it
  has total
     := sum amount of every item_line of it
  has total_taxable := sum amount of every
     item_line of it whose item is taxable
  has discount := the total * (.10 if its
     customer is classA else .05 if
     its customer is classB else 0 )
  has total_due := the total - the discount;

end;;
```

Figure 1. Dawn Billing Example

the item_lines are an integral part of the order (unlike the customer). On creation of each order, it is related to an invoice which has the same customer. If no such invoice exists one is created as a result of the "findorcreate" function.

The next statement indicates that item_lines exist in lists which are contained by other objects. Each item_line is related to an item, a number of them being purchased, and a dollar value of the purchased items which is computed as shown.

The next statement defines the structure of the set of invoices produced. At most one invoice exists at any time for a customer to which the invoice is related. The invoice is automatically related back to the set of orders which are related to the invoice. That is specified by the "converse" attribute. A list of item_lines are implicitly created as part of the invoice by the expression "copy every item_line of every order of it". This is equivalent to a doubly nested loop of statements in other languages. A total, total_taxable, discount, and total_due are defined as relations in the invoice, and computed automatically according to the expressions given.

Any time an order is created or altered the invoices are altered to reflect the changes. Given documentation in English (more detailed than the above) plus on-line explanatory aids it is hoped that persons with little computer training will be able to understand the meaning of such programs. It is also hoped that such people who understand the needs of their small business information systems will be able, with interactive computer assistance, to adapt prepackaged application programs written in similar style to their specific needs.

Such packages would include functions for printing things like formatted invoices and orders. The interactive facilities of Dawn provide for the creation and alteration of such functions with little reading and no writing of Dawn expressions. Interactive input of orders would also be perfomed in a menu driven style. The effectiveness of the interactive facilities is enhanced by exploiting the accurate descriptions of the data structures given by the declarations.

At present only a query subset of Dawn has been implemented, but a more complete implementation is planned.

While many features of Dawn are matters of subjective design, the following section argues that any language having as much ease of use will share a number of its underlying characteristics, which are different from those of current languages.

TECHNICAL DEPENDENCIES

There are reasons why technical capabilities which improve ease of use of computers are interdependent, and must be built on top of each other in a fairly definite order as illustrated in Figure 2.

1. Programs adaptable by unskilled users.
Except for a minority of situations where computer applications can be installed using only paramaterisation by users, someone who understands the needs of the installation must understand and modify the code. When the user is relatively unskilled the programs must be much more readable (2) than those in current languages. Successful modification of a program by an unskilled user also depends on effective analysis (9) of both the program and the user's inputs, so that inappropriate inputs can be corrected and queried and the user can be shown the effects of the changes.

2. Readable Programs

For programs and system desciptions to be highly readable to humans it is necessary to make a major change from procedural to declarative styles (3) of presentation. Some technical information is best presented procedurally, but it is most often presented to people packaged in small relatively independent declarative statements rather than the large networks of interdependent and uninformative imperatives used in current progamming. Readability may also be enhanced by automated analysis of progams (9) which can present the progams in a variety of styles and with varying amounts of explanation in direct response to user requests. Readability also depends on eliminating computer oriented irrelevancies (6) as shown indirectly in figure 2.

3. Declarative style

Declarative statements about systems (this sentence, for example) almost always include references to whole sets of objects (4) from their problem domains. Current major languages (Cobol, Fortran, PL/I, BASIC, RPG) do not have set references beyond rudimentary ones such as those in character string operations.

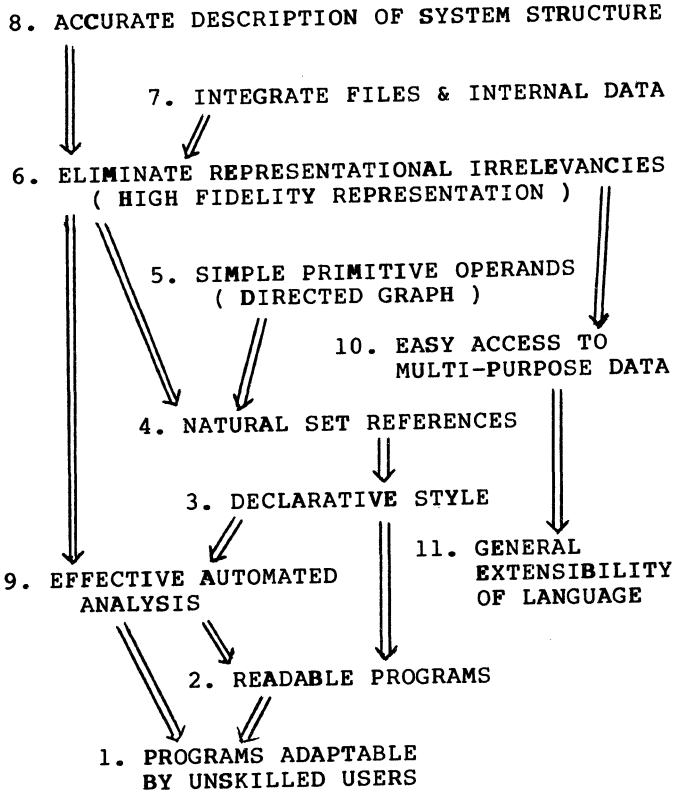


Figure 2. Dependencies for ease of use

For business programming most proceduralness can be eliminated and what remains will tend to be intuitively understandable. Computations are triggered mainly by the occurence of patterns which are recognized in the data structure. Responses to the recognition of patterns will in turn create patterns which are recognized and responded to, creating ripple effects through the system. While business systems are structurally rich, the patterns which get recognized are fairly simple ones, like crossing of numeric threshholds, and changes of set membership between empty and non-empty.

4. Natural set references

The effect of a set reference is obtained in current languages by iterating through a loop of imperative statements. Instead of being contained within a single statement a set reference is distributed over several imperative statements which contain irrelevant information about how the members are represented and accessed. To package the needed variety of set references within individual statements it is a practical necessity that representational and accessing irrelevancies be minimized (6). A few languages do permit set references, with varying degrees of generality. These include APL, LISP, SETL, and various data base query languages. A consistent feature of such languages is that their kinds of primitive operand are simple and few in number. The question then arises as to whether that is a necessary feature of a language with general set references. DAWN has over 20 distinctly different kinds of set reference plus many functions which construct sets. The reasons for including them seem adequate to ensure that anything much simpler would soon be extended to include comparable constructions. If they were defined for anything other than a small number of simple primitive operands it seems very likely that the language would be needlessly complex (5).

5. Simple primitive operands

The need to simultaneously have a few simple primitive operand types and avoid representational irrelevancies imposes severe constraints on the acceptable choice of primitive operand types. Reasons are given below why the nodes and arcs of directed graphs are probably optimal.

6. Eliminate representational irrelevancies

Systems are composed of objects, relationships between them, and processes that transform them. They are not, in general made of: variables, records, fields, bits, addresses, integer indexed arrays, etc. The latter are examples of representational irrelevancies introduced to encode models of information systems in current computer languages. Any language provides a data model comprised of the primitive data structures in which states of the information system must be represented. Mismatches between the structure of the information system (or problem environment)

and the structures provided by the data model force the user to introduce representational conventions mapping his information structures onto the available data structures. The user understands the conventions (at least temporarily) but current languages provide no way to inform the machine about them. Strong typing like that of Pascal provides a way to formalize some information about the representation conventions. The encapsulations of Clu and Alphard provide additional information by restricting the operations which can be performed on the data structures.

A Dawn reference such as:

every employee of every department where population of its location > 100000

is not expressible in current major languages (nor newer ones like Pascal, Clu, or Alphard) without conspicuous inclusion of representational irrelevancies. The ability to interpret such a reference is dependent on information about the structure of the information system. The system must be aware of the existence of 1 to N employee relationships between departments and other objects (presumably people). It should also be aware that locations are defined for departments but not employees. This illustrates the need for accurate, detailed description of the structure of information systems (8) to avoid representational irrelevancies.

The dependency may also be expressed by noting that complete information about data structures is always needed for successful execution. Any incompleteness in the description of the information structures implies some difference between the information structures and the data structures and hence some irrelevancies will be introduced in representing one by the other.

Eliminating representational irrelevancies (or providing high fidelity representation) also requires that any multiplicity of ways of introducing representational irrelevancies be avoided. To do so requires that structural and referencing distinctions between files and internal data be eliminated (7).

7. Integration of files and internal data

Elimination of structural and referencing distinctions between files and internal data is prerequisite to most of the above steps and implies a major departure from most current languages. Cobol, for example, is extensively designed around that distinction.

8. Accurate description of system structure

Systems have structure and function. Programming languages have tended to emphasize detailed precise description of the function of systems while omitting, distorting, and burying information about their structure. The data declarations of Algol-like languages give some information about the structure of the problem environment, but it is very limited and distorted by the procrustean data structures available. Many features of the information structure are deducible only by detailed study of what the programs do and do not do. This is particularly true with APL programs.

For systems to be comprehensible and effective their structure and function should be well suited to each other. Mismatches between structure and function lead to irrelevant complications of both structure and function. Since the days of octal coding newer languages have tended to give steadily more emphasis to description of system structure. Heterogeneous records and strong typing are examples of improvements. A number of data base data models have since gone further in providing structural description but not within the framework of a strong programming language. The DAWN language continues this trend, and may carry it close to a natural conclusion. At present the information sciences remain very deficient in the most elementary requirement for any kind of scientific activity - tools for accurate description of the relevant phenomena.

Though any kind of system must have a structure, the structure of information systems is a somewhat subtle idea. Brains and computers tend to adapt very well to a wide range of system functions with no externally obvious effects on their structure at all. However, complex information systems usually include models of physical systems whose structure is relatively well understood. The representational irrelevancies, and failures to accurately represent system structure become apparent when examining such models of physical systems as represented in current languages.

The main feature of DAWN which enables it to improve ease of use is that it permits a very complete description of both the structure and the function of a broad class of systems using a declarative style with a high level of representational fidelity.

9. Effective automated analysis

Improvements in application development for the forseeable future will depend strongly on making the computer a more effective, active partner in all phases of the application development process. This depends on the computer performing automated, or semi-automated analyses of computer programs and other user inputs. For many years automated program analysis has been a swamp where well motivated efforts have bogged down mainly in representational irrelevancies. Some progress has been made, mainly in efficient compilation. Performing effective program analysis based on current languages involves the difficult technical problems of translating internally from low level to higher level language with better

representational fidelity. That would lead to a degrading man-machine relationship in which the human must write his program in relatively machine oriented terms while the machine analyzes the program in more human terms.

Any analysis activity involves abstracting away information which is irrelevant to a particular purpose. A sensible way to start is by abstracting out information which is not relevant to any of the user's purposes but imposed by the tools he is using. The best way to do that is to refrain from including the irrelevancies in the first place (6).

Effective automated analysis also depends on a more declarative style of programming (3). It is generally easier for both people and machines to extract useful information from assertive statements than networks of imperatives.

Improving automated analysis is probably the most important challenge facing software technology. It can help in many ways, such as: reliability, informal interactive development, performance, recovery, migration between systems, etc. It is hard to overemphasize the need to "drain the swamp" of representational irrelevancies so this can proceed. This consideration was the main motivation for the original development of Dawn.

10. Easy access to multi-purpose data

The economic justification for putting information into a computer depends on capturing and preparing the data with a minimum of human effort and distributing the costs over as many users as possible. This economic pressure plus other technical developments are leading us in the direction of multi-user data bases and computer networks where many people and many application programs draw on increasingly diverse sources of information in ever larger integrated systems.

For this trend to proceed in an orderly way there is a basic problem to consider: Any person who manipulates the same body of information represented in two different data models must understand both representations and the transformation between them. Any such burden on the user is very damaging to ease of use.

To illustrate, if a user wants to write a program accepting inputs from a Cobol file and an APL workspace he has a data conversion problem. He would normally take one of the inputs, understand its representation, design a new representation, and specify a transformation. Conceivably he could manipulate a Cobol file which is automatically created and structured to represent any APL workspace, or an APL workspace whch is structured to represent any Cobol file. That would compound the representational irrelevancies of both models.

Such compounding of representational irrelevancies is fairly harmless if at least one of the data models avoids introducing significant representational irrelevancies. It is possible to represent Cobol files or APL workspaces in DAWN in a fairly straightforward way. Thus a user of DAWN can reach into , extract, and manipulate data in alien systems with relatively little difficulty. This assumes software support which is specific to the implementation of the alien data model but need not be specific to the application.

If the information structure is fully and accurately described in a formal way then conversion to a different data model can be fully automatic. However, if the description is deficient (as in the case of APL and Cobol) human intervention is needed to make up for the deficiency.

The user who works in a "high fidelity data model" (that is, one relatively free from represenatational irrelevancies) can access data in other models with little loss of convenience compared with native users of those models. The inconveniences would include syntactic differences and loss of performance. The user who works in a lower level model is limited to accessing other data in the same model or using data translated to his model from one whose information structures are fully described. Otherwise he must specify a conversion or accept a compounding of representational irrelevancies. In any case the representational choices must be understood by the user, whether made in preparing the original data, by his own manual translation, or by some mechanical translation. A probably unimportant exception would be representational choices in the original data which are formally described and hence automatically removable.

The above considerations indicate that providing easy access to diverse sources of multi-purpose data is strongly dependent on elimination of representational irrelevancies (6) and accurate description of system structure (8).

Compounding of representational irrelevancies is probably a natural characteristic of large networks. Distortions are cummulatiive when viewing anything through multiple interfaces. Whatever the user's view of how systems should be organized, the network will interface with subsystems built according to a different philosophy. The user's best protection against being overburdened by the more remote and uncontrolable irrelevancies is to use a high fidelity model himself which avoids aggravating the problems.

Easy access to diverse sources of data also depends on the relatively non-technical issue of common acceptance of a small number (preferably one) of standard data models. That process is helped when technical

analyses lead naturally to a narrow range of choices. It is hoped that the rationale below favoring the directed graph as a basis for a satisfactory model will help do that.

11. General Extensibility of Language

Languages for query, formatting, job control, graphics, simulation, text processing, etc. exist separately mainly to escape from representational irrelevancies in current languages. Their capabilities can be expressed naturally using DAWN with added functions. The point is not that Dawn provides all the language facility that a user may need, but that the basic parts of the language including data structuring, referencing, and basic functions provided, do not impose irrelevancies which lead the user to abandon them when specialized function and notations are needed. Relatively superficial language features such as specialized syntax may be added in situations where normal syntax would be unduly repetitious or non-traditional.

The term "programming language" is probably misleading when discussing generality. A system with a vocabulary of about a hundred words is not appropriately described as a language, and certainly not a general one. Dawn could more accurately be described as a generally extensible language core for programming and information system description. General extensibility depends mainly on avoiding representational irrelevancies (6) which limit the reusability of underlying structures. It is possible of course, to achieve high representational fidelity for some problem domains but not others. Integrated business systems of moderate size have very rich structural requirements which cover many other needs. The adequacy of Dawn for very different areas like theoretical physics and artificial intelligence has only been partially explored, but successfully so far. To include query capability, natural set references (4) are needed.

However special-purpose a language may be, most of the data it references will be multi-purpose. Easy access to multi-purpose data (10) is also a basic requirement. Effective extensibility also requires that the language plus extensions for some problem domain be subsetable so it will be as easy to learn as a completely specialized language with the same function. The lack of irrelevancies suggests that no unneeded complications will occur, but that supposition needs to be tested.

One kind of kind of generality in Dawn is that distinctions between data objects and computer system objects are eliminated from a language point of view. Although system oriented functions are needed, the underlying language for data processing and system control are merged. Since the language describes and references objects which are not data objects it is more appropriate to speak of the operand model of the language rather than its data model.

The generality of Dawn in describing both data and system objects is needed for large networks because system objects with novel structure and function will be routinely added to the network. The present practice of changing the control language whenever that happens will become increasingly unacceptable.

NEED FOR A DIRECTED GRAPH OPERAND MODEL

The simultaneous requirement for eliminating representational irrelevancies and using a small number of simple primitive operand types imposes severe constraints on the acceptable choice of primitive operands. If everything is built from a few kinds of "building block" we would resonably expect substantial irrelevant "granularity" in the results. This will happen if the primitives are large in the sense of having significant internal structure. For this reason the integer indexed array, which is a primitive for so many programming languages, is not satisfactory. The integer indices are usually unnecessary identifiers which users are forced to associate with objects in their problem environment.

The prevailing mismatch between programming languages and data base data models indicates the difficulty of satisfying the above two requirements. The unfortunate results are extensive. Dawn appears to be the only multi-purpose programming language with a data model suitable for data base.

While not rigorous the following rationale indicates that the nodes and arcs of directed graphs are optimal. Directed graphs have proven to be a useful way to represent many abstractions of many kinds of systems. All abstractions of all systems described in DAWN are represented entirely by directed graphs, and statements about them. To test for optimality we consider two questions:

(1) Can nodes or arcs be replaced with different primitives in a way that provides overall simplifications.
(2) If not, are there other primitives whose additional inclusion would provide overall simplification?

Consider (1). By itself, a node has no internal structure so it does not inherently incorporate anything extraneous when used to represent an object in the problem environment. There is nothing simpler to replace it with. The arc of a directed graph is also a very simple structure. It is adequate in representing relationships between objects and for building representations of complex structures. There are very few kinds of primitive element that are simpler. The node, the bit, and the arc from a non-directed graph are all simpler, but by themselves or in combination they cannot do the job of the directed arc. A set of bits in a "field"

may be used to represent a directed arc, but only if the bits themselves are related to each other by some separate mechanism (at least to provide sequencing). Conceivably the node and directed arc could be replaced by some more complex primitive such as binary subgraphs similar to those of LISP, an APL variable, or something as yet unsuggested. As noted above any primitive with substantial internal structure is very likely to introduce irrelevancies. Anything could be constructed of binary subgraphs, but they would be playing roles essentially the same as the node and arc. It seems unlikely that anything more complex than nodes and relations can be used to represent very many kinds of structure without irrelevancy unless parts of those primitives are ignored or suppressed in some way. The possibility of building from simpler things seems precluded by the extreme scarcity and inadequacy of simpler things. Nodes could be viewed as a special case of relations in which there is no interest in the objects they relate to or from. However, that does not seem to lead to useful simplifications.

Concluding at least tentatively, that nodes and relations are unlikely to be displaced as primitive operands the second question remains as to whether additional primitives, not built from nodes or relations can be added to a multipurpose language providing an overall simplification. Some candidates might be: undirected arcs, numbers, bits, character strings, arrays, N-way relations. Each of these are represented by structures of nodes and relations in DAWN. For each, treating them as separate primitives would complicate the language by adding special cases to the definitions of references, declarations, and primitive functions among other problems. Little would be gained since nodes and relations do a good job of representing each, though not a perfect job.

The set of real numbers is represented as an infinite unchanging set of nodes. Other objects do not contain numbers; they may contain relations to numbers. Such relations may be informally regarded as "fields" containing numbers. However, numbers are primitive in one respect. The arithmetic functions are primitive.

Character strings are represented in DAWN as shown in Figure 3. There is a set of nodes, ordered by a collating sequence representing the distinct members of the character set. A character string is a node with a set of relations from the string to individual characters in the alphabet. Those relations are ordered by another set of relations.

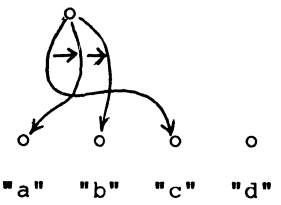


Figure 3. The character string 'cab'

Users will nearly always coneptualize, and manipulate such character strings as though they were primitives of the more conventional form [c|a|b] occupying fields. Most users will probably learn subsets of the language in which the underlying structure of character strings is not mentioned. The existence of the underlying structure gives meaning to many language constructs which would otherwise require special definition. Of course, a different representation is needed when text with variable typefaces, underscores, etc. are included.

The treatment of arrays and subscript notation as derived rather than primitive concepts may be one of the more important simplifications in DAWN. See Figure 4.

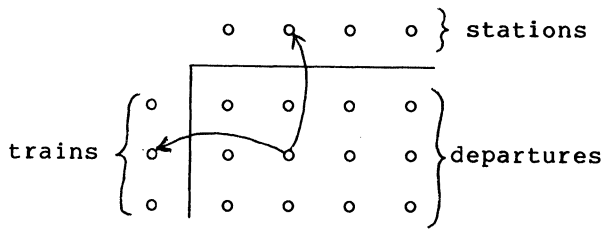


Figure 4. An array in Dawn

An N dimensional array in DAWN is a set of nodes each of which has N "key" relations which give its coordinates in the array. Using this approach subscript notation is just a syntactic abbreviation. For example:

```
departure_time (train_17, Denver)
```

would mean:

```
departure_time of departure where
  (train of the departure is train_17 and
  station of the departure is Denver)
```

A result of this treatment of arrays is that all mention of arrays in the DAWN definition is very brief and localized. A definition of a BASIC-like subset of DAWN in which arrays are treated as primitive seems useful and possible, but it has not been defined.

A bit is represented in DAWN by a node with a relation to one of the special nodes true and false. The conscious use of bits for representing other structures seems to arise mainly from temporary hardware limitations.

Another candidate for a primitive operand is the undirected arc. It is usually represented in DAWN by a pair of converse relations connecting the same two objects, as shown in Figure 5.

Figure 5. Undirected arc and representation

Any change to one member of a converse pair forces a corresponding change to the other.

For some cases it can be represented by a node with two relations. Except for sibling, spouse, adjacency, and a few others, symmetric relationships are relatively uncommon. While non-symmetric relationships could be represented by undirected arcs it is intuitively natural to regard them as pairs of relations and to use different names to refer to them. For example, the "is-owner-of" relation may imply responsibilities which the converse "is-possession-of" relation would not.

Another candidate primitive is the 3-way or N-way relations. Such a relationship is usually represented in DAWN by a node with 3 or N relations any of which could be associated with a converse relation.

The costs of not adding them as primitives are that the representation in DAWN primitives will sometimes show through. Syntactic abbreviations and special functions will paper over the representations most of the time but not always. The main difficulties will be caused by mixtures of variant ways of representing such structures.

For each of these candidate primitives the value of not introducing them as primitive operands, separate from nodes and relations, is a very substantial simplification of the language. Going from 2 to 3 kinds of primitive operand would tend to add special cases to the definition pervasively.

I tentatively conclude that introducing primitives in addition to nodes and arcs would be consistently counterproductive. I would welcome commentary from any source which tends to confirm or refute this conclusion.

[1] "An Interactive Business Definition System" by Hammer, Howe, and Wladawsky, Sigplan Notices, April 1974, pages 25-33.

PUTTING THE NAG LIBRARY ON THE VAX11/780

B. Ford, S.J. Hague and S. Vaughn
Numerical Algorithms Group Inc.
1250 Grace Court, Downers Grove, Illinois 60515

ABSTRACT

The NAG Library is a tailored collection of subroutines covering areas of numerical mathematics and is designed to assist computer users solving a wide range of mathematical and statistical problems. It is supported by detailed user documentation, by example programs demonstrating the use of the routine, and by a set of implementation test programs used to certify the performance of the software (according to standards of accuracy and efficiency) in each computing environment. The NAG Library has been written in FORTRAN, Algol 60 and Algol 68. In FORTRAN it is available on 25 machine ranges involving 53 different hardware/compiler configurations and including large scale and mini computer families. In particular it is available for DEC System-10 (KA, KI and KL), DEC System-20, DEC PDP 11 and DEC VAX 11. The paper will discuss aspects of algorithm design and subroutine construction which permits the apparently opposing goals of software transportability and accuracy and efficiency to be achieved. Implementation of the NAG Library was one of the first application packages completed on the VAX 11. The project provided significant exercising of the standard features of the FORTRAN-IV-PLUS compiler and the system's arithmetic, standard utility programs and file handling. Since a comparable activity has been completed upon a number of other configurations (including the DEC PDP 11) a brief qualitative assessment of the VAX will also be included.

1. INTRODUCTION

The NAG Library is a tailored collection of computer subroutines in mathematics and statistics, for use in a wide variety of disciplines. It serves the needs of both the novice and the expert, and its users include undergraduate students, professors, engineers, research scientists, statisticians, economists, applied mathematicians and many others. The principal language version is FORTRAN but there is an Algol 60 version offering nearly parallel facilities. A third language version has been developed in Algol 68. On almost all the medium and large-sized computer systems in general use, the FORTRAN Library is available, and it has also been implemented on a number of small and mini computer systems. The NAG Library is now used in more than 230 computer installations in over 20 countries around the world.

The current Mark (edition) of the NAG FORTRAN Library contains 345 user-callable routines which cover these areas of numerical analysis and statistics:

Summary of Contents of NAG Library

Simultaneous Linear Equations
Eigenvalues and Eigenvectors of Matrices
Linear Algebra (Miscellaneous)
Non-Linear Optimization
Operations Research
Non-Linear Equations
Quadrature
Numerical Differentiation
Ordinary Differential Equations
Integral Equations
Curve and Surface Fitting
Fast Fourier Transforms
Special Functions
Basic Statistics
Statistical Distribution Functions
Correlation and Regression Analysis
Analysis of Variance
Random Number Generators
Sorting
Error Trapping
Mathematical Constants and Machine Parameters

New Marks are produced approximately once a year, and at the next Mark (Mark 7 - due sometime in 1979) at least 70 new routines will be added including extensions to the ordinary differential equations and optimisation chapters, the launching of a partial differential equations chapter and an expansion of the statistical facilities. At later Marks the most significant features are likely to be a further strengthening of statistical chapters and the possible inclusion of graphical software.

The Numerical Algorithms Group (NAG) was formed in 1970 in Britain. The activity has four aims.
(a) To create a balanced, general purpose numerical algorithms library to meet the mathematical and statistical requirements of computer users, in

FORTRAN and Algol 60.
(b) To support the library with documentation giving advice on problem identification and algorithm selection, and on the use of each routine.
(c) To provide a test program library for certification of the library.
(d) To implement the library as widely as user demand required.

Its 150 members include numerical analysts and software specialists of international repute, and the activities of these advisors and contributors are coordinated by a full-time staff of 28 people most of whom are based in the NAG Central Office, Oxford. NAG Incorporated was established in March 1978 with a full-time office in Downers Grove, Illinois.

A description of the overall development of the NAG Library is given in [6]. In this paper, we concentrate on the process of implementation; that is, the process of adapting the Library on different computing systems. In Section 2, general principles of implementation are described. The VAX implementation in particular is described in Section 3.

2. IMPLEMENTATION - GENERAL CONSIDERATIONS

The term 'implementation' in the NAG context means the production of a well-tested and suitably prepared version of the NAG Library for a specific computing system; that is, a particular combination of hardware, operating system and compiling system. (The term is also used to denote the final version of the software itself). The testing process involves the execution of two sets of programs; the example programs which appear in the user documentation, and the more stringent, implementation test programs. Each user routine has its own example program and has an associated stringent test program.

It will therefore already be obvious that the implementation process is a major exercise involving the running of hundreds of programs. The reason for this significant attempt at certification in a particular environment is that it is not adequate simply to provide just a source text library. Even if an algorithm is competently encoded into a library routine, its behaviour can be affected, even ruined, by a variety of factors such as compiler errors, file corruption or mis-handling of data. In almost every implementation, the intensive and systematic running of a large number of test programs, some of which are particularly severe, has revealed at least some minor defect or foible in the compilation system or even in the machine arithmetic. In some cases these defects have been far from minor! Indeed the implementation exercise, viewed not as certification in an environment but as a test of the environment itself, has proved so effective that a number of manufacturers have asked NAG to allow them to use NAG test software as a 'weapon with which to assault a new compiler'.

Implementation experience in NAG has gradually been accumulated over the last six or seven years. When the library project began in 1970, though there was a strong emphasis on standards and conventions, they were formed largely in the context of a single machine library (the ICL 1906A/S). It was never envisaged that the number of machine range versions would reach its present figure. Soon, however, even after the first few additional implementations in 1972 onwards, the fundamental principle of multi-machine software development became clear. That principle can be summarised in one word, 'anticipation' - we must try at each stage in the development process to anticipate future constraints or requirements that are or may become relevant in particular context. This must not be done, however, at the cost of unduly affecting the behaviour of the software in existing environments or of introducing unacceptable generality or complexity of use.

In mathematical terms, the principle of anticipation is implied by the term 'adaptable algorithms', i.e. methods which adapt themselves to perform satisfactorily in particular environments. The algorithm developer must therefore have some means of characterising, in symbolic form, pertinent aspects of a computer system. This 'machine-modelling' approach is reflected in the NAG Library by the use of the X-chapter utility functions, which are adjusted to yield relevant values such as the base of the arithmetic or the largest positive integer, for each particular implementation. By using these basic utilities, the behaviour of the algorithm can be tailored to a specific machine (e.g. maximum precision on a CDC Cyber machine) but the coded algorithm as a library routine remains essentially implementation-independent. In some cases, e.g. approximations of special functions, a more elaborate approach (see [1]) is necessary but nevertheless can still be regarded as part of the adaptability policy which is further described in [2].

Even with adaptable algorithms, there may still be changes necessary at the software level because of syntax or semantic differences between language dialects on different systems. See [3] for a survey of portability in the NAG project. For aesthetic and organisational reasons we wish to minimise those changes. The principle of anticipation suggests therefore that library routines are coded in a restricted language subset which is contained within all present and, hopefully, most future implementation dialects. The NAG Library, in all three language versions, is written on the basis of this *intersection* approach. The Library is therefore in large part *portable* in the sense that much of it requires no change when moved from one system to another. It is not feasible, however, to avoid all changes (this is particularly true for Algol 60). If we can anticipate that changes will be necessary, making those changes (e.g. from single to double precision in FORTRAN or from one Algol 60 symbol representation to another) should be made as easily and reliably as possible. The use of programming aids to perform these changes mechanically or to impose coding standards prior to multi-machine implementation is increasing (see [4]). An appropriate term for the NAG Library is therefore *transportable* as defined in [5], i.e. consistent with performance requirements, coding changes between implementations are, where possible, avoided or mechanised.

Before considering the tactics to be adopted during an implementation, it is useful to review the scale of the exercise about to be undertaken. The overall task is to create a compiled library involving several hundred routines and to run all example and stringent test programs against that library until acceptable results are produced. For a new machine

range implementation at Mark 6 of the FORTRAN Library for example, this would involve receiving items of data from Central Office on this scale.

503	routines (incl. auxiliaries)	50,700 records
334	example programs	15,700
203	example program data sets	1,900
334	example program results	8,700
284	stringent test program	40,700
192	stringent test program data	10,200
284	stringent test results	63,100
2,137		191,100

At various stages in the implementation exercise, as software is modified or results generated for comparison, the number of separate items, records or characters is often increased by a factor of at least two. The implied requirements for file storage space might not disturb the prospective implementor on a large IBM 360/195 nor would the implied requirement for processor time involved in compilation and execution *en masse* 'frighten' the Cray-1 implementor. On many other machines however, particularly small or mini systems, it is vital that, from the outset, implementors appreciate the total volume of information and number of items involved.

3. IMPLEMENTATION OF THE NAG LIBRARY ON THE DEC VAX11/780

We now describe the implementation of the NAG FORTRAN double precision Library, Mark 6, on a DEC VAX11/780 at Marlborough, Mass. The implementation was derived from the NAG base double precision library, which is in effect a generalised version of the IBM 360/370 double precision implementation. Both the base and IBM implementations are prepared at Cambridge University.

3.1. Stages In The Implementation

(a) The complete Mark 6 double precision Library tape (including routines, example programs, data and results*, stringent test routines, data and results*) was transferred on an IBM format tape to a DEC PDP 11/70 at Reading. This involved converting over two thousand Library items into DEC ASCII disc files under the IAS system.

(b) Machine-specific constants were inserted in the X chapter utility routines. Similarly appropriate expansion coefficients were embedded in the Special Functions S chapter. These coefficients were identical to those used for the PDP/11 Mark 5 implementation on the assumption (later amply verified in computational practice) that the two machine families have the same arithmetic properties.

As double precision complex is not available in PDP/11 or VAX11 FORTRAN, those routines which in other NAG implementations use COMPLEX*16 had to be modified. Complex variables and arrays were replaced by DOUBLE PRECISION arrays with an extra leading dimension of 2.

(c) The modified base software was written out to two magnetic tapes in /DOS format, taken from the PDP11 at Reading, England, and read into the VAX11 filestore at Marlborough, U.S.A.

(d) Using the FORTRAN-IV-PLUS compiler, the FORTRAN Library routines were compiled. The LIBRARIAN facility was then used to create a compiled library, ready to be tested.

(e) Two FORTRAN programs, already developed during the PDP/11 implementation, were modified to generate from a supplied index of names, job control to compile, execute (and optionally compare the results of) programs. The first program, EXJOBGEN, applied to example programs; the second, STJOBGEN, to the more stringent implementation test programs.

(f) The testing began with the execution of the Special Function chapter which contains routines for the accurate computation of Bessel functions, error functions, arctangents etc. Thus, running the associated test software, whilst not providing a comprehensive examination of all aspects of the underlining arithmetic, does sometimes reveal deficiencies or undermine assertions. (In the VAX11 case, claims about PDP/11 compatibility were substantiated by this preliminary exercise, but a compiler problem was revealed - see 3.2).

(g) All other example programs, apart from those for routines in assembly language, were executed, and the results produced were checked. No particularly significant discrepancies were found between the VAX11 and IBM results.

(h) A similar operation as in (g) but for implementation test programs was carried out next. The results checking phase revealed one significant case of inaccuracy, concerning the curve fitting routine E02DAF. (This was later ascribed to an error in the optimisation phase of the compiler, and an appropriate correction initiated). The rest of the programs run produced results which were acceptably close to the reference results.

(i) About the same time that stage (e) was started, work also began on the writing of several routines in the VAX-MACRO assembly language. One of these routines accumulates inner products in greater than double precision, and the others are concerned with random number generation. During the development of these routines, the DEBUG facility was used extensively. When testing was satisfactorily completed, the final stages of implementation could commence.

(j) All outstanding test programs, which depended on the completion of stage (i), were run, and their results checked.

(k) A release tape containing the tested compiled Library, the routine source text and example program text, data and results, was constructed in the PDP/11-compatible /DOS format.

* IBM results supplied for reference purposes

(1) A Library Support Note, describing the composition of the above tape and the installation of the Library, was written. This note is primarily for the benefit of computer staff responsible for mounting and supporting the Library. An Implementation Document was also written. This gives implementation-dependent information for users of the VAX11 Library.

3.2. Assessment Of The Implementation

Implementation of the Mark 6 VAX11 FORTRAN double precision Library involving stages (a) to (l) was completed in about 5 man-weeks; this is much the shortest period that any implementation of the NAG Library on a new system has taken. We now consider some of the factors behind that undoubted success and compare aspects of the VAX11 experience with that on PDP11 and with previous implementation experience generally.

(a) Reliability Considerations - The VAX/VMS system used for the implementation proved to be thoroughly reliable throughout the exercise, apart from the perhaps inevitable hitch at the very start (see (b)). After that isolated incident, all basic facilities employed performed satisfactorily. No information was lost or corrupted during the processing of more than 2000 files.

Arithmetic behaviour was according to specification (that has not always been our experience on other systems) and the mathematical functions in the FORTRAN compiler library performed as expected. There were two minor numerical 'quirks':

- all constants less than 10^{-38} had to be entered as 0.0D0.

- the expression DBLE(FLOAT(N)) lost accuracy when N is a integer with more significant figures than a single precision floating point real number (problem cured by using DFLOAT).

The FORTRAN compiler itself emerged with satisfactory credit from the implementation exercise. The problems posed by that exercise should not be underestimated. Many of the test cases present a searching examination of the compilation system and even compiling the Library itself, which has a highly modular structure in parts, has on other occasions revealed deficiencies in the compiler and related utilities. In the VAX11 case, two compilation difficulties arose:

- five routines in the Special Function chapter contained nested multiplication with more than 20 levels of nesting in one statement. The problem had been encountered with the PDP/11 FORTRAN-IV-PLUS compiler and is more a compiler working stack limitation. Whereas on the PDP/11, the compiler had been reconfigured, for the VAX implementation, the five routines were modified to permit them to compile.

- one routine in the Curve and Surface Fitting chapter would not work when it was optimised. When recompiled with the optimising option switched off, it produced answers which were in the main satisfactory but were unacceptable in one test case. After investigation, the likely case of the problem was isolated in the compiler itself, and appropriate corrective action initiated.

(b) Comparisons With PDP/11 Experience - Familiarity with IAS gained whilst performing the PDP/11 implementation was useful in switching to VAX/VMS, which has most of the same commands (although parameters are sometimes different) and has the same filestore structure.

The ability to exchange disc and tapes between the PDP/11 and VAX is obviously most useful (though the very first attempt at transfer in stage (c) failed because of system corruption).

Coding in VAX-MACRO proved fairly easy to write and test with knowledge of PDP-11 MACRO. Although direct conversion was not feasible, the random number and inner product routines were quickly developed and tested, mainly due to the availability of the DEBUG facility.

Numerically, the behaviour of the two implemented libraries, even on the most sensitive test cases, was found to be almost or exactly identical.

On the basis of comparing approximate timing of about 12 test programs, it appeared that VAX11 execution speeds were between 1½ and 2 times as fast as those obtained on the 11/70, except for programs which took only a few seconds to run. In those cases the 11/70 code was faster, presumably because of VMS overheads on VAX.

(c) Comparisons With Other Implementation Experience It would be perhaps unfair and unwise to draw sweeping conclusions about VAX computers in comparison with other specific machine ranges. One system may appear to be superior or more suited to our needs than another in one respect but inferior or less convenient in other regards. Our goal has always been to implement our Library quickly but thoroughly; it has not been to conduct a wide-ranging and objectively-designed comparison of different computer systems. With that proviso, we would offer the following general observations about our experience:

- the VAX/VMS system demonstrated a basic reliability during implementation. Other systems, particularly those recently introduced, have not on the whole displayed the same reliability.

- the filestore structure and handling facilities proved particularly convenient for the file-intensive implementation activity.

- execution speeds compared favourably with other machine ranges.

- Some specific utilities were found especially useful (and are lacking on some comparable systems) e.g. automatic ordering of library entries, sorted directories.

- the job control language was easy to learn and proved reasonably un-verbose; for instance to compile and execute a test program took 7 VAX commands and over 20 on another manufacturer's system.

ACKNOWLEDGEMENTS

The NAG VAX11 implementation was undertaken by Sallie Vaughn of the NAG Central Office, using the VAX-11/780 of the Engineering Systems Group at

Digital's Marlborough offices.

REFERENCES

[1] Schonfelder, J.L., 'The Production and Testing of Special Functions in the NAG Library'. In 'Portability of Numerical Software', Oak Brook, 1976. W.R. Cowell (Ed), Lecture Notes in Computer Science, No. 57, Springer-Verlag, New York, 1977.

[2] Ford, B., 'The Evolving NAG Approach to Software Portability'. In 'Software Portability'. P. Brown (Ed), Cambridge University Press, 1977.

[3] Bentley, J. and Ford, B., 'On the Enhancement of Portability Within the NAG Project - A Statistical Survey'. In 'Portability of Numerical Software', Oak Brook, 1976.

[4] Du Croz, J.J., Hague, S.J. and Siemieniuch, J.L., 'Aids to Portability Within the NAG Project'. In 'Portability of Numerical Software', Oak Brook, 1976.

[5] Hague, S.J. and Ford, B., 'Portability - Prediction and Correction', Software Practice and Experience, Vol. 6, 1976.

[6] Ford, B., Bentley, J., Du Croz, J.J. and Hague, S.J., 'The NAG Library Machine', Software Practice and Experience, 1978 (to appear).

APPENDIX

Technical Details Of The NAG VAX11 Implementation

Processor: VAX11/780
Operating System: VAX/VMS
Compiler: FORTRAN-IV-PLUS To.7-92
Options: Traceback
Optimise
Check
Overflow
Precision: Double

For further details of the NAG Library Service, please contact

The Secretary,
Numerical Algorithms Group Incorporated,
1250, Grace Court,
Downers Grove,
Illinois 60515

Tel: (312) 969 7107

MATHEMATICAL-STATISTICAL LIBRARIES STATE-OF-THE-ART

Thomas J. Aird
IMSL
Sixth Floor, GNB Building
7500 Bellaire Boulevard
Houston, Texas 77036
(713)772-1927

ABSTRACT

This paper discusses mathematical-statistical, Fortran libraries. The first part covers some general topics: how libraries are developed and supported; who uses libraries, etc.. The second part deals with specific changes that IMSL is making to its library for Edition 7 (to be released in January, 1979).

GENERAL TOPICS

The term "library", as used in this paper, means a unified collection of basic computational routines (written in Fortran) along with associated documentation covering mathematics and statistics. A library is more than just a collection of subroutines. The term "unified" is a crucial part of the definition. It means that a great deal of effort has been devoted to standardizing routine names, argument lists, and documentation. This standardization is intended to make a library easy to use. A proper naming scheme means that routines are easy to locate in the reference manual and standardized documentation makes it easier for the user to understand once the proper routine is located. The IMSL Library is divided into seventeen chapters designated by the first letter of the chapter title; A, B, C, D, E, F, G, I, L, M, N, O, R, S, U, V, and Z. Figure 1 gives the chapter titles. There are more than 400 routines in the IMSL Library and the chapter grouping into natural subsets of mathematics and statistics, aids the user in locating specific routines. Each routine name begins with the corresponding chapter letter and reference documentation is arranged alphabetically within the manual.

LIBRARY CHAPTERS

ANALYSIS OF EXPERIMENTAL DESIGN DATA
BASIC STATISTICS
CATEGORIZED DATA ANALYSIS
DIFFERENTIAL EQUATIONS; QUADRATURE; DIFFERENTIATION
EIGENSYSTEM ANALYSIS
FORECASTING; ECONOMETRICS; TIME SERIES
GENERATION AND TESTING OF RANDOM NUMBERS; GOODNESS OF FIT
INTERPOLATION; APPROXIMATION; SMOOTHING
LINEAR ALGEBRAIC EQUATIONS
MATHEMATICAL AND STATISTICAL SPECIAL FUNCTIONS
NON-PARAMETRIC STATISTICS
OBSERVATION STRUCTURE
REGRESSION ANALYSIS
SAMPLING
UTILITY FUNCTIONS
VECTOR, MATRIX ARITHMETIC
ZEROS AND EXTREMA; LINEAR PROGRAMMING

Figure 1: Library Chapters

Other important library attributes are: (1) consultation, (2) maintenance, and (3) evolving with the state-of-the-art. Consultation implies that someone is available, by telephone or mail, to answer questions about the library. Maintenance means that any errors in library routines are corrected and not left in a continual state of being "rediscovered". Evolving with the state-of-the-art means that obsolete algorithms are being replaced when improved versions are available. In order to provide this service, IMSL maintains an advisory board comprised of experts in the area of numerical mathematics, statistics, and computer science. Figure 2 lists the IMSL advisors and gives their area of specialization and affiliation.

As an example of how a library is used, suppose that one wishes to solve a system of linear equations AX=B. The required steps are as follows:

(1) Locate and read documentation for the appropriate routine.

(2) Write a program to call the library linear equation solver routine. The program must dimension and initialize the matrix A and vector B, dimension WK, and set N and IA correctly.

The CALL statement might be

```
CALL LEQT1F (A,1,N,IA,B,0,WK,IER)
```

if the IMSL Library is being used.

The user of a linear equation solver should,

(1) know something about linear systems, i.e., be familiar with the terms matrix, vector, row, column, order, singularity, solution, right-hand side

(2) know how to write a Fortran program that passes arguments to a subprogram.

The user should not be required to know about the algorithm used to solve the problem in order to properly call the routine.

The following figures 3 to 6 give various categorizations of IMSL subscribers to show "Who is using the IMSL Library".

R. ANDERSON	STATISTICS	UNIV. OF KENTUCKY
K. BROWN	NUMERICAL MATH NONLINEAR OPTIMIZATION	UNIV. OF MINNESOTA
W. CODY	NUMERICAL MATH SPECIAL FUNCTIONS	ARGONNE NATIONAL LAB
C. DE BOOR	NUMERICAL MATH SPLINE APPROXIMATION	UNIV. OF WISCONSIN
W. GENTLEMAN	NUMERICAL MATH LINEAR ALGEBRA	UNIV. OF WATERLOO
T. HULL	NUMERICAL MATH DIFFERENTIAL EQUATIONS	UNIV. OF TORONTO
M. JOHNSON	MECHANICS APPLIED MATH	UNIV. OF WISCONSIN
W. KAHAN	NUMERICAL MATH	UNIV. OF CALIFORNIA BERKELEY
R. KARPINSKI	COMPUTER SCIENCE LANGUAGES	UNIV. OF CALIFORNIA MEDICAL CENTER
P. LEWIS	STATISTICS RANDOM NUMBER GENERATION	NAVAL POSTGRADUATE SCHOOL
M. LYNN	COMPUTER SCIENCE	UNIV. OF CALIFORNIA BERKELEY
C. MOLER	NUMERICAL MATH LINEAR ALGEBRA	UNIV. OF NEW MEXICO
B. PARLETT	NUMERICAL MATH LINEAR ALGEBRA	UNIV. OF CALIFORNIA BERKELEY
M. POWELL	NUMERICAL MATH NONLINEAR OPTIMIZATION	CAMBRIDGE
J. RICE	NUMERICAL MATH APPROXIMATION	PURDUE UNIV.
J. THOMPSON	STATISTICS FORECASTING	RICE UNIV.
J. TRAUB	NUMERICAL MATH ZEROS	CARNEGIE-MELLON UNIV.
K. WARWICK	STATISTICS FACTOR ANALYSIS	WARWICK RESEARCH

Figure 2: The IMSL Advisory Board

COMPUTER	NUMBER	% OF TOTAL
IBM	249	39
CDC	140	22
DEC 10	69	11
UNIVAC	55	9
HIS	34	5
XEROX	28	4
BGH	23	4
DGC	20	3
DEC 11	7	1
TELEFUNKEN	3	1
OTHER	6	1
(JUNE 1978)	634	100

Figure 3: IMSL Subscribers by Computer Type

INDUSTRIAL		234
COLLEGES, UNIVERSITIES		268
GOVERNMENT		97
AUSTRALIA	4	
CANADA	7	
DENMARK	1	
FRANCE	3	
GERMANY	1	
ISRAEL	1	
ITALY	1	
NETHERLANDS	2	
SOUTH AFRICA	2	
SWEDEN	1	
U.S.A.	73	
VENEZUELA	1	
TOTAL (MARCH 1978)		599

Figure 4: IMSL Subscribers by Category (Industry, College, Government)

ADVERTISING	2
AEROSPACE & AIRCRAFT	35
ATOMIC & NUCLEAR RESEARCH	3
AUTOMOBILES & TRUCKS	14
BANKS	4
BUSINESS EQUIPMENT & SUPPLIES	11
CHEMICALS	16
DRUGS & HEALTH CARE	14
ELECTRICAL EQUIPMENT	8
ELECTRIC POWER	6
ELECTRONICS	15
ENGINEERING & CONSTRUCTION	5
FARM EQUIPMENT	3
FOODS	1
GLASS & GLASS PRODUCTS	2
HOTELS, MOTELS, RESTAURANTS	1
INSURANCE - PROPERTY & CASUALTY	2
MACHINERY & EQUIPMENT	3
METALS & MINING	2
NATURAL GAS	2
OIL	22
OIL SERVICE	1
PAPER & PRODUCTS	1
PRINTING & PUBLISHING	2
RAILROADS	2
SERVICE & RESEARCH	42
SOAP	1
STEEL	2
TELECOMMUNICATIONS	9
TIRE & RUBBER	2
TV & RADIO BROADCASTING	1

Figure 5: IMSL Subscribers by Industry

AUSTRIA	3
AUSTRALIA	18
BRAZIL	3
CANADA	37
COSTA RICA	1
DENMARK	2
FINLAND	2
FRANCE	6
HONG KONG	1
ISRAEL	7
ITALY	2
JAPAN	4
MEXICO	2
NETHERLANDS	13
NEW ZEALAND	1
NORWAY	2
SAUDI ARABIA	1
SOUTH AFRICA	4
SPAIN	1
SWEDEN	9
SWITZERLAND	5
TURKEY	1
UNITED KINGDOM	6
VENEZUELA	1
WEST GERMANY	18

Figure 6: IMSL Subscribers Non-United States by Country

Library development and support is an expensive and complex task. A reasonable estimate of development cost for high quality software is $10 to $30 per source code statement. This figure does not include costs for original research or for support activities. The IMSL Library contains approximately 50,000 source code statements and the initial development cost was about $1,000,000. The annual cost of supporting the library is approximately $500,000.

Some of the trends in mathematical software are outlined below.

(1) User standards of acceptability are becoming much higher. Users expect software to be easy-to-use, robust, and portable.

(2) Much more effort is going into the implementation (programming) stage of code development. The importance of this activity is being recognized. Several books have appeared that are devoted to specific Fortran codes. For example:

Gear, Numerical Initial Value Problems in Ordinary Differential Equations, 1971.

Brent, Algorithms for Minimization Without Derivatives, 1973.

Lawson/Hanson, Solving Least Squares Problems, 1974.

Smith/et al, Matrix Eigensystem Routines - EISPACK, 1974.

Shampine/Gordon, Computer Solution of Ordinary Differential Equations, 1975.

de Boor, A Practical Guide to Splines, 1978.

(3) There are about 1000 papers published in journals each year that deal with numerical computation research. Many journals publish algorithms. In 1974, ACM initiated a journal called Transactions on Mathematical Software.

(4) Many math software conferences are being held each year.

(5) A great deal of interest in portable software.

Cowell, Portability of Numerical Software, 1977.

(6) A new Fortran standard was adopted in 1978.

(7) Several funded projects concerned with developing high quality software: EISPACK, FUNPACK, LINPACK, ROSEPACK. Results from these projects are in the public domain.

IMSL EDITION 7

Edition 7 was discussed in an article in the IMSL Numerical Computations Newsletter, Issue 15, February 1978. That article is reproduced below.

"Seven years ago IMSL announced a subroutine library for IBM computers. Since that time, the library has been developed for seven other computer types. The addition of computers one at a time has resulted in a product that is quite complex to evolve and maintain. There are five separate, two volume, reference manuals, covering 400 subroutines, each having eight versions (ten versions for S/D routines). A great deal of input has come to IMSL

regarding how the library should be improved and evolved. The sources are: IMSL subscribers (there are 700 at the present time), the IMSL Advisory Board, an NSF-supported portability study, and vendors of computer services. Edition 7 of the IMSL Library will represent a substantial effort to achieve the following goals:

(1) The new edition must include structural improvements in the software and its documentation.

(2) The results of the effort must be pleasing to current subscribers and provide improved library service.

(3) The product must allow subsequent new editions to be produced and maintained more easily for all computer types supported by IMSL.

We feel that the Edition 7 changes will serve these goals. In addition, the revised structure will aid subscribers who change computer types or have multiple computer types at their installation. Feasibility of product development by IMSL for new environments will be enhanced.

A major part of the Edition 7 effort will be to produce one (3 volume) reference manual which will serve all computer types. Some of the planned improvements are listed below.

(1) Tabs will be provided for the manual: INTRO, CONTENTS, KWIC INDEX, OTHER INDICES, CHAPTER A, ..., CHAPTER Z.

(2) Each routine will have an example. Type statements, dimension statements, input, and output will be shown. A brief description of the problem being solved will be given.

(3) Standard wording will be used to describe arguments that relate to DIMENSION statements.

(4) Some "Programming Notes" will be moved from the typed part of the document to the machine readable section under the heading "REMARKS." This structure should aid on-line documentation systems.

(5) A new routine, UHELP, will be written to provide on-line access to important information about the IMSL Library.

(6) In addition to the current KWIC INDEX, other indices based on classification schemes such as SHARE, ACM, and ISI will be included in the reference manual.

Another part of the effort will deal with code changes needed to standardize argument lists and structure across computers. Some routines must be rewritten to achieve this uniformity, and we want to minimize the size of this set. Edition 7 will be upwards compatible with the current library. Whenever an argument list must change, a new name will be used and subscribers will be permitted to retain old codes and to continue using them. IMSL's current policy of supporting deleted codes for one year after edition release will be continued.

Two new routines are planned for handling error message printing and I/O unit selection:

(1) UERSET will allow the user to selectively eliminate printed error messages.

(2) UGETIO will allow the input/output unit designators, used by IMSL routines, to be changed by the installation or by the user during program execution.

In summary, Edition 7 will represent a content upgrade consistent with that of prior new editions. Product/service structural improvements that reflect the seven years of user-IMSL experience will appear as well. Edition 7 is scheduled for release to subscribers in January, 1979."

NUMERICAL METHODS IN LABORATORY MEDICINE USING THE MUMPS PROGRAMMING LANGUAGE

Frank B. Griffith
University of Arizona Health Sciences Center
Tucson, Arizona

ABSTRACT

The MUMPS programming language contains an extended precision capability, the $M function. Using this feature, fundamental numerical and statistical analysis programs have been written for laboratory medicine applications at the Arizona Health Sciences Center. The programs include statistical analysis, series approximations to transcendental functions, and fitting polynomial curves to two-dimensional data.

LABORATORY APPLICATIONS

There are specific areas of laboratory medicine in which the test data are two-dimensional and have historically been presented and analyzed graphically. These include time/concentration measurements in toxicology, zone size/concentration in antibiotic assays, and absorbance/concentration measurements in serum proteins (immunoglobins). These tests are typically done with known standards, and measurements are plotted on graph paper to obtain a standard curve from which the patient test value is interpolated. With fundamental methods of numerical and statistical analysis, it is possible to define which of several function types fits the standard curve test and to use that function for test value computations.

ACCURACY

The $M function, as defined in MUMPS, provides the user with the extended precision arithmetic capabilities necessary in laboratory medicine applications. Laboratory data are typically read with an accuracy of two or three decimal places, with an occasional method requiring four places. Therefore, all arithmetic in the support functions in done in the $M format. In each of the five function routines, the user defines an epsilon to be used in the convergence delta check. Each function is written as a converging series approximation with the series evaluation terminating when the last addend of the series is less than epsilon. The typical value of epsilon is .000001.

POLYNOMIAL CURVE-FITTING METHODS

There are currently four function types which can be defined and evaluated for a given set of data points. Each of the four is derived by the "least squares" method, with the goodness-of-fit being evaluated as the square root of the sum of the squared differences. The four functions are:

linear	$y = a + bx$
quadratic	$y = a + bx + cx^2$
cubic	$y = a + bx + cx^2 + dx^3$
exponential	$y = ae^{bx}$

LOGARITHMS

The log function is written for natural logarithms, base e (2.718281828), noted as *ln*. The convergent series is:

$$ln\ x = 2\left(\frac{x-1}{x+1} + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \frac{1}{5}\left(\frac{x-1}{x+1}\right)^5 \ldots\right) \text{ for each } x>0.$$

The user can apply a conversion to the calculated logs to change bases. The relationship $\log_a b = ln\ b\ /\ ln\ a$ makes the specific ln approximation a general purpose log function.

EXPONENTIATION

The exponential function is written to evaluate e^x, where e = 2.718281828. The series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \text{ converges for all real x.}$$

The user can evaluate any real number to any real exponent by using the logarithm/exponentiation routines. To evaluate $y=x^n$, where x and n are positive real numbers:

$$y = x^n$$
$$ln\ y = n\ ln\ x$$

Evaluate *ln* x and multiply by n to get a constant c:

$$ln\ y = c$$
$$y = e^c$$

Now evaluate e^c to obtain the desired y as a function of x.

SQUARE ROOTS

The Newton-Raphson method for determining roots has been applied to determine square roots. The procedure is iterative, requiring an initial estimate of the root of n. The initial estimate, x_1, is taken to be $R(n). This will be very close to the final value, so that the routine will usually converge to an epsilon of .000001 in three iterations. To find the square root of N, the equation $x^2=N$ will be solved for x:

$$x^2 = N$$
$$f(x) = x^2 - N = 0$$
$$f'(x) = 2x$$

The Newton-Raphson method computes successive approximations by evaluating:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

The series is assumed to have converged when the absolute difference between two successive estimates $[x_{i+1} - x_i]$ is less than epsilon.

TRIGONOMETRIC FUNCTIONS

The convergent series approximations to Sin (x) and Cos (x) are:

$$\text{Sin } (x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \ldots \text{ for all real values of x.}$$

$$\text{Cos } (x) = x - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} \ldots \text{ for all real values of x.}$$

The argument x is expressed in radians and a conversion expression to change from degrees to radians is:

Angle (Rad) = Angle (degrees) #360/360*2*π

There are basic definitions and relations whereby the other four trigonometric functions can be evaluated with the Sin and Cos functions.

CHARACTERIZATION OF PDP-11 PSEUDO-RANDOM NUMBER GENERATORS[a]

Paul R. Nicholson[b], John M. Thomas
and Charles R. Watson
Ecosystems and Energy Systems Departments
Pacific Northwest Laboratory
Operated by Battelle Memorial Institute
Richland, Washington 99352

ABSTRACT

The pseudo-random number generators of four PDP-11 languages are characterized. The algorithms for the PDP-11 functions RND (BASIC-11 and BASIC-PLUS-2), RAN, and RNDU (FORTRAN IV and FORTRAN-IV-PLUS) are essentially identical. The period is approximately 5×10^8. Statistically significant differences between observed and expected values did not exceed those expected for the frequency test. However, results of the runs and lag-product tests indicate some short-term behavior that may be undesirable for certain applications. A "pseudo-period" was detected and could also influence a few applications of the DEC algorithm where long sequences of numbers are used.

> "It has been said that more computer time has been spent testing random numbers than using them in applications! This is untrue, although it is possible to go overboard in testing."
>
> Knuth, Vol. II, p. 66

INTRODUCTION

Random numbers are used in many computer simulation techniques. The testing necessary to adequately characterize a particular random number generator depends on the consequences (within a simulation) of using numbers generated in a non-random fashion. The effect of a slightly non-random simulation in a blackjack game used for recreational purposes is by no means the same as when the results are to be used to "invest" at a Las Vegas casino.

In critical cases, such as personally financed gambling, the user should probably write a random number generating routine using one of the sophisticated algorithms recommended by Knuth.[1] At the other extreme, some users can accept the results from a vendor supplied routine at face value, with only cursory evaluation, because the consequences of using an average random number generator may be quite modest. Most cases fall in the middle. The majority of users may be aware that good quality random numbers are necessary for an application, but are hesitant to sacrifice the efficiency and convenience of the built-in random number generators unnecessarily. In this paper we characterize the DEC pseudo-random number generators to aid users in evaluating their applicability for particular problems.

Our original intent was to test and compare the pseudo-random number generators supplied with the DEC PDP-11 languages that were available to us. We expected to find a difference in quality between results obtained on our small 11/34, running BASIC-11 under RT-11 and those from FORTRAN or BASIC-PLUS-2 on our 11/70 under IAS. However, we found no qualitative difference; the algorithms of RND in BASIC-11 and BASIC-PLUS-2, RAN and RANDU in FORTRAN-IV and FORTRAN-IV-PLUS are essentially the same, differing only in their starting values and speed of execution. This is not apparent in the Language Reference Manuals, but a careful examination of the MACRO source language subroutines revealed the same basic algorithm.

Many methods for generating random numbers have been proposed and tested;[1,2,3] only a few have been found acceptable. Of the acceptable methods, the linear congruential algorithm

$$V_{i+1} = (aV_i + c) \bmod m$$

has the most widespread use. This algorithm can be implemented to run very efficiently when the constants a, c and m are optimally chosen. If m is the wordsize of the computer, then the mod function can be implemented in machine language by a simple bit rotation.

(a) Prepared for the U.S. Department of Energy under Contract No. EY-76-C-06-1830
(b) Current address: University of Washington, Seattle, WA 98195.

The maximum possible period (number of consecutive different random numbers) of a random number than generator using this algorithm is m. When m is the wordsize of a 16-bit computer, a period greater 2^{16} (65,536) is theoretically impossible. Such a short period would not be worthwhile for most serious applications; the supply of random numbers could be exhausted in less than five seconds of CPU time! To alleviate this problem, the DEC algorithm stores m in two words; thus the theoretical period on a PDP-11 is 2^{32} (greater than 4 billion). This theoretical period may be attained only if the remaining two constants, a and c, are optimally chosen. Knuth[1] discusses this problem in detail. The DEC algorithm sets $a = 2^{16}$, a number which may be efficiently entered in MACRO. To further speed the calculation, the algorithm uses c = 0, so the final equation is

$$V_{i+1} = (2^{16} + 3) \quad V_i \mod 2^{32}$$

The resulting "random number" is then divided by a constant scaling factor to put it in the interval (0,1). However, when c = 0 the maximum theoretical period is reduced, but we show below that it is still long enough for most purposes.

The inter-language differences arise because each language uses a different starting value, V_0. Further, the user has the option of entering the generator at a fixed or random starting value. Thus the numbers returned by the various PDP-11 generators will be different even though the underlying sequence of numbers is quite similar and would be identical if the periods were maximized.

Knowing that conclusions from one language would be applicable to the others simplified our evaluations. We were able to run time-consuming programs in FORTRAN or BASIC-PLUS-2 on off-shift time using the 11/70 and perform shorter tests in BASIC on prime-time using the 11/34.

There are dozens of statistical tests recommended for the evaluation of computer generated pseudo-random numbers.[1] We chose four: an evaluation of the period length and frequency, and tests of runs and lag products. The latter procedure is a variant of the serial correlation test. Others, including the poker test, the coupon collector's test, the maximum-of-t, test and the serial correlation test, may be necessary to insure the suitability of the DEC algorithm for some specific applications.

PERIOD LENGTH

Theory

Because of the mathematical characteristics of generating functions, all random-number generators currently available operate in periods (cycles). After a certain period, sometimes very long, the sequence will start again, i.e., numbers will begin to repeat. For all but the most intensive uses, the period need not be greater than several million, but in fact it is reasonable to expect the period of a liner congruential generator to be greater than 100 million.

Methods

A computer routine to empirically determine the period of a random number generator is conceptually and operationally simple, but because of the computing times involved, it is not often attempted. As a rough guide, FORTRAN RAN calls can generate 100,000 numbers in 6 seconds, whereas the equivalent BASIC statements take about 3 minutes to execute. Based on these figures the generation of 100 million numbers would take 1-2/3 hours and 58 hours for FORTRAN and BASIC, respectively. However, two more factors must be considered. First, each number generated by RND must be compared to another in an IF statement. Second, a period of over 500 million can reasonably be expected. Generating time in BASIC would be over 12 days on an 11/34 computer. Both generating and comparing would take about 17 days. Clearly, this sort of testing should not be attempted in BASIC, nor in FORTRAN when computer time is expensive.

To test for the period of RAN, we generated, saved, and compared several consecutive random numbers as follows:

```
[1] generate original sequence:
    call RND several thousand
    times, then
    a = RND No.
    b = RND No.
    c = RND No.
    c = RND No.
    count = 0
[2] loop and test:
    a1 = RND No., increment count
    a1 ≠ a?, go to 2
[3] test other 3 values:
    b1 = RND No.
    c1 = RND No.
    d1 = RND No.
    print: a, b, c, d
        a1, b1, c1, d1,
        and count
    b1 ≠ b, increment count
        go to [2]
    c1 ≠ c, increment count
        go to [2]
    d1 ≠ d, increment count
        go to [2]
    Done
```

Results

We found that individual random numbers do occur more frequently than expected. For lack of a better phrase we call this phenomenon the "pseudo-period." During our investigation of the period for BASIC-PLUS-2 RND, the program exited from Step [2] of the above algorithm 16 times. In 15 of these 16 cases the tests in Step [3] failed and returned to the loop-and-test stage (Step[2]). As shown in Table 1, whenever we found an exact match of a, we observed near matches of b, c and d.

Clearly, the high correlation for the three successive numbers (after the match of a and a_1) is undesirable; but until further research is conducted, the extent and meaning of this problem (for practical applications) will remain unclear. Marsaglia[4] and Maclaren and Marsaglia[5] indicate

Table 1. Original and "Matched" Sequences of Pseudo-Random Numbers Generated to Check the Period of RND.

Original Sequence a	b	c	d	Millions of Random Numbers Generated	Length of Psuedo-Period in Millions
.310183	.0876232	.734091	.615940		
match	"pseudo matches"				
.310183	.0870738	.730795	.601108	17	17
.310183	.0885387	.739585	.640660	48	31
.310183	.0880504	.736655	.627476	72	24
.310183	.0873790	.732626	.609348	73	01
.310183	.0862970	.729331	.594516	153	80
.310183	.0883556	.738486	.635716	306	150
.310183	.0878063	.735190	.620884	317	11
.310183	.0882946	.738120	.634068	338	21
.310183	.0871349	.731167	.602756	347	09
.310183	.0873180	.732260	.607700	367	20
.310183	.0885998	.739951	.642308	397	30
.310183	.0881115	.737021	.629124	429	32
.310183	.0878673	.735556	.622532	469	40
.310183	.0868907	.729697	.596164	480	11
.310183	.0856210	.733725	.614292	495	15
.310183*	.0876232*	.734091*	.615940*	537	42

* Exact match of the original sequence.

that uniform random numbers calculated using congruential generators can be shown to fall in sets of parallel hyperplanes. Our empirical evidence may indicate such an effect since an exact match of four consecutive numbers was not obtained until 537 million numbers were generated.

FREQUENCY TEST

Theory

Pseudo-random-number generators are designed to produce a uniform distribution on the interval zero to one. If 1000 uniformly distributed numbers are grouped into 10 classes of equal size, then 100 numbers should be found in each class. In general, the expected value is T/n, where T is the total number of observations, and n is the number of classes.

Method

We generated 1,000,000 numbers, classified them in 100 categories (or cells), and repeated the procedure 39 times. The theoretical and observed values were tested for goodness-of-fit using the chi-square (χ^2) test.

Results

The frequency of statistically significant ($p < 0.05$) χ^2 values for the 39 cases was close to the number expected, Table 2. In addition, we recorded the number of times the highest and lowest observations occurred within a class interval for each of the 39 repeated runs (Table 3). Since expected frequencies were just under 2 per constructed class interval (of length 5), we arranged the original data into larger classes (length 25). Clearly, large and small values (i.e., < 10000 or > 10000) seem to occur with about the expected frequency (Table 3). Finally, we pooled the results for 39 cases (390,000/category) and χ^2 was still not statistically significant.

RUNS TEST

Theory

Tests for uniformity do not take into account the order in which random numbers are generated. A generator which produces 100 values of .1, 100 values of .2, and 100 values of .3, etc. would pass a frequency test which considered 1000 observations and 10 classes. Thus, to supplement the frequency test, the arrangement of numbers within a sequence must be evaluated.

Two statistical tests (runs up and down, and runs above and below the mean) were used to evaluate "runs" in sequences of pseudo-random numbers. The two procedures are similar, but not identical. However, we only illustrate the results of "runs" above and below the mean procedure here. Consider the rounded uniform pseudo-random number sequence (actually generated by RND in BASIC): .4, .6, .1, .9, .7, 0, .8, .4, .2, .1, .4, .9, .8, .1, .8, .7, 1., .9, .8, .4, .2

The object of the test is to evaluate how many numbers are counted before one is encountered below the mean (i.e., in this case, below 0.5). In the sequence above, the first number is below the mean and corresponds to a run of length zero. Repeating, one value (.6) occurs before another number below the mean is encountered (a run of length one). The following 10 runs were extracted from the sequence above by repeated use of the procedure just outlined.

0 1 2 1 0 0 0 2 5 0

Using this information and the expected theoretical distribution of runs,[6] we constructed Table 4.

Table 2. Distribution of 39 Chi-Squares Calculated from 1,000,000 Random Numbers Classified in 100 Equal Categories.

Range of Chi-Square for 99 Degrees of Freedom	Probability of Observing Chi-Square	Expected (Rounded) Frequency	Observed Frequencies
less than 68.56	$0 < P \leq 0.01$	0.4	
68.57 - 72.91	$0.01 < P \leq 0.025$	0.6	1
72.92 - 76.76	$0.025 < P \leq 0.05$	1.0	1
76.77 - 81.33	$0.05 < P \leq 0.10$	2.0	
81.34 - 89.25	$0.10 < P \leq 0.25$	6.0	9
89.26 - 98.50	$0.25 < P \leq 0.50$	10.0	11
98.51 - 108.20	$0.50 < P \leq 0.75$	10.0	8
108.21 - 117.31	$0.75 < P \leq 0.90$	6.0	5
117.32 - 122.94	$0.09 < P \leq 0.95$	2.0	2
122.95 - 127.93	$0.95 < P \leq 0.975$	1.0	1
127.94 - 133.85	$0.975 < P \leq 0.99$	0.6	1
greater than 133.85	$0.99 < P \leq \infty$	0.4	
Total		40.0	39

Table 3. Frequencies of High and Low Counts Within Class Limits (Based on an Expected Value of 10,000 per Cell).

Cell numbers (Class limits)	Number of Times Classes in this Range Contained* Lowest Counts	Highest Counts	Lowest Counts <10,000	Highest Counts >10,000	Expected
0 - 5	4	1			
5^+ - 10	2	0			
10^+ - 15	3	3	11	8	9.8
15^+ - 20	2	2			
20^+ - 25	0	2			
25^+ - 30	2	2			
30^+ - 35	4	1			
35^+ - 40	1	1	10	8	9.8
40^+ - 45	2	3			
45^+ - 50	1	1			
50^+ - 55	1	3			
55^+ - 60	3	1			
60^+ - 65	4	1	12	14	9.8
65^+ - 70	3	7			
70^+ - 75	1	2			
75^+ - 80	1	1			
80^+ - 85	2	3			
85^+ - 90	0	3	6	9	9.8
90^+ - 95	2	2			
95^+ - 100	1	0			
Total	39	39			

* 2 observations/cell were expected

To evaluate the statistical significance of results such as those in Table 4, the chi-square test can be used. We calculated a value for χ^2 of 6.20, which in this case is not greater than the expected chi-square for 5 degrees of freedom.

Table 4. An Example of the Results from a "Runs" Test Below the Mean.

Length of Run	Number Observed	Number Expected
0	5	5
1	2	2.5
2	2	1.25
3	0	0.625
4	0	0.3125
5	1	0.15625

Method

We used RND to generate 200 sets of 7000 runs and recorded these results in 11 categories (runs of 0 through 9 and greater than 9). Runs both above and below the mean (0.5) were calculated, and the algorithm was terminated when a total of 7000 runs were obtained.

Results

Each calculated chi-square (200 repeats of n = 7000 runs) for tests of runs below and above the mean is tabulated in Tables 5 and 6 respectively. Since 10 of the calculated chi-squares would be expected within the 5% area on either tail of the theoretical χ^2 distribution (for 10 degrees of freedom), the observations seem to be mostly in accord with expected values. However, in the case of runs below the mean we obtained 19 values below the 5% point and for runs above the mean we obtained 15 values above the 5% point. Frequencies of observed and expected values were in accord for other cells (not above or below the 5% points), Tables 5 and 6. If these results were validated by more extensive testing, a short cyclic pattern may be indicated. In such circumstances, simulations using output from RND should be replicated several times to be sure of repeatable output.

The frequency of run components (0 to 9 and 9) that contributed more than 2 to the total calculated chi-square are tabulated in Table 7. Only statistically significant chi-squares ($P<0.05$, upper tail) were investigated. There is no evidence that any run length occurred more or less often than expected (although a few seemingly aberrant values occurred) for the cases where chi-square was statistically significant.

AUTOCORRELATION

Theory

Tests for autocorrelation examine the tendency of numbers to be correlated with others a uniform distance apart in a sequence [i.e., successive pairs or every third, fourth, fifth, etc. number can be examined (6, p. 241)]. One way to conduct the autocorrelation procedure is to use the lag product test[7]. In this test, numbers are evaluated sequentially based on the interpair distance between them. The products

$$C_k = \left[\frac{1}{N-k} \sum_{i=1}^{N-k} U_i \cdot U_{i+k} \right] - 0.25$$

are formed, where

$k = 1, 2, 3, 4, 5, \ldots 25$

lags. Where there is no correlation, C_k is normally distributed with mean of 0 and variance

$$\sigma^2 = \frac{13N - 19k}{144(N-k)^2} .$$

Table 5. Observed and Expected Frequencies of Chi-Square for 200 Cases of n = 7000 Runs Below the Mean.

Range of Chi-Square for 10 Degrees of Freedom	Probability of Observing Chi-Square in This Range	Observed Frequencies	Expected Frequencies	Observed Minus Expected Frequencies	Summary
0 - 2.2	.005	1	1	0	lower 5%
< 2.2 - 2.6	.005	1	1	0	observed 19
< 2.6 - 3.3	.015	5	3	+2	expected 10
3.3 - 3.9	.025	12	5	+7	
< 3.9 - 4.9	.05	13	10	+3	
< 4.9 - 6.7	.15	32	30	+2	middle 90%
< 6.7 - 9.3	.25	50	50	0	observed 170
< 9.3 - 12.5	.25	48	50	-2	expected 180
< 12.5 - 16.0	.15	18	30	-12	
< 16.0 - 18.3	.05	9	10	-1	
< 18.3 - 20.5	.025	4	5	-1	upper 5%
< 20.5 - 23.2	.015	5	3	+2	observed 11
< 23.2 - 25.2	.005	0	1	-1	expected 10
< 25.2 - ∞	.005	2	1	+1	

Table 6. Observed and Expected Frequencies of Chi-Square for 200 Cases of n = 7000 Runs above the Mean for 200.

Range of Chi-Square for 10 Degrees of Freedom	Probability of Observing Chi-Square in This Range	Observed Frequencies	Expected Frequencies	Observed Minus Expected Frequencies	Summary
0 - 2.2	.005	0	1	-1	lower 5%
<2.2 - 2.6	.005	1	1	0	observed 9
<2.6 - 3.3	.015	3	3	0	expected 10
<3.3 - 3.9	.025	5	5	0	
<3.9 - 4.9	.05	12	10	+2	
<4.9 - 6.7	.15	25	30	-5	middle 90%
<6.7 - 9.3	.25	55	50	+5	observed 176
<9.3 - 12.5	.25	47	50	-3	expected 180
<12.5 - 16.0	.15	23	30	-7	
<16.0 - 18.3	.05	14	10	+4	
<18.3 - 20.5	.025	8	5	+3	upper 5%
<20.5 - 23.2	.015	3	3	0	observed 15
<23.2 - 25.2	.005	3	1	+2	expected 10
<25.2 - ∞	.005	1	1	0	

Table 7. Runs Which Contributed More Than 2 to Statistically Significant Chi-Squares (Upper 5% End of the χ^2 Distribution Only) for 200 Cases of 7000 Runs Above and Below the Mean.

Runs	Length of run(s) which contributed at least 2 to a statistically significant (P<0.05) chi-square (18.3 at 10 degrees of freedom).											
	0	1	2	3	4	5	6	7	8	9	9	Total
Above the mean	3	3	3	5	4	8	3	2	7	8	3	49
Below the mean	1	4	3	3	3	3	2	3	4	5	5	34
Total	4	7	6	8	7	11	5	5	11	13	8	85

The results can be studied by computing

$$Z = \frac{C_k}{\sigma}$$

and evaluating both the statistical significance of individual C_k's and the distribution of the calculated Z's. This is a test of whether each calculated C_k is statistically different from its expected value (0). In addition, the frequency with which C_k's above and below zero occur can be investigated.

Method

We generated 4 repeats each of 25 replications of 100, 200, and 1000 numbers using BASIC-11 RND. Substantially more repeats (10) were generated for sets of 500 random numbers. We computed C_k and σ^2 for each value of k from 1 to 25 and compared the calculated Z-statistics with expected values (based on the normal distribution), evaluated the statistical significance of Z's and investigated the occurrences of C_k's above and below zero.

Results

The summarized results from 4 repeats of 25 replications of 100 random numbers each are in Table 8. The data are arranged by both number of negative C_k's and number of statistically significant Z-statistics within each of 25 lags. The summary of positives and negatives within each of the 25 lags is at the bottom of the first four columns and the total number of statistically significant ($P<0.10$; 0.05 in each tail) differences is beneath the second four, columns. These two sets of four columns and associated totals were used to construct Tables 9, 10 and 11 respectively. Results based on 200, 500, and 1000 random numbers were also summarized in a manner similar to Table 8, and are included in Tables 9-11. Finally, we used a separate algorithm to compute 100 sets of 25 C_k's based on 1000 random numbers and tabulated the results for each lag (1 to 25). This process was repeated 10, times and we calculated and tabulated the Z-statictics. The distribution of calculated Z-statistics was compared to expected values based on the normal distribution. Results are in Table 12.

Several features evident in Table 8 are supported in summary Tables 9-12. It appears (Table 8) that C_k's tend to be either mostly below or above the mean. We observed 47 cells with 20 or more C_k's below the mean and 38 cells with 20 or more C_k's above the mean. However, cells with a more even distribution of C_k's around the mean are under represented; we observed only 15 in 100 trials. This observation is substantiated in Table 9, which shows similar findings for other lengths of random number sequences. These data indicate that in most cases (over 80%) the C_k's for a sequence of 25 lags will be either mostly above or below the expected mean.

However, Chi-square was not statistically significant for the results summarized in Table 10. Thus, about the same frequency of negative and positive C_k's occurred when numbers of cells with more than half of the C_k's below the expected mean (0.25) were tabulated (e.g. fifth column of Table 8). Therefore, the predominantly positive and negative sequences (Table 9) occur with about equal frequency and probably in random order. These predominate sequences of negative or positive C_k's (within 25 lags) were not necessarily significantly different ($P<0.10$) from their expected value (0), based on the calculated Z-statistic (last 4 columns in Table 8). In addition to the predominately negative and positive C_k's, data in Table 8 (last 4 columns) imply that a majority of statistically significant Z-statistics tended to occur together in groups. However, numbers of significant differences in a column were near the expected value of 62.5. Results for all random number sequence lengths in Table 11 support this observation since an average of between 6 and 7 repeats of 25 replications contained statistically significant C_k's (for 25 lags) and between 2 and 3.25 of these contained greater than 12 statistically significant C_k's. Using a seperate algorithm, the distribution of Z's calculated for 100 sets of 1000 random numbers were compared to expected values; no aberrant behavior was observed. Results for the statistically significant ($P<0.05$) cases are tabulated in Table 12. No particular lag or position in the distribution seemed to occur more frequently than expected. However, repeats 1 and 6 seemed to exhibit unexpected characteristics most often.

We interpret the observations summarized in Tables 9-12 to imply that lag correlations (in lags of 1 to 25) are either predominately positive or negative more often than expected, and that when a statistically significant Z-statistic occurs, sometimes it is in a "clump" of similarly statistically significant Z's in a lag sequence. Both observations indicate some undesirable short-term behavior in the generated random number sequences. The length of strongly negative or positive sequences is some- times 3 or 4 repeats long (see Table 8, column 1, replications 8 to 10, 17 to 20, and 21 to 25). When this occurs with 1000 random number sequences, up to 4000 numbers may then have lag correlations predominately above or below the expected value. These data suggest that in applications where fewer then 5000 numbers are used, procedures should be repeated several times.

Table 8. Results of the Lag Product Test (For Lags of 1 to 25) Applied to 100 Sequences of 100 Random Numbers.

Replication (Each Replication Contains the Results of 25 Lag Product Tests)	Number of Lag Products (C_k's)* Which Were Below the Expected Mean (0.25) — Repeat Number				Numbers of Cells With More Than 12 Negative C_k's		Number of Significant ($P < 0.10$) Lag Product Tests (Calculated Z's) — Repeat Number			
	1	2	3	4			1	2	3	4
1	22	0	24	25	3		0	1	0	10
2	25	22	25	25	4		0	0	0	0
3	2	25	25	0	2		0	0	0	6
4	13	15	6	25	3		0	0	0	0
5	3	25	25	0	2		0	0	1	15
6	0	0	25	9	1		11	3	20	0
7	12	14	25	24	3		0	0	0	0
8	25	25	5	0	2		0	1	0	25
9	25	0	14	25	3		0	17	0	1
10	25	0	25	7	2		0	1	7	0
11	0	8	3	10	0		12	0	0	0
12	5	22	21	25	3		0	0	0	3
13	1	22	21	19	3		2	0	0	0
14	3	25	25	2	2		0	0	0	0
15	19	12	0	25	2		0	0	1	1
16	25	25	25	0	3		0	1	7	10
17	0	25	0	21	2		24	0	0	0
18	0	24	2	4	1		1	0	0	0
19	0	3	0	1	0		0	0	2	0
20	0	25	25	0	2		0	0	25	3
21	25	6	4	0	1		0	0	0	0
22	25	25	0	0	2		1	0	24	13
23	24	0	0	25	2		0	0	0	0
24	25	0	24	25	3		0	0	0	0
25	9	25	3	25	2		0	14	0	0
Number of cells with:					Totals					
20 or more negatives	10	13	13	11	47	51	Total:	51	37	87	87
20 or more positives	11	7	10	10	38		Expected:	62.5	62.5	62.5	62.5
Between 20 negative and 20 positive	4	5	2	4	15						

* Each cell contains the number of negative values calculated using lag product test (see text) for each lag from 1 to 25.

Table 9. Frequency of Occurrence of Cells With More Than 20 C_k's Below the Mean, Cells With More Than 20 C_k's Above the Mean, and Other Cells (Lags from 1 to 25) for Random Number Sequences of Various Lengths

Length of Random Number Sequences	Number of Repeats of 25 Replications	Number of Cells Evaluated	Number of Cells With 20 or More Negatives		Number of Cells With 20 or More Positives		Other Cells Less than 20 Negatives or Positives	
			(n)	(%)*	(n)	(%)	(n)	(%)
100	4	100	47	47	38	38	15	15
200	4	100	48	48	35	35	17	17
500	10	250	108	43	105	42	37	15
1000	4	100	40	40	41	41	19	19

* Percent of Total Number of Cells; 100 or 250.

Table 10. Frequency of Cells With More Than Half of C_k's Below the Expected Mean Versus Expected Within 4 Repeats of Each of 25 Replications for Random Number Sequences of Various Lengths.

Length of Random Number Sequence	Observed Frequency of Cells with >12 Negative C_k's in 4 Repeats of 25 Lags (k)					
	0	1	2	3	4	
100	2	3	11	8	1	
200	1	3	14	6	1	
500 (Repeats 1-4)	2	8	9	5	1	
500 (Repeats 5-8)	2	6	9	6	2	
1000	0	4	16	5	0	
Total Observed	7	24	59	30	5	χ^2 = 6.68 n.s.
Total Expected*	7.8	31.3	46.9	31.1	7.8	

* Calculated; based on the binomial distribution where n = 25, P = 0.50

Table 11. Means and Standard Deviations of Number of Significant Differences ($P < 0.10$, based on Z-statistics) and Numbers of Cells with Greater than 12 Significant Differences for Random Number Sequences of Various Lengths.

Length of Random Number Sequence	Numbers of Repeats of 25 Replications	Number of Significant ($P < 0.10$) Differences Found in Each Repeat		Number of Cells Where >12 of 25 Lags (1 to 25) were Significant ($P < 0.10$)	
		Mean	Standard Deviation	Mean	Standard Deviation
100	4	7.75	1.71	2.50	0.58
200	4	7.50	2.08	2.25	0.96
500	10	6.15	2.07	2.00	1.25
1000	4	7.25	2.06	3.25	1.50

Table 12. Distribution of the Z - Statistic Calculated From C_k's and Standard Deviations of the Lag Product Test (Based on 10 Repeats of 1000 Random Numbers).

Repeat Number	Length of Lag (k)	Calculated Chi-Square	Distribution of Calculated Z - Statistics for 100 sets of 1000 Random Numbers							
			≥ 1.645	<1.645 >1.1	<1.1 >0.55	<0.55 >0	<0 >-0.55	<-0.55 >-1.1	<-1.1 >-1.645	<-1.645
1	2	19.8	7	3	16	30	20*	7*	15	2
1	13	16.4	6	4	14	36*	15	14	8	3
1	21	18.9	8	6	10	32*	21	6*	13	4
1	25	18.7	6	7	16	32*	13	8	15*	3
2	5	15 3	3	19*	15	15	21	15	8*	4
6	3	16.2	10*	5	13	13	29	16	12	2
6	5	16.5	8*	9	13	11	29	12	15	3
6	8	21.6	11	4	14	19*	19	15	17*	1
7	19	14.3	9*	7	8	26	26	10*	6	8
10	1	16.5	3	9	14*	15	22	29	5	3
	Expected Frequencies		5	8.6	15.6	20.9	20.9	15.6	8.6	5

* Contributed more than 4 to a statistically significant ($P<0.05$, $\chi^2 = 14.1$, df = 7) chi-square.

SUMMARY

The DEC algorithm for computing pseudo random numbers is essentially the same in BASIC-11, BASIC-PLUS-2, FORTRAN IV, and FORTRAN IV-PLUS. The period is about 5×10^8. An irregular "pseudo-period" was observed 15 times before there was an exact repeat of an original sequence. This pseudo-period may need further investigation for some applications where pseudo-random numbers are used. The statistically significant differences (between the frequencies observed and those expected from a uniform distribution) were within the expected range. We observed more statistically significant runs than expected, which may suggest that the algorithm produces short-term runs. The results of the lag-product test again implied short-term runs (up to 4000 numbers) and suggests that statistically significant autocorrelations occur in "clumps."

In general, the data seem to support the view that the DEC pseudo-random number generator will produce a reasonable series of numbers for many common applications. Repeated runs may be necessary to confirm results in applications where only a few numbers (less than 5000) are used. Users of extremely long runs of random numbers should be aware of possible problems associated with the "pseudo-period."

ACKNOWLEDGEMENT

We thank Bill Parm of Digital for assistance with the algorithms and Ms. Connie Connally for help in editorial matters.

REFERENCES

1. D. E. Knuth. 1973. The Art of Computer Programming, Vol. 2. Addison-Wesley, Menlo Park, California.

2. P. T. Brady. Random Number Generator for the PDP-5/8. DECUS Program Library No. 8-25.

3. G. A. Griffith. Pseduo Random Number Generator For use with FOCAL. DECUS Program Library No. FOCAL 8-1.

4. G. Marsaglia. 1970. Regularities in Congruential Random Number Generators. Numer. Math. 16:8-10.

5. D. M. MacLaren and G. Marsaglia. 1965. Uniform Random Number Generators. J. Assn. Compt. Mach. 12:83-89.

6. J. W. Schmidt and R. E. Taylor. 1970. Simulation Analysis of Industrial Systems. Irwin, Homewood, Illinois.

7. R. O. Gilbert. 1973. An Evaluation of Four Pseudo-Random Number Generators, BNWL-1743, Battelle, Pacific Northwest Laboratories, Richland, Washington.

NET — A POWERFUL FILE-TRANSFER FACILITY*

R. D. Burris, C. E. Hammons, and C. O. Kemper
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37830

ABSTRACT

The NET command and system provide access to a network that includes DECsystem-10s, PDP-11s, PDP-12s, PDP-8s, IBM System/360s, and a CDC 7600. A DECNET-style command syntax is used. Full wildcard capability is provided for transmission, retrievals, and third-party transfers. The NET command was made to interface to existing programs transmitting to the IBM and CDC equipment. For transfers between DECsystem-10s, a spooler was written to support file transfer through asynchronous (teletype) lines, a DAS78 front end, or a DL10 link to a PDP-11/45. The PDP-11/45 is itself a node in the network. Users on many nodes can direct files to the 11/45 for storage, printing, plotting, or routing to some other node. It is between DECsystem-10s that the most powerful and free interchange of data is possible. Possible source devices include all directory devices and ersatz devices. Destination devices include all devices and queues that do not require operator intervention.

INTRODUCTION

General

Data processing centers are frequently dedicated to the equipment of one manufacturer. The advantage of such dedication can best be shown by contradiction — when the software of several vendors is present, the installation users must know several ways to perform every task. The investment in education, the number of mistakes provoked by this situation, the concomitant losses in productivity, and the decrease in user morale are usually sufficient to convince management that the multiple vendor approach is unattractive. With the recent developments in networking and distributed processing, however, it has become much more attractive to have the capability to use such networks and the facilities of other computer centers that may not have become dedicated to the same vendor. This paper describes the successful merger of three such networks, which use equipment from International Business Machines (IBM), Control Data Corporation (CDC), Digital Equipment Corporation (DEC), and CRAY.

Contents

We consider the following topics:

1. the goals, structure, and implementation of the system;
2. the network as it appears to the user;
3. a special monitor interface between the DECsystem-10 and a DEC PDP-11/45;
4. The PDP-11/45 networking system structure and functions.

SYSTEM DESIGN

General

Figure 1 depicts the networks under consideration. The Fusion Energy Division (FED) and the Computer Sciences Division (CSD) nodes, both DECsystem-10s, serve as portals to the Magnetic Fusion Energy (MFENET) and CSD networks, respectively. The PDP-11/45 is the central node in the data gathering network of the FED. The project described is the integration of these networks into one supernetwork and the provision of a means of effective access to that network.

Existing facilities

Personnel of MFENET provide and support links between DECsystem-10s in that network and the CRAY-1. Personnel of CSD support links between the CSD DECsystem-10 and the IBM System/360 computers. Rather than attempt to replace or duplicate the work of those groups, the development of the supernetwork incorporates their programs with only minor modifications. Figure 2 shows the desired software structure of the supernetwork after support of the FED to PDP-11/45 link and the 11/45 to CSD links and after the creation of a consistent user interface.

Design parameters

The system to be developed must adhere to the following guidelines.

*Research sponsored by the Office of Fusion Energy (ETM), U.S. Department of Energy under contract W-7405-eng-26 with the Union Carbide Corporation.

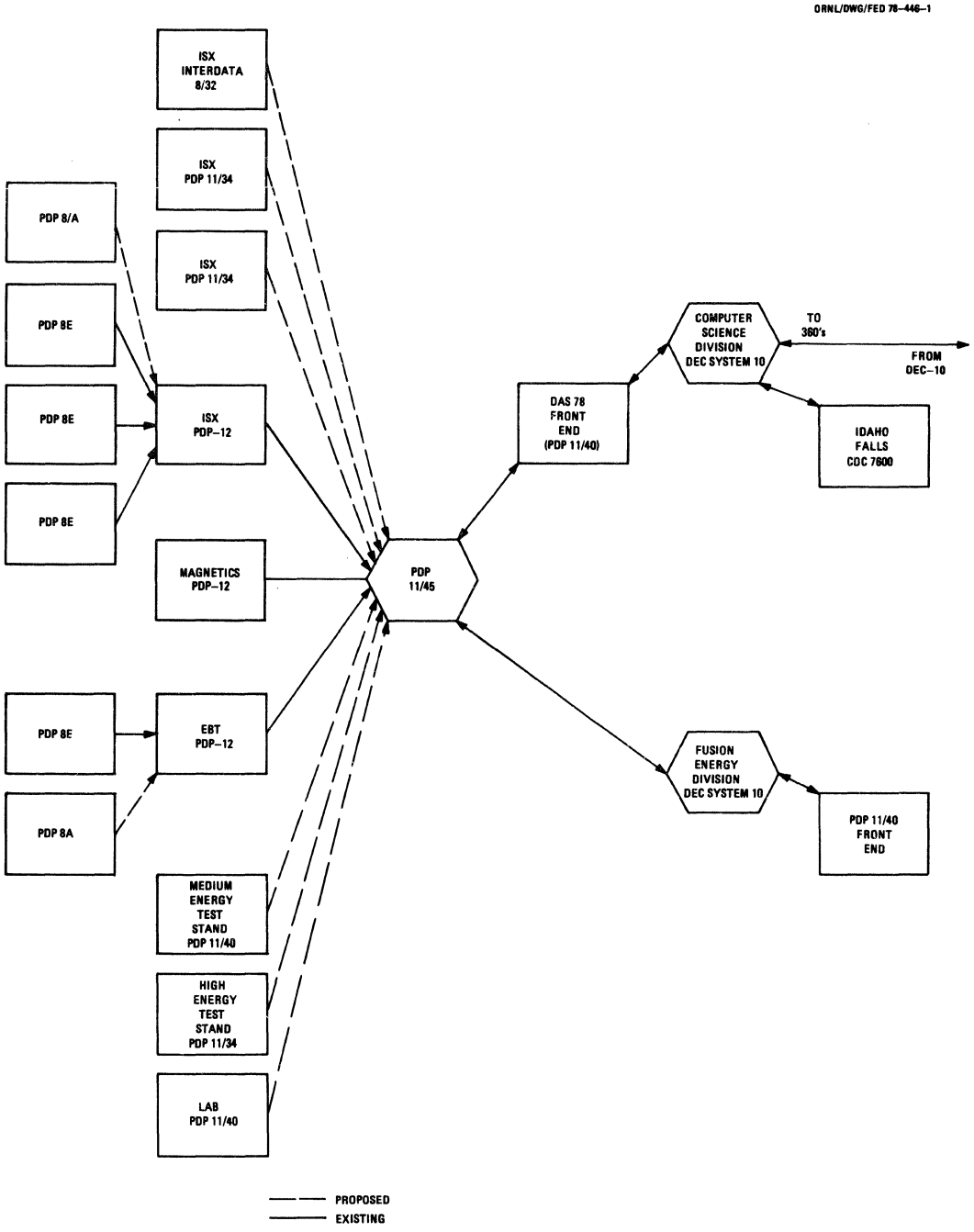
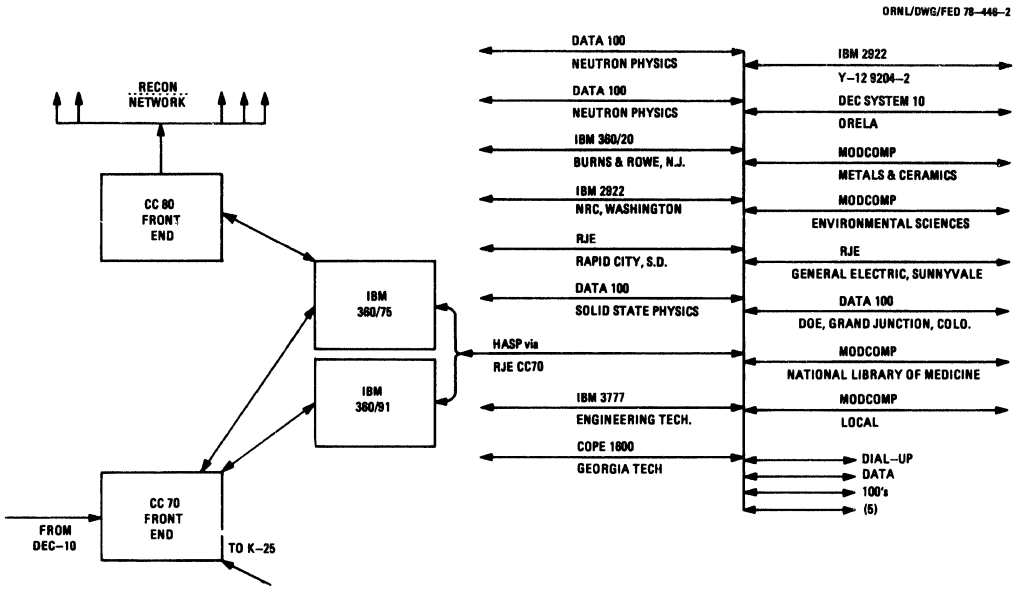
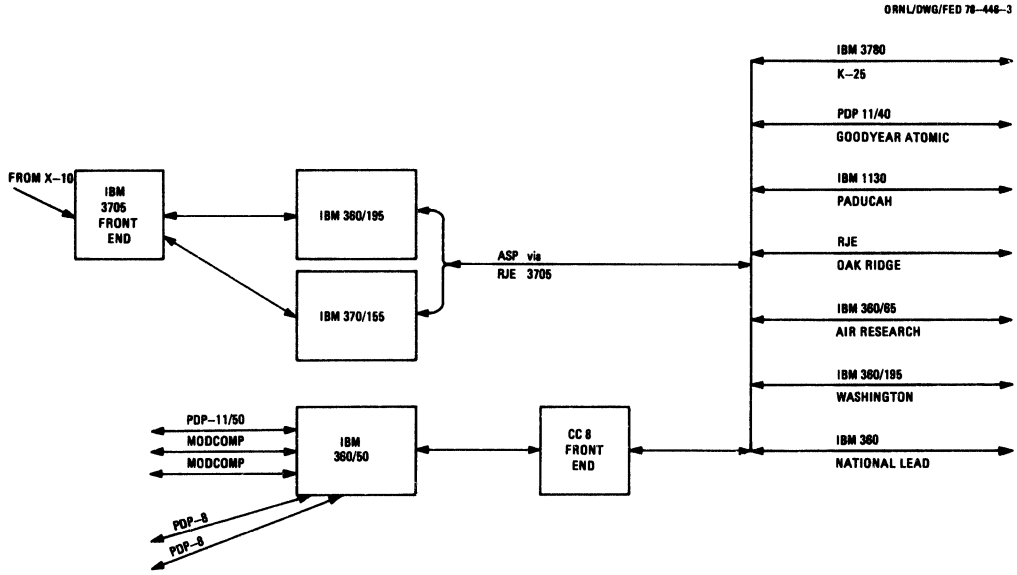

ORNL/DWG/FED 78-446-1
ISX INTERDATA 8/32
ISX PDP 11/34
ISX PDP 11/34
PDP 8/A
PDP 8E
PDP 8E
PDP 8E
ISX PDP-12
MAGNETICS PDP-12
PDP 8E
PDP 8A
EBT PDP-12
MEDIUM ENERGY TEST STAND PDP 11/40
HIGH ENERGY TEST STAND PDP 11/34
LAB PDP 11/40
PDP 11/45
DAS 78 FRONT END (PDP 11/40)
COMPUTER SCIENCE DIVISION DEC SYSTEM 10
TO 360's
FROM DEC-10
IDAHO FALLS CDC 7600
FUSION ENERGY DIVISION DEC SYSTEM 10
PDP 11/40 FRONT END
PROPOSED
EXISTING

Fig. 1(a)


ORNL/DWG/FED 78-446-2
RECON NETWORK
CC 80 FRONT END
IBM 360/75
IBM 360/91
CC 70 FRONT END
FROM DEC-10
TO K-25
HASP via RJE CC70
DATA 100
NEUTRON PHYSICS
DATA 100
NEUTRON PHYSICS
IBM 360/20
BURNS & ROWE, N.J.
IBM 2922
NRC, WASHINGTON
RJE
RAPID CITY, S.D.
DATA 100
SOLID STATE PHYSICS
IBM 3777
ENGINEERING TECH.
COPE 1600
GEORGIA TECH
IBM 2922
Y-12 9204-2
DEC SYSTEM 10
ORELA
MODCOMP
METALS & CERAMICS
MODCOMP
ENVIRONMENTAL SCIENCES
RJE
GENERAL ELECTRIC, SUNNYVALE
DATA 100
DOE, GRAND JUNCTION, COLO.
MODCOMP
NATIONAL LIBRARY OF MEDICINE
MODCOMP
LOCAL
DIAL-UP DATA 100's (5)


ORNL/DWG/FED 78-446-3
FROM X-10
IBM 3705 FRONT END
IBM 360/195
IBM 370/155
ASP via RJE 3705
PDP-11/50
MODCOMP
MODCOMP
IBM 360/50
CC 8 FRONT END
PDP-8
PDP-8
IBM 3780
K-25
PDP 11/40
GOODYEAR ATOMIC
IBM 1130
PADUCAH
RJE
OAK RIDGE
IBM 360/65
AIR RESEARCH
IBM 360/195
WASHINGTON
IBM 360
NATIONAL LEAD

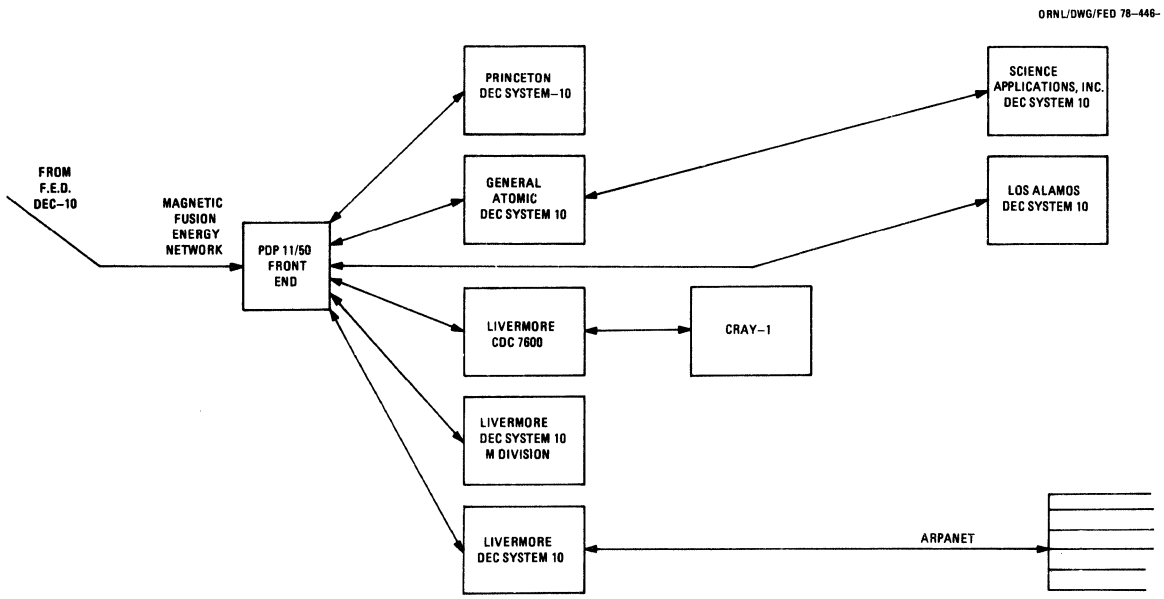

ORNL/DWG/FED 78-446-
FROM F.E.D. DEC-10
MAGNETIC FUSION ENERGY NETWORK
PDP 11/50 FRONT END
PRINCETON DEC SYSTEM-10
GENERAL ATOMIC DEC SYSTEM 10
LIVERMORE CDC 7600
CRAY-1
LIVERMORE DEC SYSTEM 10 M DIVISION
LIVERMORE DEC SYSTEM 10
SCIENCE APPLICATIONS, INC. DEC SYSTEM 10
LOS ALAMOS DEC SYSTEM 10
ARPANET

Fig. 1(b)

OVERALL NETWORK LOGIC

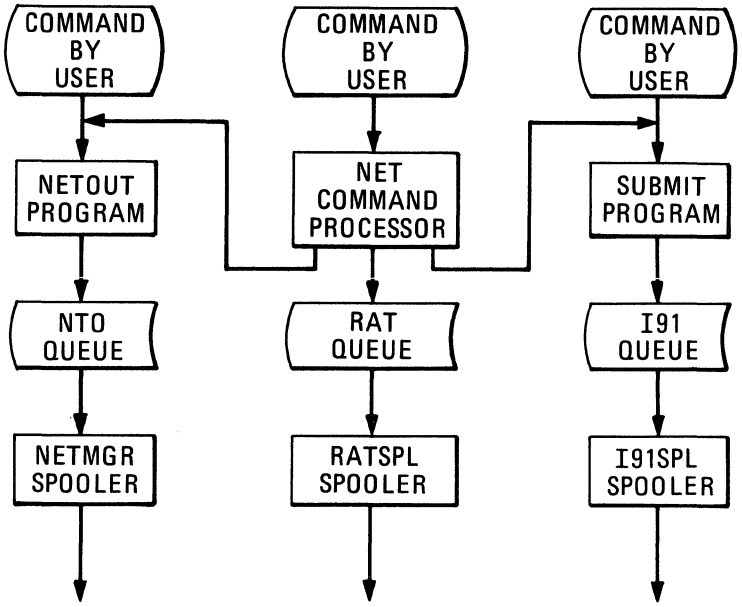


Fig. 2

1. A simple command should provide access to all nodes of the network.
2. The syntax of that command should be familiar or easy to learn.
3. The system should provide as much power to the user as possible. Desirable options include remote printing, plotting, punching, and batch, as well as simple disk-disk transfers.
4. The command should provide wildcard file specification (i.e., the user should be able to specify that all files of a particular type should be transferred and to specify that request as if it were a single request).
5. Existing software should be used wherever possible and modified as little as possible for ease of debugging and maintenance.
6. The software should be easily adapted to different hardware/software so that upgrading of the system can be accomplished as easily as possible.
7. The network should not impact system security at any site.

System structure

In our networks, each link has the same general structure (see Fig. 2) -- a user interface program, which places files to be transmitted in a queue, and a queue processing program to remove the files from the queue and transmit them. The reason this structure is followed is to disassociate the user from waiting for the transaction to be completed (consider what happens when the link is down and the user has to wait) and to keep the spooling program as small as possible by handling command decoding in a separate program.

The new facility was to include interfaces to existing software. Rather than write a user interface that would support all the old syntaxes and functions as well as the new ones, the new facility was made capable of translating the new command string into the syntaxes of the old commands and then invoking the old user interfaces. By taking this approach, the old spoolers need no changes because they were fed by the same user interfaces and revisions of those interfaces could be supported easily.

NETWORK-USER INTERFACE

General

As stated previously, providing an interface featuring ease of use and great tolerance to user errors is considered extremely important. These characteristics are provided by choice of a familiar syntax for the command, by the provision of effective defaults, and by careful attention to error correction facilities.

Command syntax

The syntax chosen for the command was the DECsystem-10 Peripheral Interchange Program (PIP) syntax. The reasons for this choice include familiarity, ease of use, and the availability of DEC's SCAN and WILD routines for use in the parsing of the command string. The format for that syntax is

DESTINATION = SOURCE

where each side of the command involves one or more file specifications. To implement the

network facility, an additional field is included on each side of the command, namely, a node specifier. The node is delimited in the DECNET style with the underscore (back arrow) character. The resultant general syntax is

NODE←FILE.A,FILE.B,...=NODE←FILE.X,FILE.Y,...

where FILE.A, etc., is a specification of a file. The format for a complete file specification is

DEV:FILE.EXT[Proj,Prog]/SWITCH/SWITCH

where
DEV is the device on which the file can be found;
FILE is the name of the file;
EXT is the extension (data type) of the file;
[Proj,Prog] is the designator of the user or the portion of the disk in which the file may be found;
/SWITCH provides additional information about the file.

Each file specifier (filespec) can have all the above information specified, or several of the fields can be specified once per side and apply to all filespecs on that side. For some fields, if no source for the data exists, defaults are assumed. An example of the complete command is

.NET X10←DSKB;DATA.A[200,143]=FED←DSKC:INFO.CD [200,21455]

Defaults

The provision of defaults makes the task of the user much easier. It is usually possible for a user to effect file transfer with as few as two entities — remote node and source file specification. The defaults assumed are listed below.

Field	Default
Node	The node to which the user is logically connected
Destination device	DSK:
Destination file name	Same as source file name
Remote PPN	The mapping of the user's logged-in PPN will be used, if such a mapping exists
Source device	DSK:
Local PPN	The PPN under which the user is logged into the system.

A user could effect a file transfer by typing

.NET X10_ = FILE.ABC

Error correction

In specifying the transmission of several files, the user is likely to make a typing error. To preclude the requirement to reenter the entire command, the user is prompted for correction of such errors as the following:

1. null device,
2. wildcard device,
3. null switch,
4. unknown switch (misspelled switch),
5. ambiguous switch (misspelled or too abbreviated),
6. omitted switch value,
7. unknown switch value (misspelled),
8. ambiguous switch value (misspelled or too abbreviated),
9. zero or excessive-length project or programmer number,
10. no file satisfying a wildcard specification.

Each of these errors is normally treated as fatal by SCAN or WILD, but special intercepts of their monitor returns have been implemented.

Functions available

General. The general function of the NET command is to break the command string specified by the user into one or more command strings acceptable to (i.e., in the syntax of) the remote computer. The strings generated are saved in a disk file and passed to the spooler via the queueing structure.

These generated strings

1. tell the local spooler what function to perform,
2. are transmitted to the remote computer,
3. tell the remote spooler what function to perform.

Three primary functions are provided in this network: file transmission, file retrieval, and third-party transfers. File transmission is the transfer of a file from the node to which the user is logically connected to some other node in the network. File retrieval is the transfer of a file from some other node in the network to the node to which the user is connected. Third-party transfers involve the transfer of a file from one node to another when neither of the nodes involved is the one to which the user is connected.

Transmission. In simple file transmission, the source node is the local computer. The command string generated by the NET command might be a hybrid form, depending on the destination node type; that is, if the destination is the PDP-11/45, then the destination side of the generated string will be in RSX-11D format while the source is in DECsystem-10 format. The spooler will find a file containing the command string(s) defining the transmission and will look in a staging area for the file to be sent. The command string will then be sent to the remote node and will be followed by the data file. Upon receipt at the remote node, the command string (always at the first block sent) will be parsed to determine disposition of the data.

If the destination node is a DECsystem-10, several destination "devices" are possible. In addition to the usual disk device, the user may specify "INP", "LPT", and any other legal queue. Thus the user may submit jobs to the batch queue on the remote 10 or cause the file to be printed or plotted, etc. In addition, because communications queues are legal destinations, the file can be submitted to the queue for the IBM System/360 or for the MFENET.

Examples of file transmission follow:

.NET←X10 FILE.A[200,33]=FED←FILE.A[4,5]

which sends FILE.A from [4,5] on the FED node to [200,33] on the CSD node;

.NET A7600←/U=X10←FILE.CTL[4,5]

which sends FILE.CTL from [4,5] on the CSD node to the user's area of the MFENET CDC 7600.

Retrieval. When the NET command processes a command in which the local node is the destination, it still builds strings and saves them for the spooler. In this case, however, no data will be copied to the staging area (because the source files are on the remote computer). Furthermore, it will be the source side of the command that will be translated to the syntax of the remote computer.

When the spooler picks up the file containing such command strings and recognizes that they are retrieval requests, it sends the command strings alone (without data) to the computer designated as the source node. That computer, when it receives a command string in which it is the source, simply places the request into the queue for the communications spooler with a flag set to indicate that the data file is not in the staging area. For example,

.NET =X10←FILE.AA[200,3]

takes the file FILE.AA from area [200,3] on the X10 node and transfers it to the user's area on the local node.

Third-party transfers and wildcard specifications. Third-party transfers refer to transmissions in which the local node does not participate as either source or destination. Wildcard specifications refer to the use of a wildcard character in place of one or more characters of a file name or PPN. When third-party transfers or wildcard retrieval requests are specified, processing must be modified somewhat.

Recall that the handling of the IBM and MFENET networks required the running of another program to provide the interface to their spoolers. If such a request were received by the spooler on one of the DECsystem-10s, it would have to overlay itself to invoke such a program, thereby terminating the communications link. If a wildcard retrieval request were received by the spooler, the spooler would have to include the SCAN and WILD programs to resolve the required file names, and the spooler thus would be much larger.

Rather than take either of these actions, these facilities have been provided in another manner. When the NET command recognizes such a situation, the original net command is saved by itself in a file in the staging area, and another command string is invented that submits the saved command as a job for the batch queue on the remote DECsystem-10.

The result of this procedure is that the command string entered by the user is actually executed on the remote computer in the batch mode. The NET command at the remote computer then takes care of wildcards or the invocation of the interfaces to the other networks.

Examples of third-party transmissions and wildcard file retrieval follow:

.NET RSX←LPT:=X10←ABC.LPT[33,45]

when the user is logged into the FED node causes the printing of the file ABC.LPT from area [33,45] of the X10 node on the PDP-11/45 impact printer;

.NET =X10←FILE.*

when the user is logged into the FED node causes the retrieval of all files that have file name "FILE" from that user's area on the X10 node.

Auxiliary functions
With the establishment of this link, several other functions became possible. One of them was a program enabling the user to execute the DIRECT command on the remote DECsystem-10. The user is prompted for the DIRECT command to be executed, and the response is submitted to batch on the remote computer. A whole family of applications could be built on this concept.

Another very powerful facility interfaces the DECsystem-10 to the plotters run from the IBM System/360s of CSD. A 360 job that can do appropriate conversions is created on the 10 and queued with the data for transmission to the 360s. By this ploy and using a special PLOT command developed at FED, the plotters of the IBM System/360 are just as accessible to users logged into the DECsystem-10 as are graphics terminals physically linked to that 10.

MONITOR INTERFACE

General information and description
The FED DECsystem-10 and the data gathering computers of the Fusion Energy Division are all nodes to the FED PDP-11/45. The link of concern here is the one between the FED 10 and the 11/45. The hardware connecting these computers is a Data Link Control unit (DL10), which permits direct access to the memory of the DECsystem-10 by the PDP-11/45. The DL10 was chosen (rather than a DA28 Interprocessor Buffer or the DN87 Communications Processor) because of the high speed shared memory segment window available via the DL10 and because of the nature of data exchange offered to the two systems (TOPS10 and RSX-11D).

TOPS10 implementation
We wanted to use a service program that was compatible with the DAS78/DN61 class of devices, but the DEC service program for this type of device placed unacceptable constraints on the subwindows supported by such processors. Therefore, we developed our own device-service program, which permits the logical subwindows to be implemented and supported as generalized I/O devices.

The DEC service program (D78INT) was replaced by our version, thus making monitor generation quite easy and permitting compatibility with future monitor releases (as long as there are no major

changes in DEC philosophy). The DL10 window is used in normal fashion and is referenced via mnemonics provided by TOPS10 for DAS78 operations (XXI0, XX00, XXI1, etc.). A feature not offered by DAS78 remote job entry software permits the job controlling transmission in the DECsystem-10 to select program units in the front end system and then communicate with these units under minimal constraints. The content and protocol of these front end programs are at the discretion of the DECsystem-10 program. The service program will support all four DL10 windows.

PDP-11/45

General description

The Fusion Energy Division's communications system consists of a PDP-11/45 central processor, memory, fixed and moving head disk, line printer, electrostatic printer/plotter, and a variety of communications interfaces. Operating under Digital Equipment Corporation's real-time multitasking executive RSX-11D, the system provides a file transfer interface between all systems connected to it, makes its own peripherals available to all systems connected to it, and provides a software development environment for other PDP-11 systems.

Communications elements

The communications system contains a wide variety of synchronous, asynchronous, and direct-memory-access (DMA) links. Support is provided for a wide variety of standard and special purpose protocols having widely differing throughput and data format characteristics.

Software elements

Aside from the executive and its system tasks, which have not been locally modified, the software consists of the following:

1. link driver, receiver, and transmitter tasks for each external system link;
2. driver and queue-server (despooler) tasks for each local, nonfile structured peripheral;
3. a spooler/conversion task, SPOOL;
4. a despooling scheduler task, SCHDSP;
5. an operator link command task, ...QUE.

Transmission processing

The protocol for all file transmissions in the network requires an ASCII command string describing the operations to be performed, optionally followed by data on which to operate. Such transmissions are processed by the PDP/45 as described in the following sections.

Link driver and receiver tasks. The link driving tasks control the communications hardware interfaces and arbitrate whatever protocol governs the specific link. When a remote system indicates a readiness to transmit, the receiving link driver requests the loading and execution of the appropriate receiver task, which will remain active until the data set has been completely received. The receiver task writes the incoming data into a file in the system's spooling area. It also creates a description of the spool file to pass to the spooling program.

Spooling program. The command string, which precedes the data set, and the description of the spool file created by the receiver task are now passed to the spooling program. This task performs any internal conversion necessitated by link command switches or by differences between the source and destination nodes and then moves the result into a transmission queue in the system's queueing area. The spooler then makes an entry in the appropriate transmission queue, which contains the link command itself and a description of the data set.

Note that the queueing structure used by the communications system is that which is implied in the RSX file system. The destination node is used as the queue name and as the file name extension. The file version number serves to order the queue. Facilities provided by the file system allow adding to, picking from, and scanning of one or more FIFO queues quite simply.

The reliability of the system is closely related to the file structuring as well. Because the system maintains all directory information on the volume to which it pertains, the queue entries and data sets are highly protected from loss through hardware failure (and the executive and file system software have never failed). Furthermore, the operator may redirect the spooling and queueing pseudo-devices to and from actual disk devices at will, providing great flexibility in system maintenance and holding data in case of link failure.

Despooling and transmission. Despooling may be initiated either by scheduling or by a request from the remote system. During the despooling process, the link command and the data set are taken from the queueing area by the despooler and passed to the link driver. As before, the link driver arbitrates the protocol and interfaces the software and the device hardware.

Exceptions. There are several important exceptions to this general flow. For example, several external systems service data collection applications only. Links serving such systems support only transmission from that system into the PDP-11/45, so no transmission queues or despooling tasks exist in the PDP-11/45 for such links.

The local printer and plotter peripherals, while considered output links for the purposes of spooling, cannot receive files from external nodes and thus do not have receiver tasks defined.

Finally, a link command may not have an associated data set, as is the case in the file retrieval and third-party requests. The only software component involved in this case is the spooler, which handles the routing.

Operator command task

The operator link command task allows the system operator to enter the link commands to the spooler as if they came from an external node. Thus the operator command task is equivalent to a receiver task with the console substituted for the external link. Data sets to be transmitted (if any) may reside on any file structure.

Conclusion

The many levels of hardware and software priority, the separation of functions, and the file structure processing provided by RSX-11D permit the combination of diverse components into a single, highly reliable unit tuned to communications effectiveness.

CONCLUDING REMARKS

The noteworthy features of this network include easy, powerful, and effective access, considerable off-loading of the DECsystem-10s to a PDP-11/45, and utilization of software from several sources to provide the unification of three networks. The network provides an extremely useful tool to users of the data gathering subnetwork or of the DECsystem-10s.

ACKNOWLEDGMENTS

Mr. Duane Winkler, Computer Sciences Division of Union Carbide Corporation Nuclear Division, wrote the software supporting the link between the CSD DECsystem-10 and the CSD IBM System/360 computers. Betty Shuttleworth, Barry Howard, and Peter Pearson wrote the software supporting the linking of DECsystem-10s to the MFENET. Jack Francis, Oliver Yonts, and Jay Reynolds of the Fusion Energy Division of Oak Ridge National Laboratory wrote software supporting links between the PDP-11/45 and data gathering PDP-12s and PDP-8s. Reid Gryder made many valuable suggestions. Ms. Rose Ann Pemberton prepared the manuscript. Our thanks to all these people.

POLYNOMIAL OF DEGREE N-1
FROM N DATA POINTS

Gerald Roux
The Boeing Company
Seattle, Washington

ABSTRACT

A method is presented to derive a polynomial of degree N-1 which passes through N data points. An algorithm implements the method of divided differences to obtain the polynomial.

Introduction

Nonlinear relationships abound in nature. The input/output functions of processes, be they chemical, electrical or mechanical are usually nonlinear. The ability to represent these nonlinear functions mathematically with a known degree of accuracy would often simplify data processing. lists of empirical data could be expressed as one mathematical funtion requiring less space to store and an easy means of data manipulation. Interpolation between discrete data points would be simplified when using a continuous representation of a field of data points.

Continuous nonperiodic data can be approximated by a series of the form

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_nx^n \qquad (1)$$

A function can be approximated as closely as desired by increasing the number of terms. an infinite series is needed to exactly represent a continuous nonperiodic function. However, n points on any continuous funtion can be exactly represented by a polynomial of degree n-1.

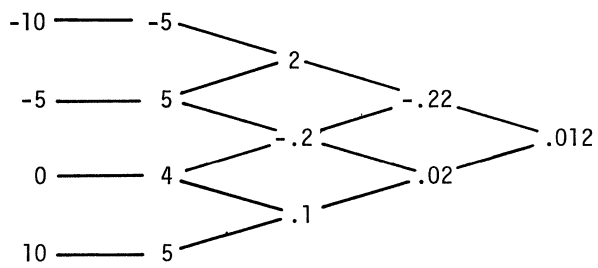
Method Of Divided Differences

Equation (1) is defined by the coefficients. the value of the coefficients can be obtained by the method of divided differences and the application of Newtons interpolation formula[1]. Given n data points (x_i , y_i) the polynomial is defined by

$$y = a_0 + a_1(X-x_1) + a_2(X-x_1)(X-x_2) + \cdots$$

$$+ a_n(X-x_i)(X-x_2)\cdots(X-x_{n-1}) \qquad (2)$$

where a_0 through a_n are the coefficients obtained from an array of divided differences. To form the array put the given data points (x_i , y_i) into two columns. Generate n-1 more columns with numbers at intersecting diagonals from the given data pairs as follows: Divide the difference of the two numbers just above and below the diagonal intersection and immediately to the left by the difference of the x's in the diagonals through the intersection. An example will illustrate the process. given 4 data points (-10,-5) , (-5,5) , (0,4) , (10,5) the following divided differences array would result.



The a_i values are along the top diagonal. Values for a_0 through a_3 are -5 , 2 , -.22 and .012.

Inserting the a_i and x_i values into equation 2:

$$y = -5+2(X+10)-.22(X+10)(X+5)+.012(X+10)(X+5)(X+0)$$

Multiplying and combining terms:

$$y = .012x^3 - .04x^2 - .7x + 4$$

The method yealds an exact polynomial for the points given. the fit to any continuous function can be made as good as the user chooses by increasing the number of points used and therefor the degree of the resulting polynomial.

Implementation

A program to derive the polynomial which will pass through any number of points using the method of divided differences has been written in BASIC. The program requires only 350 words of memory to implement and an additional 186 words to solve a 10th degree polynomial. the program listing is shown below. Program flow is as follows. The (x_i,y_i) data points are loaded into array AA(0,i) and AA(1,i) locations in line 110. The divided differences array is then developed from these (x_i,y_i) points in program lines 112 through 122 using nested FOR loops to perform successive divisions of the differences. The completed differences array for the given points is

10	-5	2	-.22	.012
-5	5	-.2	.02	0
0	4	.1	0	0
10	5	0	0	0

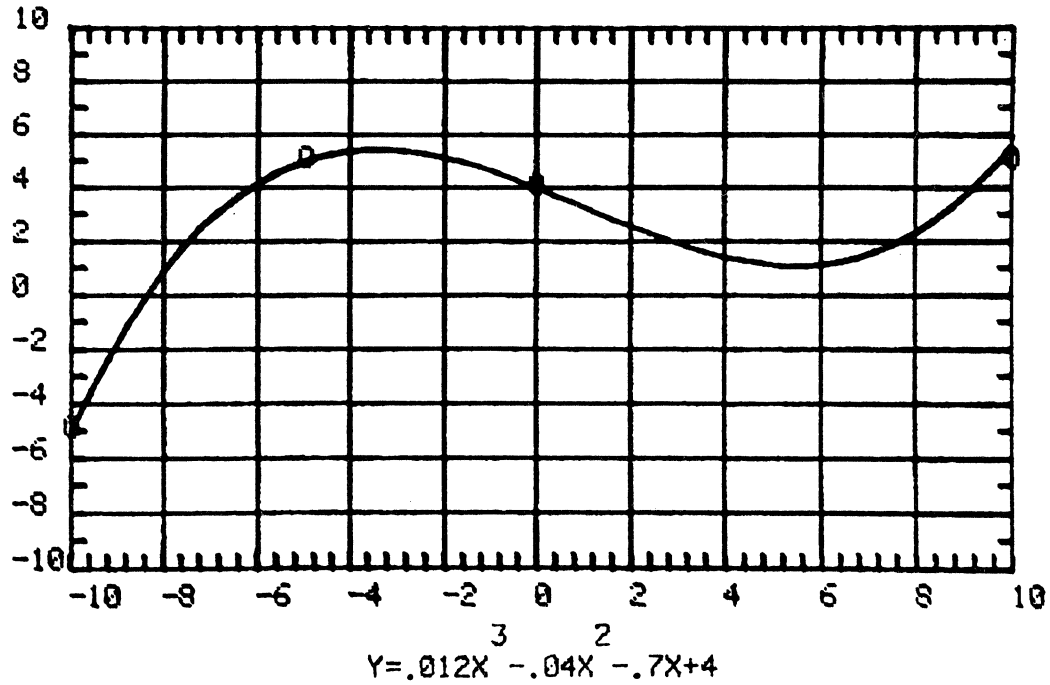
The a_i values for equation 2 are in AA(i+1,0) and the x_i values for equation 2 are in AA(0,i). Array AA therefore contains all the information necessary to complete the desired polynomial. Lines 226 through 236 assemble the polynomial.
The locations in working array BB and array CC correspond to one greater than the exponential value of X in the final polynomial. The final content of each location in array CC is the coefficient value. That is, the content of CC(4) is the coefficient of X^{4-1}.

Successive multiplications of the $(X-x_i)$ factors are performed in line 132 and the results are summed into the appropriate locations of working array BB. Line 128 then multiplies array BB by the a_i value in AA(i+1,0) and sums the result into array CC. After n iterations the final coefficient values for X^0 through X^{n-1} are in CC(0) through CC(n-1). The polynomial is then read from array CC by lines 140 through 146. The example results are shown below.

Reference

(1) G.A.Korn and T.M.Korn, Mathematical Handbook for Scientists and Engineers, 2 nd. ed. McGraw-Hill, 1969, p. 747

```
100 REMARK    PROGRAM TO GENERATE POLYNOMIAL TO PASS THROUGH N POINTS
              USING THE METHOD OF DIVIDED DIFFERENCES.
101 RELEASE VARIABLES
102 PRINT "HOW MANY POINTS ARE THERE?":INPUT N
104 DIMENSION AA(N,N-1),BB(N),CC(N)
106 PRINT "LIST EACH POINT (X,Y)"
108 FOR I=0 TO N-1
110 INPUT AA(0,I),AA(1,I)
112 NEXT I
114 FOR S=1 TO N-1
116 FOR I=0 TO N-1-S
118 LET AA(S+1,I)=(AA(S,I+1)-AA(S,I))/(AA(0,S+I)-AA(0,I))
120 NEXT I
122 NEXT S
124 LET BB(1)=1
126 FOR J=1 TO N
128 LET R=AA(J,0):LET CC=CC+(R*BB)
130 FOR I=N TO 1 STEP -1
132 LET Q=0-AA(0,J-1):LET BB(I)=((Q*BB(I))+BB(I-1))
134 NEXT I
136 NEXT J
138 PRINT :PRINT "THE POLYNOMIAL FOR THE   ";N;"  POINTS GIVEN IS:"
139 PRINT :PRINT :PRINT
140 FOR I=1 TO N
141 IF CC(N-I+1)>0:GOTO 144
142 PRINT #12*(I-1);CC(N-I+1);"X";"^K";N-I:GOTO 146
144 PRINT #12*(I-1);"+";CC(N-I+1);"X";"^K";N-I
146 NEXT I
148 STOP
```



$$Y=.012X^3-.04X^2-.7X+4$$

ATROPOS - A VERSATILE DATA ACQUISITION AND ANALYSIS SYSTEM†

C.A. Logg and R.L.A. Cottrell
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

ABSTRACT

At SLAC we frequently need versatile, portable, rugged, and compact test and control modules to use in the development and testing of detection equipment for high energy physics experiments. The basic system we have developed is based on an LSI-11 microcomputer with 24K RAM, 4K ROM, 2 serial interfaces (one to the console terminal, the other to the large SLAC IBM computer complex (the 'TRIPLEX')), a programmable clock, and a CAMAC crate controller. Data logging support is provided for magnetic tape, floppy disk, and an interactive program ACQUIRE which runs on the TRIPLEX under the timesharing system ORVYL. The data is read from various CAMAC modules, collected, buffered, and optionally logged. At a lower priority, the data read is sampled and analyzed in real time on the LSI-11 to produce various histograms and tables. Concurrently a more extensive analysis can be performed by the TRIPLEX program on the data which is logged to it. Interactive facilities provided by the microcomputer operating system enable the user to change CAMAC module addresses and the function codes used with them, specify various data cuts and transformations that are to be performed on the sample data, and specify new histogram limits and titles. Results of the real-time analysis, by both the microcomputer and the TRIPLEX program (if it is attached), may be displayed in graphical or tabular form on the console terminal.

The basic system hardware cost (exclusive of the magnetic tape drive and floppy disk drive) is around $7000. The software is written in a modular fashion so that the user can supply his own data reading and analysis routines. This system has been in use for two years by various groups on several LSI-11s at SLAC.

INTRODUCTION

ATROPOS is an LSI-11 microcomputer[1] based software system for data acquisition and analysis which is now in use at SLAC. The design aims have been to provide a system that:

* is CAMAC compatible
* is portable
* is simple to use
* is modular in software and hardware so it can be easily tailored to individual needs
* can make use of the various facilities[2] of the large SLAC IBM computer complex (the 'TRIPLEX')†† without modification to the TRIPLEX hardware or system software; in particular:
  + can be programmed by making use of the sophisticated software development facilities available on the TRIPLEX for source code preparation and storage, object code production and linking, and down line loading
  + can be used interactively with the TRIPLEX to provide a more sophisticated online analysis than the LSI-11 can provide on its own
* can be used effectively in stand alone mode (i.e., independent of the TRIPLEX and any logging facilities)
* is low in cost.

The basic pattern around which the system must work is that when an 'event' happens, the data resulting from that event must be read in, analyzed locally, and sometimes logged to some form of mass storage. A grouping of events in time (from time A to time B) is called a 'run'. When we 'BEGIN' a run, various initializations must occur. The user must be able to 'PAUSE' the run, possibly to examine the data already taken, to change or fix the hardware, or maybe to change slightly the analysis which is happening on an event by event basis. Then the user will want to 'RESUME' the run. At some point the user will want to 'END' the run, and henceforth the data that was taken will be referred to as the data from RUN N where N is some number. During a run, the user must be able to examine in graphical and

† Work supported by the Department of Energy under contract no. EY-76-C-03-0515.

†† The SLAC computer complex is composed of two IBM 370/168's which run OS/VS2 Release 1.6, and one IBM 360/91 which runs OS/MVT Release 21.8, all under the control of ASP version 3.2.

tabular form the results obtained so far. He may selectively restart some of the analysis (clear histograms) or modify the analysis (redefine or define some data transformations and/or histograms where the results are binned, or even reconfigure the CAMAC devices, i.e., change their addresses and/or function codes).

The following sections describe the system we have developed.

HARDWARE CONFIGURATION

The basic system (Fig. 1) consists of an LSI-11 with EIS/FIS,[1] 24K 16 bit words of RAM, 4K 16 bit words of Intel 2708 EPROMs, 2 serial interfaces (DLV11),[1] a programmable line time clock (KW11L),[3] a CAMAC crate, a dedicated type U CAMAC crate controller (Schlumberger JLSI-10), and an alpha-numeric video terminal (usually an Ann Arbor 4080D).

The 4K of EPROM contains a kernel of routines which support the link to the TRIPLEX, and provide some basic I/O and conversion facilities for use by the stand alone systems which are down line loaded into the RAM. A further description of the EPROM programs can be found in Reference 4.

One of the serial interfaces is for the connection to the TRIPLEX. Communication to the TRIPLEX is via an EIA RS232C asynchronous serial line, utilizing a telephone and modem or a hardwired line. The second serial interface is for the console terminal.

More elaborate ATROPOS systems (Fig. 2) have included:

* a Tektronix 4013 as the console terminal; it is used concurrently as the graphics display device
* a third serial interface that goes to an Ann Arbor terminal (without keyboard) which is used to provide a constantly refreshed tabular display of the event data, current run statistics and 24 bit CAMAC scaler readings
* an 800 bpi 9 track tape drive for local logging of data
* a floppy disk for backup reloading of the program and local logging of data.†††

SOFTWARE DEVELOPMENT AND MAINTENANCE

The programs are written and maintained by using the software development facilities available on the TRIPLEX. The facilities include a PL-11 cross compiler,[5] a MACRO-11 cross assembler,[6] a cross linker,[6] the WYLBUR text editing system,[7] IBM OS data sets, and all the peripherals of a large computer center. A further description of how we utilize these facilities for our software development can be found in Reference 4.

The programs are edited, compiled and/or assembled and linked together on the TRIPLEX to form an absolute load module which is then down line loaded into the LSI-11. This allows us to have complete access to our software development facilities from any site where we have access to a hardwired line to the TRIPLEX or a telephone and modem.

OPERATOR INTERACTION FACILITIES

Operator interaction is provided via the console keyboard and optionally via a specially designed experiment control panel (ECP) (see Fig. 2) which

††† The ROM kernel has a facility for dumping the RAM contents onto floppy disk and restoring it from floppy. It takes about 3 minutes to down line load ATROPOS (16K words) into the LSI-11 over a 9600 baud TRIPLEX line and about 90 seconds to reload from the floppy disk.

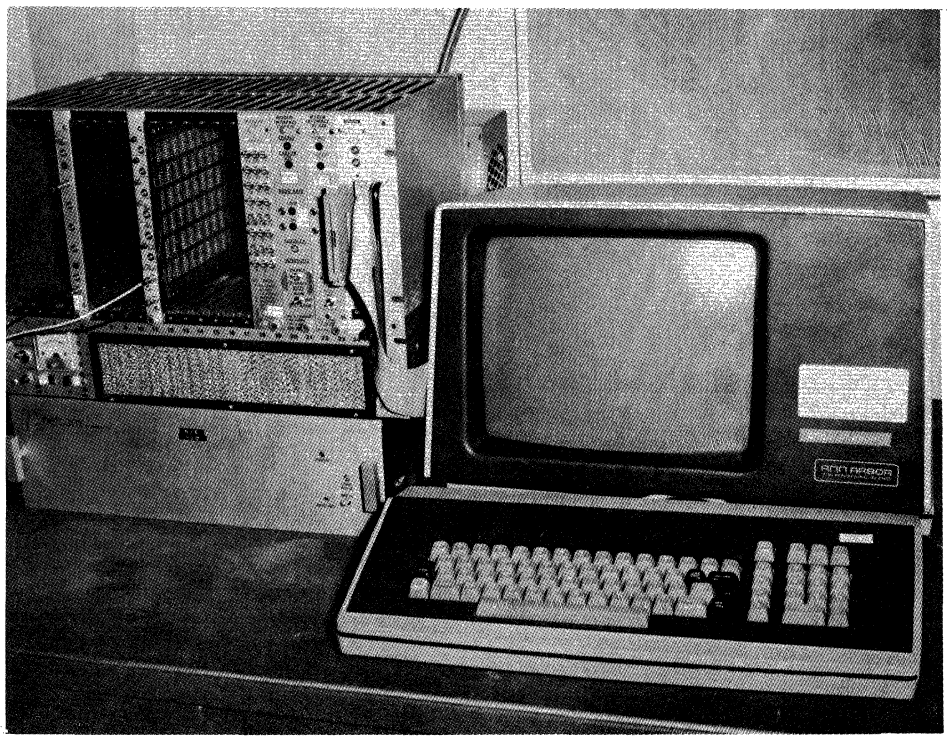


Fig. 1. The basic system is composed of an LSI-11, a CAMAC crate, and an Ann Arbor terminal.

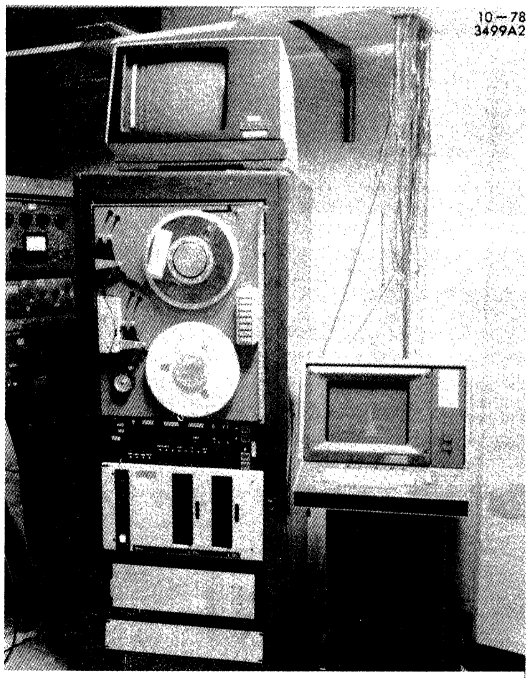


Fig. 2. One of the more elaborate ATROPOS systems consists of (from top to bottom): an Ann Arbor terminal (without keyboard) as an auxiliary display, a magnetic tape drive, an ECP, a dual floppy disk drive, the LSI-11 itself, and on the right, the Tektronix 4013 terminal which is used as the LSI-11 console and the graphics display device.

interfaces to the system via a single width CAMAC module.

The console keyboard is interrupt driven. The user enters, through the keyboard, a command text string and terminates it with a carriage return. The input text string is then matched against a set of command strings and the appropriate subroutine is called to execute that command. Some commands will prompt further for a text string, octal, decimal integer or floating point number accordingly.

The ECP has two 16 bits octal thumbwheels (referred to as ETW0 and ETW1), two 4 digit decimal thumbwheels (ETW2 and ETW3), and one hexadecimal thumbwheel (ETW4). It also has 16 sense lines (toggle switches), 16 push puttons, 4 digits of 7 segment LEDs, and 16 LED status lights. When one of the buttons is pushed, a CAMAC LAM interrupt occurs. The routine which responds to this CAMAC interrupt reads the push buttons and calls the appropriate routine to handle the push button command. Every 1/30th of a second, the ECP thumbwheels and toggle switches are read, and their values are saved in the appropriate CONSTANTs (ETW0, ETW1, ETW2, ETW3, ETW4, TOGL). The CONSTANTs are discussed in the section entitled User Accessible Variables. The 7 segment LEDs are used together with one of the octal thumbwheels to provide monitoring of memory locations. The status lights are used to indicate the status of the program and the data taking.

For an ATROPOS system which does not have an ECP, commands are added to the keyboard command list which enable the user to set the toggle switch bits in TOGL, thumbwheel CONSTANTs (ETW0, ETW1, ETW2, ETW3, ETW4), and issue, through the keyboard, commands that would have been provided by the ECP push buttons.

A list of the commands available follows. Those with a double asterisk are also provided by the ECP if one is available. The command HELP when entered via the keyboard will print a list of available commands together with a short description of each.

**	BEGIN	- Begin a run.
**	BKCLR	- Clear the background queue.
**	CONLIS	- List the CONSTANTs.
**	CONSET	- Set a CONSTANT.
*	DETACH	- Terminate the logging.
**	END	- End a run.
*	EXPDEF	- Define an expression.
*	EXPDEL	- Delete an expression definition.
*	EXPEXEC	- Execute the defined expressions. This is provided for system testing purposes.
*	EXPLIST	- List the expressions which are defined.
*	EXPSHOW	- List the values the expressions had after their last evaluation.
*	HALT	- Execute a HALT instruction and put the LSI-11 into ODT mode.
**	HCLR	- Clear a histogram.
*	HDEF	- Define a histogram.
*	HDEL	- Delete a histogram.
*	HELP	- Print a list of available commands.
**	HGET	- List the histogram definitions.
**	HLOG	- Log the contents of a specified histogram.
**	HOUT	- Display a histogram.
**	HSUM	- Display the statistics for a specified histogram.
*	JOBTIME	- List the time and date the load module was created.
*	DDT	- This command provides limited absolute address memory access and modification. This is provided for system testing purposes.
**	PAUSE	- Pause a run.
*	RECOVER	- Recover from a TRIPLEX crash.
*	RESTART	- Restart the logging.
**	RESUME	- Resume a run.
*	TOGL	- This allows the user to set a sense-line. This command is provided only on systems which do not have an ECP.
*	TRACEOFF	- Disable the trace trap. This is provided for system development.
*	TRACEON	- Set a trace trap. This is provided for system development.
*	WYLOFF	- Disable all TRIPLEX communication.
*	WYLON	- Reenable TRIPLEX communication.
*	CTRL U	- This is equivalent to pushing the UPDATE button on the ECP. It updates the display according to what is contained in ETW0.

The commands which are provided for system testing purposes are usually included in the final production system, since they may be useful in checking out the CAMAC hardware and other peripherals.

USER ACCESSIBLE VARIABLES

In an ATROPOS system there are several groups of variables, commonly referred to as the CONSTANTs, which the user has access to. Each group of CONSTANTs is identified by an ID in the range 100 thru 177 (octal). The CONSTANTs include:

* various run statistics such as the run number (RUN#), the number of events read so far (READ), the number of events logged (LOGD), the number of events not logged (LOST), and the number of events analyzed locally (SAMP)
* various parameters associated with the histogram display format (TOGL,ETW0,ETW1,ETW2,ETW3,ETW4)
* event data pedestals
* event data CAMAC addresses
* event data CAMAC read functions
* the event data
* event data histogram ID's

Commands are provided so that the user can easily look at, and, in some cases, change the values of the CONSTANTs. Each CONSTANT has associated with it an up to four character name, or an up to four character name and an index. A facility is provided whereby when the user tries to set a CONSTANT, a subroutine can be called to check the value it is to be set to. That subroutine then can either issue an error message to indicate a bad value (and possibly a reminder of the purpose of the CONSTANT) and set a flag (so the value will be reprompted for), or set the value to some reasonable value.

CONSTANT accessing commands are:

* CONSET - examine a CONSTANT and set its value
  For example:‡

```
. CONSET
CONST NAME= AA1   (AA1 is a CAMAC address variable.
                   164502 is its octal Q-BUS
                   address, and 1  10  1 is the
                   decoded crate, station, and
                   subaddress.)
(ABS.ADDR.,C,N,A)=  164502   1   10   1
CRATE=1
STATION=<cr>      (A null response does not change
                   the old value.)
SUBADDRESS=2
CONST NAME= AP1
0 : 75            (0 is current contents, replace
                   it with 75)
CONST NAME= <cr>  (The <cr> terminates the CONSET
                   dialogue.)
.
```

* CONLIS - lists a group of CONSTANTs in tabular form
  For example:

```
. CONLIS
CONSTANT GROUP? ? (A '?' gives a list of the
                   CONSTANT group titles.)
101-STATISTICS
102-ECP OPTIONS
103-ADC DATA PEDESTALS
104-TDC DATA PEDESTALS
105-ADC DATA CAMAC ADDRESSES
106-TDC DATA CAMAC ADDRESSES
107-ADC DATA READ FUNCTIONS
110-TDC DATA READ FUNCTIONS
111-ADC HISTOGRAM ID
112-TDC HISTOGRAM ID
113-DRIFT CHAMBER READOUT PARMS
114-SCALER CAMAC ADDRESSES
115-ADC DATA
116-TDC DATA
117-DRIFT CHAMBER TIME, ADR
120-SCALER DATA
121-DRIFT CHAMBER WIRE PEDESTALS
122-DRIFT CHAMBER WIRE GAINS
CONSTANT GROUP? 105
(These are the CAMAC address variables.  The
numbers in parentheses are decimal and are the
crate, station, and subaddress values which have
been decoded from the octal QBUS address which is
listed just after them.)
AA0 : (1,10,0) : 164500; AA1 : (1,10,1) : 164502;
AA2 : (1,10,2) : 164504; AA3 : (1,10,3) : 164506;
AA4 : (1,10,4) : 164510; AA5 : (1,10,5) : 164512;
AA6 : (1,10,6) : 164514; AA7 : (1,10,7) : 164516;
AA8 : (1,10,8) : 164520; AA9 : (1,10,9) : 164522;
AA10 : (1,10,10) : 164524; AA11 : (1,10,11) : 164526;
CONSTANT GROUP? 115
AD0 : 115; AD1 :   0; AD2  : 129; AD3  : 145;
AD4 : 105; AD5 :   9; AD6  :   9; AD7  : 125;
AD8 : 115; AD9 : 110; AD10 :  29; AD11 : 105;
CONSTANT GROUP? <cr>
(The <cr> ends the CONLIS dialogue.)
.
```

‡ In the examples user responses are underlined, and all user responses are terminated with a carriage return. The symbol <cr> refers to a carriage return typed before any other input on that line (i.e., a null response). Lower case text is a comment on the example which would not appear in a real session.

DATA ACQUISITION AND LOGGING

When an ATROPOS system is created, the appropriate routine must be supplied to respond to the event interrupt. The CONSTANTs contain the CAMAC addresses of the modules from which the data is read. These are set by default at system initialization time, but may be changed by the user interactively at any time. Basically all the routine does is read the data, call a system routine to log it, and pass the data onto the sampling routine which is also user supplied and invoked at a lower level. In order to change the 'device' to which the data is logged (the TRIPLEX, magnetic tape, or floppy disk), one has to specify the appropriate object module library and relink the program. No actual coding changes are necessary. Due to the memory required, one cannot create a system which contains simultaneously all three logging facilities. All logging is interrupt driven and there is provision for both synchronous and asynchronous operation. All logical records passed to the logging routine are buffered. In the case of the tape and floppy devices multiple buffers are used. The number (and length in the case of the tape device) is determined dynamically by the event data rates and the available memory. Each physical record includes information to identify the computer system which is generating it and the current run number. Data descriptors are included with the tape and floppy device data to simplify conversion of data types (e.g., floating point, integer, ASCII character strings) from LSI-11 format to IBM 370 format. Event data records also include the event number to provide a check on the data. In the case of the floppy disk logging, each run is written as a separate file in RT-11 format.[8]

ACQUIRE - THE TRIPLEX LOGGING TASK

For the logging to the TRIPLEX, ATROPOS communicates with a job ACQUIRE,[9] which is running under the interactive time sharing system ORVYL.[10] The TRIPLEX terminal communication protocol is basically half duplex with a DC1 character being sent by the TRIPLEX to turn the line over to the terminal. Characters sent by the 'terminal' when it does not own the line are ignored by the TRIPLEX apart from a BREAK which interrupts the TRIPLEX and, after it has been processed, causes the TRIPLEX to turn over the line. Characters may be sent by the TRIPLEX at anytime. When ACQUIRE is attached, it intercepts any messages from ATROPOS to the TRIPLEX, and processes them. There are three types of messages sent by ATROPOS to ACQUIRE: data messages, ACQUIRE commands, and regular WYLBUR commands. The protocol for the ATROPOS/ACQUIRE communication enables ACQUIRE to distinguish between the three kinds.

Data messages start with a DC3 character. They also have a checksum for the data record on the end of them. The communication line to the TRIPLEX only supports 7 bit character transmissions and some of those characters are used as control characters by the communications controller at the TRIPLEX. Therefore each 8 bit byte of binary data is encoded into two 7 bit ASCII hex characters before transmission. When ACQUIRE receives a message with a DC3 on the beginning of it, it computes a checksum, decodes the binary data, saves the data record in the data file (if the checksum is good), and sends a reply back to ATROPOS as to the status of that record. If the record had a checksum error, ACQUIRE notifies ATROPOS and the message is retransmitted. The transmission rate that can be reasonably sustained over a 2400 baud communication line is about sixty 8 bit bytes/second of logical data.

ACQUIRE commands are commands typed by the user that (in most cases) start with an equals sign ('='). These commands allow the user to control the analysis which ACQUIRE is performing on the data which is logged to it. For example, the user may define or redefine the ACQUIRE histograms, clear them, display them (on the LSI-11 terminal or any TRIPLEX graphic device (e.g., the Versatec plotter)), or manipulate the format in which the ACQUIRE histograms are displayed.

Any command not preceeded by an equals sign or a DC3 character is checked against a list of ACQUIRE commands, and if it is not there then it is sent by ACQUIRE onto WYLBUR for processing. In this fashion, the terminal can be used to communicate with WYLBUR and hence be used for program development etc. even while taking data.

Since the ACQUIRE task is written in IBM FORTRAN IV it may be modified by the experimentor. Further it has access to almost all the facilities of the TRIPLEX. For example, it uses the facilities of DPAK and HPAK[11] and Unified Graphics[12] for displays and histogramming, disk mass storage facilities (IBM OS data sets), and the output facilities (lineprinter, Versatec plotter, and other peripherals) available on the TRIPLEX.

HISTOGRAMMING FACILITIES IN ATROPOS

Histograms provide a convenient way of analyzing and displaying the data which is being accumulated. In order to provide as much flexibility as possible, our facilities allow the user to define and/or redefine the histograms both at compile and execution time. Up to 63 histograms (ID's range from 1 to 77 octal) can be defined at any given time. For each the user must specify the ID, the number of bins, the low bin, the bin width, and a title which is displayed when the histogram is displayed. The output facilities are modularized so that one can easily change display devices without a lot of recoding. Typically we use a Tektronix 4013 terminal for a console terminal and display the histograms on that.

Histogram commands available are:

* HDEF which enables the user to define a histogram. He supplies an ID, number of bins, low bin value, bin width, and histogram title.
  For example:

```
·HDEF
ID? 1
# of BINS? 100
LOW? 0
WIDTH? 5
TITLE? TOTAL NUMBER OF WIRES FIRED EACH EVENT
·                    (Up to 80 characters)
```

* HCLR which prompts for the ID of the histogram to be cleared. A response of 0 clears all the histograms. The push button HCLR command uses the contents of ETW0 as the ID.
* HDEL which prompts the user for the ID of the histogram to be deleted. A response of 0 deletes all histograms.
* HOUT enables the user to display the contents of a histogram in a graphical (Fig. 3) or tabular form.
* HSUM displays the statistics of a specified histogram.
  For example:

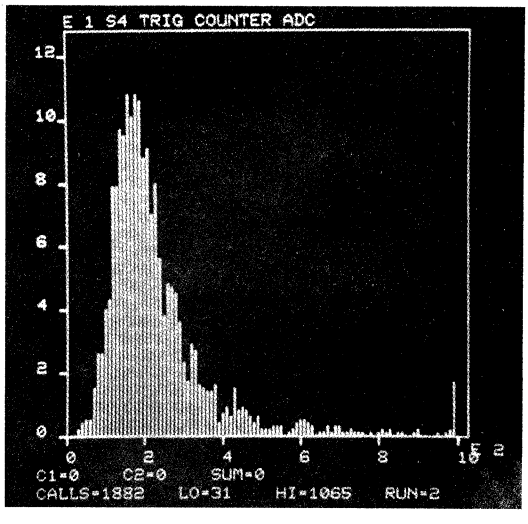


Fig. 3. Example of the graphical display of an ATROPOS histogram.

```
·HSUM
ID=30
HSUM FOR:  PESTOV 1 TDC
  ID : 24   LBIN : 250   HBIN : 448   CALL : 6280
UFLO : 28   OFLO : 6125
Y SUM= 6280.000   XMEAN= 444.5710   SIGMA= 24.23909
```

* HLOG provides the user with a way of saving (on the logging device) the histogram contents for future reference.
* HGET allows the user to examine all or one of the histogram specifications. For example:

```
· HGET
ID? ?   (Response of '?' or 0 results in the
         listing of all defined histograms)
ID=10,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=S1 TRIG COUNTER ADC
ID=11,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=S2 TRIG COUNTER ADC
ID=12,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=S3 TRIG COUNTER ADC
ID=13,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=S4 TRIG COUNTER ADC
ID=14,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=PESTOV 1 ADC
ID=15,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=PESTOV 2 ADC
ID=16,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=PESTOV 3 ADC
ID=17,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=PESTOV 4 ADC
ID=20,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=PESTOV 5 ADC
ID=21,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=PESTOV 6 ADC
ID=22,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=PESTOV 7 ADC
ID=23,TYPE=2@,#BINS=100,XLOW=0,XWID=10,
      TITLE=PESTOV 8 ADC
ID=30,TYPE=2@,#BINS=100,XLOW=250,XWID=2,
      TITLE=PESTOV 1 TDC
ID=31,TYPE=2@,#BINS=100,XLOW=250,XWID=2,
      TITLE=PESTOV 2 TDC
ID=32,TYPE=2@,#BINS=100,XLOW=250,XWID=2,
      TITLE=PESTOV 3 TDC
ID=33,TYPE=2@,#BINS=100,XLOW=250,XWID=2,
      TITLE=PESTOV 4 TDC
ID=34,TYPE=2@,#BINS=100,XLOW=250,XWID=2,
      TITLE=PESTOV 5 TDC
ID=35,TYPE=2@,#BINS=100,XLOW=250,XWID=2,
      TITLE=PESTOV 6 TDC
ID=36,TYPE=2@,#BINS=100,XLOW=250,XWID=2,
      TITLE=PESTOV 7 TDC
ID=37,TYPE=2@,#BINS=100,XLOW=250,XWID=2,
      TITLE=PESTOV 8 TDC
.
```

EXPRESSION FACILITIES

Very often the user would like to look at various transformations of the data the user is taking, and the user wants to be able to change these transformations easily. The simple expression definition and evaluation facilities provided by ATROPOS allow him to do this. There are basically five parts to the expression facilities.

* The expression definition facility is entered by the command EXPDEF. The command EXPDEF prompts the user for an INDEX, an expression, any conditions the user wants to attach to the evaluation of that expression, and an ID of a histogram where the result is to be histogrammed. If no histogram of the result is desired, the user responds 0 to the ID prompt. The expressions are evaluated in strictly left to right order. There is no heirarchy of operators (they all have equal precedence) and parentheses cannot be used to group subexpressions. The result of any expression can be used in a later expression in which case the previous expression is referenced by preceding its index with '%:'. If, at evaluation time, the previous expression was not evaluated because one of the attached conditions was not satisfied, then a zero is used for its value. Elements of the indexed CONSTANTs are referred to by giving the name followed by a colon followed by the index. Every operator is denoted by a single character. The operators allowed in the arithmetic expressions are +,*,-,/, > (arithmetic shift right), < (arithmetic shift left), & (logical and), | (logical or), _ (an underscore for minimum function), and a ∧ (a carat for maximum function). In the conditionals, > (greater than), < (less than), = (equal to), and # (not equal to) are permitted. Note that the conditional A>B>C is equivalent to A>B and A>C, and not A>B and B>C. An operand may be a named CONSTANT or a decimal, octal or hexadecimal integer. Each expression is checked for syntax when it is entered by the user. If at evaluation time an error (overflow or undefined CONSTANT) is detected, the entire expression string is printed on the console terminal with a pointer to the operator where the error occurred. Note that 32 bit intermediate results are propagated if the next operation is an addition, subtraction, or division. For all other operations, if the intermediate result is greater than 16 bits, then an overflow is presumed to have occurred and the evaluation of that expression is aborted with a warning. Although the expression facility is very simple, it appears to be adequate for our purposes and requires little memory for expression storage or for parsing and evaluation. As an example of the expression definition use: suppose we have 3 ADCs (analog to digital converter), each with a pedestal, and 3 associated TDCs (time to digital converter); we want to histogram the residual of each ADC reading and its pedestal, and the TDC/100 associated with the largest residual.

```
·EXPDEF
INDEX=1            (index by which the expression
                    is defined)
EXP=ADC1-APED:1    (the expression)
COND=<cr>          (no conditions attached)
ID=1               (histogram result in histogram
                    with ID=1)
INDEX=2
EXP=ADC2-APED:2
COND=<cr>
ID=2
INDEX=3
EXP=ADC3-APED:3
COND=<cr>
ID=3
INDEX=4
EXP=TD1
COND=%:1 > %:2 > %:3
COND=<cr>
ID=0
INDEX=5
EXP=TD2
COND=%:2 > %:3 > %:1
```

```
COND=<cr>
ID=0
INDEX=6
EXP=TD3
COND=%:3 > %:1
COND=%:3 > %:2
COND=<cr>
ID=0
INDEX=7
EXP=%:4 + %:5 + %:6/100
COND=<cr>
ID=4
INDEX=<cr>
.
```

* EXPDEL prompts a user for the index of the expression which is to be deleted, then deletes it.
* EXPLIST lists all of the defined expressions, their associated conditions, and histogram ID. For example:

```
. EXPLIST
1 : ID=000001 : ADC1-APED:1
2 : ID=000002 : ADC2-APED:2
3 : ID=000003 : ADC3-APED:3
4 : ID=000000 : %:1>%:2>%:3 : TD1
5 : ID=000000 : %:2>%:3>%:1 : TD2
6 : ID=000000 : %:3>%:1 AND %:3>%:2 : TD3
7 : ID=000004 : %:4+%:5+%:6/100
.
```

* To perform the evaluations at the appropriate time, the user puts a call to EXPXEQ in his code. This will usually be in the sampling routine. EXPXEQ will evaluate all the expressions, and histogram the results in the appropriate histogram for those which were successfully evaluated. The command EXPEVAL executes an EXPXEQ call. This is usually used for testing purposes.
* EXPSHOW lists the results from the last EXPXEQ call, along with a flag which indicates whether a given expression was successfully evaluated.

TAILORING AN ATROPOS SYSTEM

Over the past two years we have tailored and used three substantially different ATROPOS systems.[‡‡] In addition, at least two other ATROPOS systems have been tailored by other groups at SLAC. In general we have found the software structure makes it simple to tailor a system to meet a new equipment configuration. In fact we have used the ATROPOS structure as a base for two other unrelated monitoring (CLOTHO) and control (LACHESIS) systems we needed.[‡‡‡]

The following subsections describe the changes needed and the routines that must be supplied in the tailoring of a system.

* The CONSTANTs

The CONSTANTs (their names, memory allocation, check routines and group names) must be set up according to what the system being tailored requires. Although this must be done very carefully, it is a fairly simple process.

* UINIT

UINIT is the basic system initialization routine. It should load the CONSTANTs and initialize them, set up the interrupt vectors for the clock interrupt, the CAMAC interrupts, initialize the keyboard processing, initialize any auxiliary devices (such as tape drives, floppy disk, auxiliary displays), and define the default histograms.

* UMAIN

The group of routines we call UMAIN are the run control routines. UBEGIN, UEND, URESUME, UPAUSE are called when the user does a BEGIN run, END run, RESUME run, or PAUSE run, respectively. In these routines the programmer must enable/disable the event data interrupts, clear the histograms (usually at BEGIN run), and take any other appropriate actions.

* EVENTLAM

This is the routine which handles the event interrupt. It reads the data and does any logging that may be necessary, and then passes the data onto the sampling routine (also user supplied and user invoked).

* UCLOCK

The user must supply this routine to respond to the clock interrupt if the target machine has a clock. For an ATROPOS system which does not have a clock, this routine can be omitted.

* BKGRND

Although ATROPOS is completely interrupt driven, we have found it advantageous to do some things in background as CPU time allows. For example, updating the console display and driving the auxiliary display are done from the background. In the case of our auxiliary display, every 1/30th of a second, the clock routine posts a command to the background queue to update the display if a command is not already present in the background queue.

Currently the user supplied routines must be written in PL-11 or MACRO-11. Soon we hope to have PASCAL[16] and FORTRAN[17] programming capabilities added to our LSI-11 software development facilities on the TRIPLEX.

CONCLUSIONS

The ATROPOS system described herein is quite useful for our needs, although it does have its limitations. Since we do not run under a disk based operating system such as RT-11, we do not have all the flexibility that we could have with, for example, overlays. With the memory constraints relieved by

‡‡ The first system we developed was used in the SLAC C-beam in December 1976 to test a new shower counter design.[14] In summer 1977 we tailored a system which we used to develop a hardware method of generating random bits.[15] That one entailed the logging of several million bits of data to the TRIPLEX. Currently we are using ATROPOS to develop detectors for a future experiment.

‡‡‡ CLOTHO, LACHESIS, and ATROPOS are the names of the three fates.[13] CLOTHO is a system which monitors the polarized electron source located in the SLAC injector. CLOTHO was the youngest of the three fates. She spun the thread of life. LACHESIS is the system which provides closed loop feedback position and energy steering of the SLAC beam in beam line 'A' of the beam switchyard. LACHESIS was the fate who determined the length of the thread of life. ATROPOS (the system described in this paper) provides analysis facilities for the result of the beam hitting a target. ATROPOS was the fate who cut off the thread of life.

overlaying, we could have a much more elaborate system. Since we do not use a disk and RT-11, we are dependent on the TRIPLEX for all our coding changes and loading. However, in our experience, this has not been a problem.

In general, we feel the advantages of our system outweigh its disadvantages. It is very low in cost (no software licences and a minimum of peripherals), portable, and very reliable. We do not have a lot of mechanical peripherals upon which we are absolutely dependent. If, for example, we are logging to tape and the drive fails, we can switch easily to the TRIPLEX logging, although we will log at a much slower rate to the TRIPLEX. For data storage and output purposes, we have at our command the facilities of the TRIPLEX, which provides things we could in no way afford for each individual ATROPOS system.

ACKNOWLEDGEMENTS

We wish to acknowledge the help received from the following people: Sylvia Sund for her work on ACQUIRE, the interactive TRIPLEX program; William Bryg for his work on the logging; Joe Zingheim for building the LSI-11s and Experiment Control Panels, and helpful discussions; and the many users of the system for their helpful suggestions and interesting challenges.

REFERENCES

1. The Microcomputer Handbook, Digital Equipment Corporation, 1977.

2. SLAC Computer Services User Note No. 99, The TRIPLEX Users Guide, June 1978.

3. MDB MLSI-SMU System Monitoring Unit, MDB Systems Inc., 1995 N. Batavia Street, Orange, CA 92665.

4. R.L.A. Cottrell and C.A. Logg, An IBM 370/360 Software Package for Developing Stand Alone LSI-11 Systems, Proceedings of the Digital Equipment Computer Users Society, Vol. 4, No. 4, pp. 985/991, April 1978.

5. Robert Russell, PL-11: A Programming Language for the DEC PDP-11 Computer, Edited by T.C. Streater, CERN 74-24 (1974).

6. S. Steppel and H.E. Syrett, XASM11/XLINK11, A PDP-11 Cross Assembler/Linker User's Manual, CGTM. No. 160 (1974), SLAC, Stanford, CA.

7. WYLBUR/370, The Stanford Timesharing System Reference Manual, Third Edition, November 1975.

8. RT-11 Software Support Manual, DEC, Maynard Mass (1976), Order #DEC-11-ORPGA-B-D, DNI.

9. R.L.A. Cottrell and S.J. Sund, ACQUIRE, Group A Program Document No. 34, January 1977, Group A, SLAC, Stanford, CA.

10. ORVYL/370, The Stanford Timesharing System, Functional Description, October 1975.

11. C.A. Logg, A.M. Boyarski, A.J. Cook, and R.L.A. Cottrell, DPAK and HPAK - A Versatile Display and Histogramming Package, SLAC Report No. 196, June 1976.

12. Robert C. Beach, The SLAC Unified Graphics System, CGTM No. 170, January 1976.

13. J.E. Zimmerman, Dictionary of Classical Mythology, Bantam Books Inc., 1971.

14. W.B. Atwood, C.Y. Prescott, and L.R. Rochester, First Test of a New Shower Detector, SLAC TN 76 7, (1976).

15. R.L.A. Cottrell and S. Sund, RANDOM: Random Bits for Source Polarization Sign Selection, Group A Programming Document No. 48, December 1977.

16. Bruce L. Hitson, PASCAL/PCODE Cross Compiler for LSI-11, to be presented at the Fall DECUS Symposium, 1978.

17. B.M. Bricaud and R.L.A. Cottrell, Integration of RT-11 FORTRAN into the IBM 360/370 Software Development Package for LSI-11s, Group A Programming Document No. 50, March 1978, Group A, SLAC, Stanford University, Stanford, CA.

MUMPS/IDS OPTOMETRIC OUT-PATIENT TURNKEY INSTALLATION - A CASE HISTORY

RALPH DIPPNER
SYSTEMS ANALYST
STATE UNIVERSITY OF NEW YORK
STATE COLLEGE OF OPTOMETRY
100 East 24th Street
New York, New York 10010

We are installing MUMPS and IDS software for a large optometric clinic out-patient accounting system. This unit is to be 'turnkey' and should be cut over by November 1, 1978. I am a systems analyst with over 10 years of experience in commercial and real-time/scientific computer usage, and am monitoring the effort as I intend to build upon it as soon as it appears to be running properly. I am currently the sole programmer/analyst at this site.

This paper covers 1) selection, 2) lease/buy, 3) user interface, 4) run-in and training conditions, 5) semi-final evaluation, 6) future course.

BACKGROUND

The State University of New York, University Optometric Center, located in Manhattan, provides eye care service to a large clinic population and training for optometrists for the State and some surrounding areas. Patient accounting, billing and preparation of a medicaid magnetic tape were performed in a batch mode off-site. Costs were linear with respect to patient load, error rates were high, lags in reporting were two to six weeks, and the system was expensive to modify. We informally evaluated over one dozen alternative processes, and issued a RFP (request for proposal) for turnkey on-line real-time system. Criteria were essentially, discounted cost, time savings, support and responsiveness. Reporting and systems requirements, both mandatory and desirable, were stated. They included as mandatory, as an overview, an ability to note patient and procedure flows, medicaid billing tape (1600 bpi), conversion of existing master tape, on-line edit, suitably fast record access (by name, partial name, or identification number), suitable restriction on access, expandability, flexibility, suitable data base backup, and separate listings of charges, and cost of later charges. Desirable features were (again as an overview); heavy and complete documentation, proven reliability, parallel runs, suitable protocols for non-experienced users, fall back and recovery drills, pricing by treatment, multi-part tear off mailers, all bill generation on-site, batch posting of medicaid returns, and handling of third party guarantors. We expected firm maintenance statements 9 a.m. to 9 p.m. Monday to Friday, both lease and purchase alternatives to be shown, reasonable lease cancellation provisions, specification of minimal system up-time, and pro-rates for non-achievement.

The RFP was sent to over 30 major computer vendors and OEM's. Nine responses were obtained, and a formal question session allowed to answer vendor questions.

EVALUATION

SYSTEM # TYPE	REASON FOR REJECTION
1. on-line off-site Fortran/MOMS/CDC	a) overpriced b) not on site
2. IDS/MUMPS PDP 11/34	
3. MUMPS on Basic DG hardware	a) overpriced b) poor cancellation provision
4. Software unknown WANG WCS/30	a) lack of lease b) lack of demonstratable software c) poor flexability for growth
5. MIIS PDP 11/34	a) heavy lease cancellation penalties b) a bit overpriced
6. COBOL MRDOS/DG	a) overpriced b) lease cancellation penalties heavy
7. BHAS II/BHIPS Burroughs B1700/1800	a) way overpriced
8. RPG II Data share/datepoint	a) overpriced b) no cancellation clause
9. Unknown software PDP 11/34 +ABD's CRT's	a) no cancellation option b) overpriced using their figures

System #4 failed mandatory specifications, due to lack of a lease offer, but all others met them.

Contract approval was obtained from the State, DEC and IDS, implementation plans drawn up, and hardware and software was finally ordered 6/14/78.

CONTRACT CONDISERATIONS:

Many contractors did not ask critical questions at the briefing, although paths were specifically opened for this. Perhaps they knew the answers, or perhaps they had a 'we will overcome' attitude (although no one thought the approach sufficiently innovational to break prices for us), and finally perhaps we will have some grief if responsibilities are not stated explicitly at contract time. Yet we did not have 10 current systems in 10 optometric health centers to evaluate.

All costs that might be startup costs should be paid over the first twelve months or longer.

The hardware manufacturer should warrant equipment and not charge maintenance over the warranty time period.

No one was willing to give penalties for slow performance, and no one offers one year guarantees on hardware and software. Bills for added help if any is needed, after warranty period, must be nailed down.

FUTURE DEVELOPMENTS:

Note most of the vendors offer heavy cancellation penalties. This is due largely to the threat of obsolescense in computer hardware, as they don't know what the market value will be when we might give a unit back to them.

The literature and conversation indicate development is "underway" for microprocessors with MUMPS supporting multi-user data bases, as well as MUMPS under or with UNIX (a cheap powerful operating system developed by Bell Labs), but our savings will be in the CPU-central processing unit (about 10% only) not in the peripherals needed, and successful development appears to be at least two years away. Rarely will the CPU be the bottleneck on processing.

Real savings occur when the contract is over, and we pay only $650/month or thereabouts for maintenance.

The recommendation made herein does not place us in the forefront of activity at all, but it does allow reasonable upgrade without substantial additional costs, while avoiding dead-end or temporary approaches.

We have three people currently at the admitting desk, and an in-house population growth rate estimated at a)low of 10% for theforseeable future and b) 100% within 2 years and 10% thereafter. We will be able to avoid admissions staffing increases only by faster thruput and fewer entry errors. It is estimated that 30% of a hospital cost is "information processing". (1) MUMPS has grown to a nationally recognized and standardized computer health data base language within the last five or so years. It is also acceptable for CAI (computer assisted instruction). A users group has been formed, and additional shared applications at low cost are emerging. It's drawbacks currently are that it is not oriented to heavy computation tasks, and currently uses no operating system other than its own internal one.

Education of health care personnel in information flows and systems has been lagging technical development and recognized cost/effective approaches. We believe we have here the basis for an effective approach.

We will stabilize this system, do a management evaluation, pay for it, then consider relative priority and profit of patient recall, patient scheduling (including O.D.s, students and rooms), extension to clinical findings, Dx and Rx assistence, automated system use training modules, better survey and statistical analysis. As sole on-site analyst with two other computers to work with, the above are also dependent upon relaxed fiscal constraints and/or more training of non-DP management in systems work and programming.

FINAL SYSTEM CONFIGURATION:

PDP 11/34 with 128Kb MOS memory, communications multiplexor with 8 channels now, expandable for $1600 to 16 channels, 5 VT52 CRTS, 1 ADM 3A, 1 Decwriter,1 LA180 high speed character printer, one 1600/800 bpi magnetic tape controller and drive, one RM02 67 megabyte controller and drive, 600' of cabling,IDS software, training and documentation.

The system showed up as a partial shipment with no software, as DEC had not finished testing the version supplied to them by IDS. Generally one should avoid new equipment if close adherance to schedules is required, since support can be poor. The boot shipped with the system here would not support the RM02 disk.

USER INTERFACE, RUN IN AND TRAINING:

Instruction has been OJT almost totally, and little added staff needed, except to bring the converted master file data up to date. Only self-pay patients with open balances were converted. This was less than initially desired, but was workable. The system went down twice with bus errors, but added anti-static carpeting corrected that. The system is started Monday at 8:00 a.m. by non-DPers, and runs until about 8 p.m. Friday, when it is successfully shut down by non-DPers. Management has yet to be fully trained, and documentation is scanty from a systems viewpoint.

SEMI-FINAL EVALUATION

Due to the lag of hardware final acceptability from DEC and incomplete master tape conversion by IDS, on-line operation commenced 11/1/78, leaving only one week for evaluation. Problem areas will be: 1) isolation of hardware vs software contributions upon system malfunction. 2) diagnosis of malfunction via X-11 if the magtape unit malfunctions, as there is no XXDP RM02 monitor, 3) sufficient training for managers as to allow more alternatives and intelligent decision making when the system needs to be modified, 4) final checkout of medicaid tape acceptability, 5) proper training on hardware startup/shutdown alternatives,6) final clerical training and system failure procedures, and 7) adequate maintenance coverage at minimum cost. Finally we may have to replace the label printing LA36 with a better but cheaper EIA tractor feed printer, since the need to leave the last 3 labels on the LA36 tractors disrupts work flow a bit. I should have then largely resolved by late November, except for some final documentation, and training of managers in MUMPS, which IDS apparently is not going to do.

We still expect an increase in collections immediately and better control thereafter. Clerks like the entry and recall speeds, we can add ports and have enough storage for growth as well as expansion to other data flow control problems.

Achnowledgement is gratefully given to Drs. A. N. Haffner, E. Johnston, M. Heiberger, and R. Weber, M. Soroka, R. McQuade, Ms Pat Ruopoli, Mss. D. Anderson, H. Kaslow, M. Flippin (DEC) and the clinic staff, whose cooperation moved this project out of the area between "more than a bit difficult" and "effectively impossible".

REFERENCE

1) Hospital Computer Systems - M.F. Collen - 1974

STANDARDIZATION IN COMPUTER GRAPHICS
An Overview

R. E. Fryer
Naval Weapons Center
China Lake, Ca. 93555

ABSTRACT

Within the last two years, several major steps have been taken toward establishing a standard for graphics support software. The Graphics Standards Planning Committee (GSPC) sponsored by ACM has developed a strawman graphics system design of 'core' features. Included in the core features are the ability to define Picture Segments with various Attributes and composed of graphics Primitives. The graphical object is Transformed by View Specifications before being displayed. Control functions for the graphics software and hardware are also designed into the Core system, as are interfaces to the application program and the environment. A formal committee of ANSI is now determining whether to recommend the development of a standard. The GSPC is continuing to refine the Core design, and coordinating trial implementations.

INTRODUCTION

The need for standards in computer graphics was often expressed in the early 70's. The first collective effort to address this topic began at a relatively small Special Interest Group conference held in April 1974 at the National Bureau of Standards(1). Following the initiatives established in part at this meeting, the GSPC was chartered. After two years activity that included extensive interaction with and contributions by European colleagues, the GSPC was realigned to include the goal of describing a set of core graphics functions that substantially met standards related criteria. The major result of these collective efforts was distributed in July 1977 in a document often called the 'Core Report'(2). This document has formed the design guide for several trial implementations. A detailed description of this background material has been prepared by Puk(3).

REVIEW OF THE CORE SYSTEM

A second part of the Core Report is a survey of several existing graphics support systems. Within this survey, a set of comparison criteria are discussed by which a graphics system may be described. This format ((2), pp. I-3, I-8) is used in broad form in this review.

Overview

The Core System design was the result of a methodological approach developed during the 1974-1976 time period. Program portability was the strongest factor in the design approach. However, absolute portability is not expected (some changes in source code are required between facilities using standard languages due to minor implementation issues - graphics code will be subject to at least these same portability problems). In the development of the Core System, an attempt was made to support portability except for minor syntax and related issues.

The following concepts were considered essential foundations for the standard design ((2), p. II-3):

Separation of input and output functions

Minimum differences between plotter output and interactive display output

Use of two coordinate systems -
world coordinates for picture construction
device coordinates for display data

A display file will contain device display data

Segments within the display file can be independently named and modified as a unit

Viewing transforms are invoked to convert world coordinates into display device coords.

Another philosophical aspect in the Core concept is support of levels of capability. To encourage acceptance of the Core approach by users ranging from plotter output through real time 3-D interactive applications, four major and 3 sub levels are defined:

Basic Output only; for 'non-dynamic', non-interactive applications (plotters, storage terminals, microform, etc.). Segments are used but not stored; full 2-D and 3-D primitive set, all viewing transforms.

Buffered Output only; segments are used and retained, so attributes may be modified.

Interactive Adds Input to the above, so segments are detectable when enabled.

Complete Adds transformations for images to exploit common hardware capabilities; incorporated in three sub-levels:
2D Image Translation transformations
2D Translation, Rotation, and Scaling
2D, 3D Translation, Rotation, and Scaling

Methodology is further discussed in (2) and (4).

Input

The Input system design is based on the following concepts. The user software interface is with logical devices, or functions - not with physical devices. This improves portability since a given installation can bind these logical devices to available physical devices or simulations of them. Five logical input devices and typical physical devices are discussed below. Some logical devices produce data when sampled; others create an event when activated. Events received by the Core are placed in an event queue.

Graphical input is defined for 3 logical devices.

Locator The function of locating a 2 dimensional position (normalized device coordinates are reported) is usually provided by physical tablet or joystick. This device is sampled, and the Core automatically echos a cursor.

Pick The Pick device generates an event. The event report contains the name of the segment picked, and the name of the primitive within the segment picked if it also was named. A typical pick physical device selection is a light pen. This device is also often accomplished with a tablet.

Valuator One dimensional input (establishing a value) is typically realized in hardware with pots and shaft encoders, though an elegent implementation has been done with a light pen. This device is sampled and provides its value in normalized device coordinates.

The other two logical input devices are:

Keyboard This event device provides single characters or strings in the event report. A terminal keyboard or a simulation of one on a tablet are the most common physical devices.

Button This device returns the logical button I/D on the event report. A function keyboard is a typical physical realization, though the function is often simulated with a light pen or tablet (menu selection).

Output

Output may be generated for 2D or 3D graphical objects (2D is a defined subset of 3D). World coordinates are used to define objects, and a right or left handed system may be selected. Coordinates are defined to be Absolute or Relative to a Current Position that always defines the start of the next graphical output.

Primitives - The object may be composed of Lines, line strings (POLYLINE), TEXT and MARKERS. Primitives are not named, but may have a PICK_ID for input discrimination of primitives within a segment.

Primitives are defined in world coordinates, and modify the Current Position (CP). The CP defines the origin for the next primitive.

Primitive Attributes, in addition to the PICK_ID mentioned, include COLOR, LINESTYLE, LINEWIDTH, and INTENSITY. These attributes, once named, must remain fixed until the primitive is deleted. It may then be reborn with a new attribute.

Text - Attributes of the TEXT primitive include FONT, CHARSIZE, CHARSPACE, CHARPLANE, and CHARQUALITY. The space attribute defines the components of the vector pointing between character origins in 3 space. CHARPLANE defines the orientation of characters. CHARQUALITY defines the extent to which the system user wants to use available hardware character generators or levels of complexity of software character generators. Low quality text allows for maximum use of existing character generators; high quality demands complete conformance with all attributes.

Picture Segments - A picture segment contains a collection of primitives and attribute specifications. Segments are initiated by a CREATE_SEGMENT call and terminated with CLOSE_SEGMENT. Segments are typed retained or non-retained. Retained segments may be allowed to have no image transform capability, or any of the sub-level options for image transformations. Non-retained segments provide a framework for the data to be plotted but no storage for the data (no display file is retained). Only retained segments may be renamed or deleted.

Segments may not contain references to other segments - that is, picture subroutining is not provided in the Core System.

Segment Attributes The segment type may not be modified once a segment is created. The Visibility, Detectability (for input), ability to Highlight, and Image Transform parameters may all be dynamically modified during or after creation of the segment.

Viewing Transforms

Two kinds of transformation are provided - viewing and image. The viewing transformation is specified by defining a reference point to view (possibly a point on an object), a normal to the plane of the projection plane (picture resulting from the projection), and the distance from the plane to the reference point. Projection may be parallel or perspective. Rotation may be defined by also specifying a VIEW_UP direction.

Windowing may be rectangular for 2D, a 3D solid frustrum or a 3D parallel solid. Windowing may be turned on or off.

Image Transforms - The image produced by the viewing transformations may, if enabled, be further modified in a level 4 (complete) Core. An analogy may be drawn between 2D Image Transformations (rotation and translation) and rotation/translation of a picture resulting from a camera shot taken from the same viewpoint defined above. This analogy begins to fail when considering scaling and 3D rotations.

Control

The Core System must be Initialized, and the user then defines the level of support required for his application. This may be used to dynamically load the required routines, or perhaps just to diagnose if the local implementation is inadequate.

The user may also INITIALIZE, SELECT, DESELECT, and TERMINATE devices. A series of changes to a picture may be collected under a BATCH_OF_UPDATES set of controls. This improves performance of some graphics systems where update time is relatively long (such as a network system or a storage terminal that must be erased and redrawn at terminal speeds).

Errors are defined to be diagnosed for each function within the Core System. Error reporting will be routed to a logging device by the Core or passed to a user Error Handling function.

The user may INQUIRE and SET values of many internal Core variables such as attributes, the CP, viewing parameters, number of surfaces, input status, etc.

Hooks - Hooks define standard ways to provide the application program with access to information or procedures within the Core. Hooks are required features in Core implementations.

Escape - The ESCAPE allows the application program to access special hardware features. This capability is provided since it will be sometimes required, and this approach provides a standard way to call non-standard functions.

ANSI STUDY GROUP

Within the American National Standards Institute (ANSI), the X-3 Standards Planning And Requirements Committee (SPARC) has formed a Study Group on Computer Graphics Programming Languages. The role of the Study Group is to investigate areas where standards in computer graphics are recommended. A recommendation for a standard must be based on technical feasibility, economic benefit, and practicality (especially implementation time). This group will recommend whether one or more standards should be developed. Though this Study Group won't prepare any recommended standards, it may provide follow-on groups (if any) with extensive guidance based on its studies.

This Study Group is formed of subgroups that are coordinating with other standards bodies (including the International Standards Organization), reviewing Core considerations, defining other graphics aspects appropriate for consideration, and carrying out related study and review tasks(5).

GSPC

The ACM special interest group SIGGRAPH has redirected the GSPC to new goals and activities to be pursued concurrently with the ANSI efforts. The major goals are(6):

> To continue technical studies relating to standards, including refinement/extension of the Core System.
>
> To disseminate and gather information relating to past and new efforts.

These functions are also being met through several subgroups that include clarification, refinements, and extensions to the Core System, surveys of trial implementations of the Core System, and further study of graphics standards methodology.

SUMMARY

The background of the current computer graphics standards effort within ANSI and SIGGRAPH has been reviewed. A quick review has been provided of the structure and capabilities of the Core System, a subroutine oriented design of a graphics support system.

The current goals and activities of the ANSI Study Group and the GSPC are reviewed. References are provided for access to the next level of detail.

REFERENCES

1. Proceedings of the Conference on Device-Independent Computer Graphics, National Bureau of Standards, Gaithersburg, MD, April 1974. Published in Computer Graphics, Vol. 8, 4, Winter 1974.

2. "Status Report of the Graphic Standards Planning Committee," Computer Graphics Vol. 11,3 Fall 1977.

3. Puk, R. F., "The Background of Computer Graphics Standardization," Computer Graphics Vol. 12, 1-2, June 1978.

4. van Dam, A. and Newman, W., "The Development of a Graphics Standard," Computing Surveys 10,4 December 1978.

5. "ANSI X-3 SPARC/CGPL Goals and Activities," Computer Graphics Vol. 12, 1-2, June 1978.

6. "ACM/SIGGRAPH GSPC Goals and Activities," Computer Graphics Vol. 12, 1-2 June 1978.

A GENERALIZED PLOTTING FACILITY*

R. D. Burris and W. H. Gray
Oak Ridge National Laboratory
Oak Ridge, Tennessee

ABSTRACT

A command which causes the translation of any supported graphics file format to a format acceptable to any supported device has been implemented on two linked DECsystem-10s. The processing of the command is divided into parsing and translating phases. In the parsing phase, information is extracted from the command and augmented by default data. The results of this phase are saved on disk and the appropriate translating routine is invoked. Twenty-eight translating programs have been implemented in this system. They support four different graphics file formats, including the DISSPLA and Calcomp formats, and seven different types of plotters, including Tektronix, Calcomp, and Versatec devices. Some of the plotters are devices linked to the DECsystem-10s and some are driven by IBM System/360 computers linked via a communications network to the DECsystem-10s. The user of this facility can use any of the supported packages to create a file of graphics data, preview the file on an on-line scope, and, when satisfied, cause the same data to be plotted on a hard copy device. All of the actions utilize a single simple command format.

INTRODUCTION

1.1 General

In the last few years the advances in computer hardware and software have been impressive. For the most part such advances have been beneficial, but the problem of "future shock" exists in computer science as well as in the rest of our culture. The problem has been perhaps more severe in the specific area of graphics, where graphics support packages and plotting devices are legion.

1.2 Environment

The system we describe was developed for DECsystem-10s, of which our installation has two. They are linked to one another and to several IBM System/360s by communications lines and software. We are concerned in this paper with timesharing users of the DECsystem-10s.

Several types of plotters are available: Tektronix graphics terminals, Versatec printer/plotters, Calcomp pen-and-ink flatbed plotters, and a FR80 COM device with graphics capability. Over the last 15 years many graphics support packages for various plotters have been written or acquired by our installation, so that now there are about 50 graphics support packages available.

1.3 Problem Description

Consider the following set of basic decisions which must be made before a graphics application is designed and developed.

1. The designer must decide upon the plotter to be used in production.
2. The implementer must decide upon the plotter to be used in debugging.
3. The implementer must choose a graphics software package which
   (a) is appropriate to the solution of the problem,
   (b) supports the production plotter, and
   (c) supports the debugging plotter.

Some problems arise in making these decisions.

1. The implementer must understand and support the interface between the graphics package and the plotter.
2. The implementer should understand all packages and interfaces so that accurate decisions are possible.
3. Everyone must consider the interface between the user and the product — if it is difficult to use, no one will use it.

In an environment where so many options are available, these problems are immense.

1.4 Problem Solution

If a system were available which would translate a given file or graphics data to a format consistent with any available plotter, and if such support were available for all supported graphic files, all the problems would be solved.

1. The designer and implementer no longer need to choose the plotter, since all applications are device-independent.
2. The implementer need not consider the chosen plotter while deciding upon the support package — the package most appropriate to the problem may be used.
3. The interface between the support package and the plotter is already done.
4. There is only one user interface.

*Research sponsored by the Office of Fusion Energy (ETM), U.S. Department of Energy, under contract W-7405-eng-26 with the Union Carbide Corporation.

Note that in such a system, the interfaces must be well done since users of all levels of expertise, from rankest greenhorn to seasoned pro, will be using them.

1.5 Purpose of This Paper

The system described in this paper is an implementation of the concept set forth in Section 1.4. At this time four graphics file formats and seven types of plotters are supported, with all 28 possible combinations of those elements implemented. A simple, flexible, powerful, and extremely forgiving interface has been developed. We discuss the design parameters for this system and the structure that evolved for their implementation, then describe the system components, and finally present scenarios of the use of the system by novice and experienced users.

2. SYSTEM DESCRIPTION

2.1 General

We have been describing some of the problems facing users to emphasize the primary design goal of this facility, the provision of a device-independent, user-oriented plotting facility. We wanted to provide a user interface which would not require much learning, would be essentially static over long periods of time, would require a minimum of user input, and would permit corrections without repeating the entire request.

2.2 Design Parameters

The generalized plotting system was designed in accordance with these specifications:

1. The system must be capable of handling several graphics file formats and several plotters.
2. The system must be easily and consistently expandable to new file formats and devices.
3. The user's view of the system should be static except for additions supporting new facilities.
4. A single command should be able to direct the conversion of any supported graphics file to a format compatible with any supported device.
5. The command should have a syntax familiar to the user.
6. The command must be easy to use.
7. The command processor must be exceptionally forgiving of user errors.
8. As much existing software as possible should be used, to minimize development time.
9. The system must be easy to maintain.

2.3 System Structure

The processing of a user command was seen to consist of two phases: the inspection of the command for validity and the translation of the specified data to the required plotter format. The system was therefore divided into two parts: the command scanner (the user interface) and the translators. Since many translation programs already existed, they were to be used to the greatest extent possible. In fact, the only modification to the translators presently implemented has been to cause the translator to look in a disk file for the name of the graphics file.

In general, the command scanner processing consists of the following steps (see Figure 1):

1. Acquire the command to be inspected.
2. Parse the command into its constituent fields.
3. Verify the existence of the required translator.
4. Acquire system default information.
5. Acquire default information for the required translator.
6. Acquire and apply user defaults (SWITCH.INI file).
7. Prompt for any omitted mandatory information.
8. Prompt for any needed corrections.
9. Build a parameter file for the translator.
10. Invoke the translator.

ORNL/DWG/FED 78-867

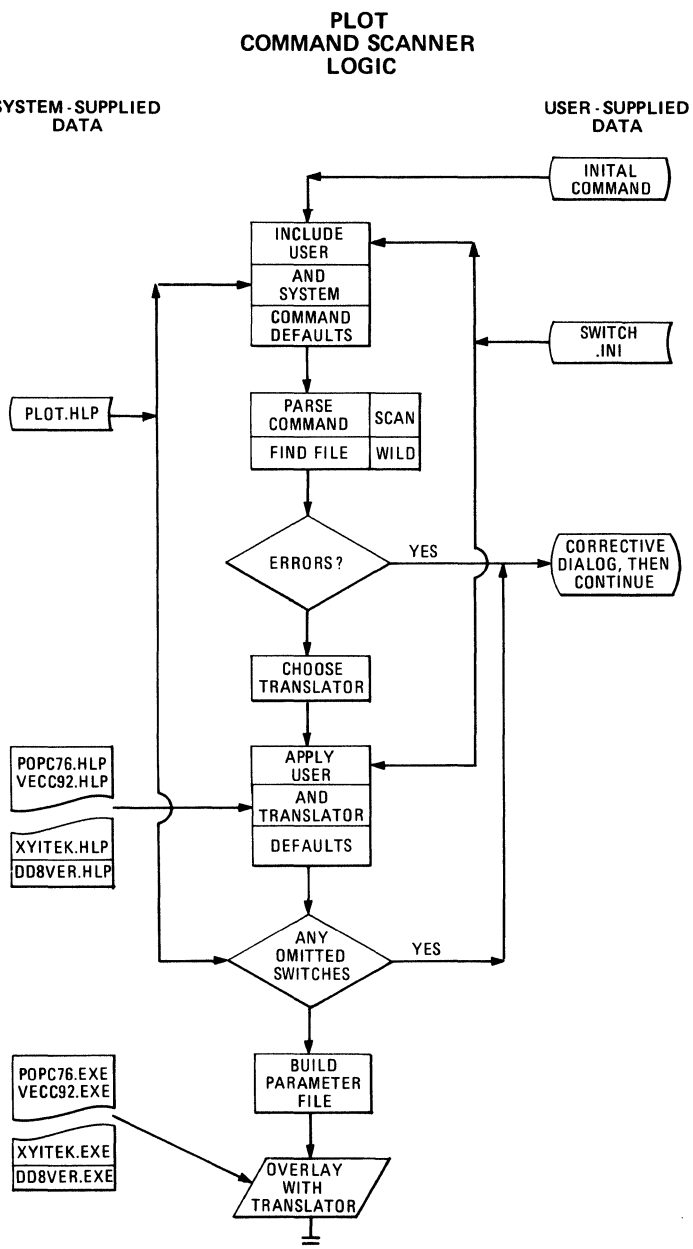


Fig. 1

3. USER INTERFACE

When a system provides great power and flexibility, so many options may be available that the system could be impossible to use. Consequently, a great deal of attention was placed upon the structuring of the system for ease of use. Among the steps taken were the specification of a simple, familiar command syntax; the definition of many defaults; and the corrections of many errors.

3.1 Command Syntax

Since the facility was intended for installation on a DECsystem-10, a syntax appropriate to that computer was desired. The Peripheral Interchange Program (PIP) syntax was chosen as being the most familiar to the user. Specifically, the syntax was

```
.PLOT PLTDEV:=DEV:FILE.EXT/SWITCH,DEV:FILE.EXT/SW
/SW,...
```

where
PLTDEV is the destination plotter,
DEV is the device upon which the source file is found,
FILE is the name of the graphics file,
EXT is the type of data (i.e., Calcomp, Tektronix, vectors, etc.), and
/SWITCH is additional information about the file or the translation to be done.

To maintain maximum compatibility with DEC software, the SCAN and WILD programs were used to process the command string.

3.2 Defaulting

The user of a facility on a computer system rarely knows all of the available options. In a facility as extensive as this one, the number of options is very large, so the user could find effective functioning very difficult unless the system is carefully designed. One way of aiding the user is an extensive set of default values.

Defaults to two types of fields are provided. The first type is called the command field and includes device, file name, and extension. The second type of field is the switch.

Defaults are also provided in a variety of ways. There are defaults provided by the system, by the user in a SWITCH.INI file, and by the user while invoking the command.

3.2.1 Command Defaults — There are two sources of default information for the command field defaults — the system and the user. If the user types

```
.PLOT
```

the system will interpret the command to mean, for instance,

```
.PLOT VER:=SEGMNT.VEC
```

These defaults are a function of the computer upon which the generalized plot facility is installed and are contained in a file known to the command scanner.

It often happens, however, that devices or file formats are used in bursts. For instance, users will perform debugging using Tektronix terminals to take advantage of quick turnaround and would thus like a different default plot device to obviate typing

```
.PLOT TEK:=
```

every time. Accordingly, the switches DEVICE, FILE, and EXTENSION have been implemented for inclusion in the SWITCH.INI file. If the user desiring extensive use of the TEK device includes the line

```
PLOT/DEVICE:TEK
```

in the SWITCH.INI file, the results of

```
.PLOT
```

are now

```
.PLOT TEK:=SEGMNT.VEC
```

Of course, if the user explicitly specifies some field, that specification overrides any default, just as the SWITCH.INI specifications override the system defaults.

3.2.2 Switch Defaults — As in the case of command defaults, there are several sources of default information for switches. Again, there are system-supplied defaults, but now the defaults are specific to each translator, rather than to the system as a whole. These defaults are kept in files known to the command scanner and are applied once the translator has been chosen. In addition to the specification of values for switches, these files contain information about the switches required for the translator and switches meaningful to it.

The user may also specify default values by including information in the file SWITCH.INI. Any switch may be included in the SWITCH.INI file. If the same switch has a system default, the SWITCH.INI value takes precedence.

A third means of specifying defaults is by sticky defaults. That is, in a command which will plot several files, switches common to all files may be specified before the first file name and will then be applied to all subsequent files until overridden explicitly. For instance,

```
.PLOT =/COPIES:3 A.VEC,B.VEC
```

will plot three copies each of A.VEC and B.VEC. The command

```
.PLOT =/COP:3 A.VEC,B.VEC,C.VEC/COPIES:4
```

will plot three copies of A.VEC and B.VEC, but four copies of C.VEC.

Explicit specifications of a value by a user take precedence over all forms of defaults. Sticky defaults take precedence over SWITCH.INI defaults, which in turn override system defaults.

3.2.3 Unspecified Values — As mentioned above, there is information available to the command scanner concerning switches which must be specified for some translators. If there is no source of information for such a switch, including any form of default, the user will be prompted for that value.

3.3 Overall Logic

Figure 2 presents the overall logic of the generalized plot system, including the flow from the command scanner through the translators to the graphics device. Note that in several places in the diagram there are entries of "NETWORK QUEUE". These represent the transmission of the file to some other computer in the extensive network of which our DECsystem-10s are a part. For instance, some portion of the work for all the Calcomp plotters is done on IBM System/360s.

3.4 Error Correction

Few events are more frustrating to a user of a timesharing system than typing a long command string and having the operating system throw it all away because the user misspelled a switch. This problem is highlighted in this facility since there are so many switches which can be used. To alleviate this problem, the user is told of the mistake made and prompted for correct data. Among the errors which are correctable are:

1. null device,
2. wildcard device,
3. null switch,
4. unknown switch (name misspelled),
5. ambiguous switch (name misspelled or too abbreviated),
6. omitted switch value,
7. ambiguous switch value (keyword misspelled),
8. ambiguous switch value (keyword misspelled or too abbreviated),
9. zero or excessive length project or programmer number, and
10. no file satisfying a wildcard specification.

4. TRANSLATORS

4.1 General

The discussion so far has considered the user-software interface. That interface makes known to the plotting system a set of information describing the work to be done and the program which is to do that work. The remaining functions

ORNL/DWG/FED 78-866

PLOT SYSTEM LOGIC

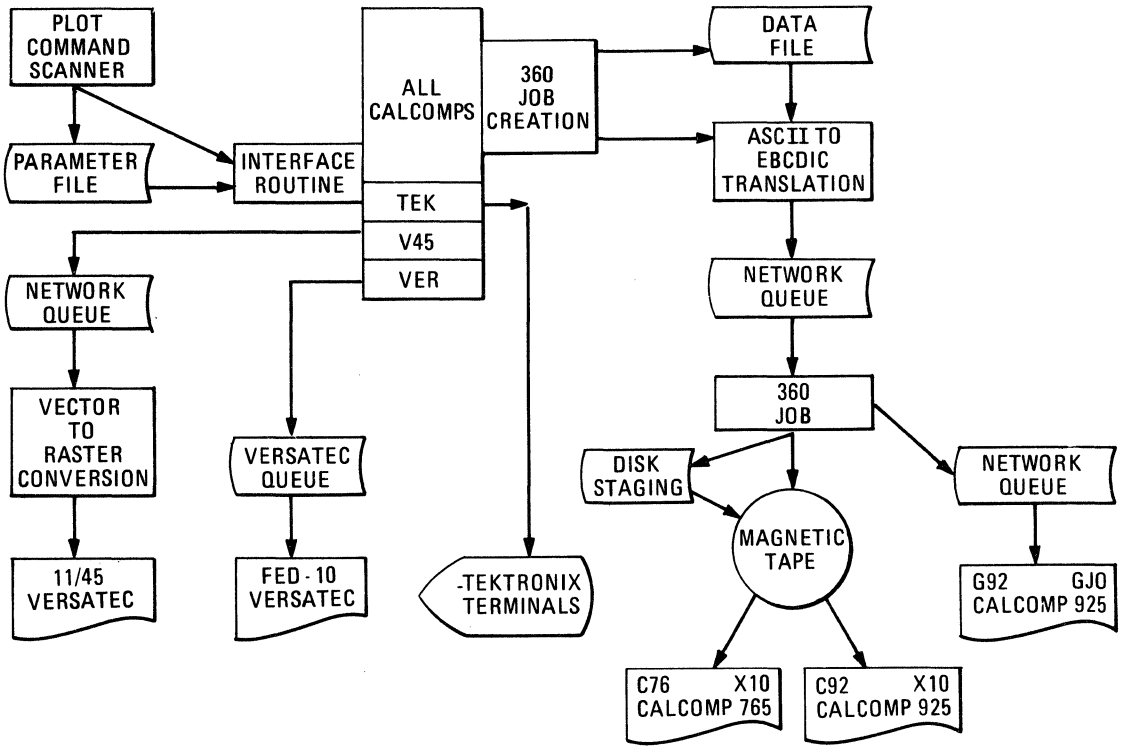


Fig. 2

to be performed are to save the information in a standard manner and to invoke the translation program. Each translator must be able to access the saved information, convert the identified graphics data to the required plotter format, and generate the plotter device access protocol.

The following steps are a simplified list of the flow of a general translator program:

1. Invoke and initialize the translator.
2. Access the information about the intermediate graphics file and specified switches.
3. Open a graphics data file.
4. Input a graphics data file buffer.
5. Decode the packed information into graphics file primitive information.
6. Translate graphics file instruction into device instructions.
7. Encode the device driving instructions into the device primitive format.
8. Output to device.
9. Repeat steps 4-8 until graphics file is exhausted.
10. Close graphics file and device.
11. Check for completion of job step and terminate or return to step 2.

4.2 Scanner-Translator Interface

There are three elements to the scanner-translator interface. First, the information identifying the graphics data and the required conversion must be saved in a format accessible to the translator. To do this a disk file is created, the name of which contains the number of the timesharing job assigned to the user doing the work, thus providing uniqueness to the disk file name. The specifiers are saved in the following order:

1. switches applicable to all graphics files in the request (global switches),
2. file specifications, including each of the remaining elements,
3. file structure,
4. file name,
5. extension,
6. PPN, and
7. switches relevant only to this request (file switches).

The entries are saved in a format consistent with their nature. File structures, names, and extensions are saved in SIXBIT, the PPNs are binary, and the format of the switch values depends upon their nature. The global switches are all ASCII, while most of the remaining switches take binary values. The global switches have positional values — the first six words of the parameter file contain the value of the NAME switch, the next ten contain the ADDRESS values, etc. The file switches are saved in the format:

1. SIXBIT switch name (truncated to six characters),
2. word length of the switch value (in binary), and
3. switch value.

Next, the proper translator must be invoked. The name of the translator is created by concatenating the extension of the file name and the plot device. For example, if the file SEGMNT.VEC is to be plotted on the Versatec, the name of the translator is VECVER, for VECtor file to VERsatec format. Control is passed to that translator by means of the RUN UUO, so that the entire memory space of the command scanner is overlaid by the VECVER translator.

Finally, the translator must be able to access the parameter file. A subroutine capable of this has been written and must be called by each translator. The calling sequence for the parameter access routine is

```
CALL PLPOST(I,J)
```

where

PLPOST stands for PLot POSTprocessor,
I is an integer code for the datum desired, and
J is the area in which the value for the datum is to be stored.

The calling program should be written to expect the datum in the format in which it is being passed, which is generally not the same as the format expected by the original translator.

4.3 Tektronix Processing

4.3.1 Translations — The graphics file to Tektronix translators are the least complicated of the programs discussed in this report. A specific decoder is used for each supported graphics file type which unpacks the graphics instructions. These instructions are then translated into synonymous instructions for the Tektronix storage tube. Vendor software, available from Tektronix (1), is used to encode the translated instructions into device primitives which actually control the position of the beam on the face of the Tektronix storage tube.

4.3.2 Plotter interface — The Tektronix storage tubes are treated as teletype ports into the PDP-10. As such, graphic postprocessing done at a terminal of this type is interactive. No spooling of job requests is required.

4.4 Versatec Printer/Plotter Processing

4.4.1 Translations — The graphics file to Versatec plotter translators are the most complicated of the programs discussed in this report. They are typically 7 to 10 times larger in executable module size than corresponding Tektronix translators, due specifically to the vector-to-raster conversion which must be performed. The same set of graphics file decoders is used to unpack individual file types. These instructions are translated into line segments. The first stage of the vector-to-raster conversion process maps all line segments into the raster buffer (which is currently an entire Versatec page 1024 by 800 dots). The second stage, which begins as soon as a new page is requested, compresses the raster buffer in preparation for transmitting this information to the Versatec plotter spooler. The compressed raster data are placed in another file and when all input has been exhausted, program control passes to the I/O interface which queues the compressed raster data to the Versatec plotter spooler.

4.4.2 Plotter Interface — At one of our installations a Versatec printer/plotter is used as the only hard copy output device, serving as both line printer and hard copy plotter. Two queues have been defined to serve that device, each queue having its own spooler. Given the queue for plot work and the spooler to handle that work, all that the translator needs to do is to enter its output in that queue. The rest of the processing is handled by the system.

At the same installation a second computer (a DEC PDP-11/45) is available with its own Versatec printer/plotter. If that Versatec is specified, the untranslated file is transmitted to the PDP-11/45 and converted to raster image there, thus providing much faster operation and reduced load on the DECsystem-10.

4.5 Calcomp Processing

4.5.1 Translations — The graphics file to Calcomp plotter translators are based upon a slightly different principle. Actually, these translators create a DISSPLA (2) compressed graphics data set from the input graphics data file. This scheme was chosen since a program which would transmit a DISSPLA compressed graphics file to the IBM System/360 and a program which would plot the same file on the remote Calcomps already existed (although not in a convenient form). Therefore, in keeping with the modest software development of the generalized plotting facility, the same set of graphics file decoders was used to unpack the data files. These graphics instructions are translated into line segments and "drawn" into the DISSPLA compressed graphics data set.

4.5.2 Plotter Interface — The Calcomp Plotters at our installation are both stand-alone plotters. That is, they are not physically connected to any computer. To do plotting the user puts the data on a magnetic tape which then is transported to the plotter. Because of various considerations, including the availability of operators, tapes, and software, all tapes for the Calcomp plotters are created by IBM System/360 computers. Since much of the plot development work is done or saved on the DECsystem-10, the generalized plotting system must be capable of generating a job to run on the IBM System/360s which will create a tape for the Calcomp plotters. Since the IBM System/360s are linked by communications lines and software to the DECsystem-10, the plot system can then enter the generated job into a queue for the IBM System/360s and let the system do the rest of the work.

The primary tenets of the generalized plot system apply to the job generating code as well as to the rest of the system. The user should be insulated from unnecessary detail and should be provided with as many defaults as possible. To that end, only a few mandatory data are defined and SWITCH.INI support for those data is provided (so that the user may define those data once and then forget about them). The required data are the user identifier (a 3-character field assigned administratively to provide job name uniqueness on IBM System/360 work), the charge number (which is used in IBM System/360 accounting and is a standard field), and the address of the user (which is used in routing output).

Given the above information, two general methods of providing a tape for the Calcomp plotters are available. For jobs doing relatively little plotting, a high turnaround method is available in which the graphics file is created with a standard name upon IBM System/360 direct access storage on a disk pack of a given name. At regular intervals the operators of the IBM System/360s invoke a program which searches that disk pack for all files of standard name construction and places their contents on a single tape for transport to the Calcomps.

Jobs creating large graphics files must create their own tapes in the interests of saving on-line direct access space. The generalized plot system, then, needs to build a different job, one which calls for the mounting of a tape and which causes the creation of a plot control card to accompany the tape to the plotter.

The creation of each type of job under the generalized plotting system obeys the same rules as the rest of the system — the user does not have to know what is going on. In fact, the user will never see the job control language until the output of the job is delivered.

It is important to note that the plotting system exists on two DECsystem-10s about six miles apart, which are linked by communications lines and spoolers so that either one may be used with the plot system with no modification of the procedure. If the user is connected to the remote DECsystem-10, an extra routine step is interposed (without the user's knowledge) which sends the generated job to the DECsystem-10 in the room next to the IBM System/360s, from which the job is entered into the IBM System/360 queue. The user cannot tell from the job output which computer was used to initiate the job.

5. SCENARIOS

5.1 General

Let us now consider the appearance of the generalized plot system to the user. We consider two users — the novice and the old pro. The novice is defined as a person who has no previous experience with the plot system, but who does know how to create a graphics file and has had enough experience with DECsystem-10 to know that HELP files exist for many facilities and that often a /H switch will provide assistance. The old pro has been using the plot system for a long time and knows about all the defaults and short cuts possible.

5.2 Novice

We begin with the novice user who has already created a graphics file, which we call SEGMNT.VEC. The user first types

```
.HELP PLOT (where PLOT is the command invoking
            the system)
```

and receives some general information about the system as a whole. Included in this file are some basic commands chosen to provide the user with some inkling of the range and depth of the

command structure. One of the commands given as an example is

.PLOT TEK:=SEGMNT.VEC

which will cause the user's file to be plotted on the user's Tektronix scope. Cheerfully, the user enters that command and finds that the resultant plot is not as intended.

Having found errors in the program and used the Tektronix plotting capability of the system to verify that the file is correct, the user now desires a hard copy of the graph. Hoping that a command like

.PLOT/HELP

will provide helpful information, as it does for many facilities on the DECsystem-10, the user tries it. The response is a list of categories of information available from which the user is asked to choose. Interested in finding out what plotters are available, the user responds appropriately and finds that a VECC92 processor is available (vector file to Calcomp 925/1036 plotter). (The HELP facility then prompts for the next category of information desired.) Continuing the dialog, the user finds that several switches are mandatory or recognized but not being in the mood to figure them all out simply types the command:

.PLOT C92:=SEGMNT.VEC

which the user hopes will take care of all problems and plot the file on the Calcomp 925/1036 plotter.

Unfortunately, further information is required before the plot can be made, since an IBM System/360 job must be created to do some of the translation. The user is then prompted for a user identifier, a charge number, and an address. The user is then pleased to find that the request is complete and the work is scheduled.

Flushed with success, the user decides to plot another file, one named FILE.ABC. Entering the command

.PLOT C92:=FILE.ABC

the user is disappointed to note that no translator ABCC92 exists and that prompts for correct device and file extension have been issued. Knowing that the information was correctly entered and knowing that the file is really in the DISSPLA POP format the user renames the file to FILE.POP and types

.PLOT C92:=FILL.PPP

misspelling grievously. The plot system announces that no such file exists and prompts for correction. When the user responds correctly, the system again prompts for user identifier, charge, and address before scheduling the work.

5.3 Old Pro — Our old pro knows all about the generalized plot system. Long ago this user tired of responding to prompts for switches and built a SWITCH.INI file containing the line

PLOT/DEVICE:TEK/UID:RDB/ADDRESS:"9201-2 Y-12"/ CHARGE:12345

so that the IBM System/360 jobs required for various applications could be easily built. Furthermore, this user knows about the default file names and devices for each installation, so to plot a SEGMNT.VEC file on a Tektronix scope this user types only

.PLOT

The device field and mandatory switch values will be provided by the SWITCH.INI field while the file name and extension fields will be provided by the system defaults.

When all the bugs are out of the application and the old pro wants a Calcomp plot, the SWITCH.INI file is modified to contain /DEV:C92 so that the command

.PLOT

takes care of it with no prompting. The old pro also knows about the copies and forms capabilities of the system and about the multiple pens available on the Calcomp 925, so when desiring to take advantage of these features the pro types

.PLOT =A.VEC/COPIES:3/FORMS:602/INK:(GREEN/L, BLACK,RED/L)

which causes the file A.VEC to be plotted three times on 602-type forms with the three pens set to liquid green ink, black ballpoint pen, and liquid red ink, respectively.

6. SYSTEMS MAINTENANCE

In a system as complex as this one, considerable effort should be directed to making the system easy to maintain and to expand. Part of the means of doing so was to use existing software wherever possible and to make only minimal changes to interface to the new regime. But various new facilities were added, including the automatic application of various translator-specific defaults and the definition of mandatory switches for each translator.

The new facilities are largely supported by means of tables which are kept on disk. One file, the PLOT.HLP file, contains the list of supported translators, mandatory and legal switches for each one, and special information about each translator, as well as the system default specifications for plot device, graphics file name and extension, and the plot command (used when the user types only "PLOT"). In that file the translator names are SIXBIT, the switch fields are in the form of bit maps, and the default command information is ASCII. A program named HELPFI was written to create and maintain this file.

The translator-specific defaults are kept in separate .HLP files with the same names as the translators themselves. That is, the defaults file for the VECVER translator is named VECVER.HLP. This file is created by the same program which maintains the PLOT.HLP file, but it is ASCII. It contains the list of default values in the form of switch strings recognizable by the SCAN

program, lists of mandatory and legal switches and of special information, and an ASCII text string providing additional information. Since this file is in ASCII no update provisions are made for it — the system maintainer needs only to use some editor, such as TECO, to modify it.

In accordance with the design specifications for this system, the HELPFI program was designed for ease of use. The maintainer is prompted for each entry and each is validated. The maintainer may create, delete, or modify translator entries.

7. CONCLUSIONS

The generalized plot system described in this paper provides to the user of a DECsystem-10 the capability to use virtually any graphics support package provided with any plotter available. The user is not required to know very much about the system and the most probable user errors are correctable via prompts. The system has been designed for ease of maintenance and expansion, so that minimal support effort is required.

REFERENCES

1. Tektronix Plot 10 Terminal Control User Manual, Tektronix, Inc., Beaverton, Oregon (1976).

2. DISSPLA (Display Integrated Software System and Plotting Language) is a proprietary software library of graphics routines copyrighted by Integrated Software System Corp., San Diego, California.

ACKNOWLEDGEMENTS

Reid Gryder, Kathe Fischer, Betsy Clark, and Charles Thompson all made valuable suggestions concerning the design of this system. Ms. Rose Ann Pemberton prepared the manuscript. Our thanks to them all.

Design Considerations and Philosophy of a Device-Independent Publications/Graphics System

Jean S. Burt
National Nuclear Data Center
Brookhaven National Laboratory
Upton, New York 11973

ABSTRACT

Over a period of ten years the National Nuclear Data Center has implemented graphics systems to meet a broad range of user requirements in the areas of interactive graphics, publications and to a lesser extent, text-editing, graphical data interpretations and on-line data evaluation. The systems have been designed to support varying levels of user sophistication with respect to programming ability and user knowledge of the hardware involved.
This paper will present an overview of the NNDC's graphics system which is available to the user via a higher level language, FORTRAN. The system has been designed using layers of software between the user and the device-dependent code. One layer is dedicated to processing the incompatibilites and inconsistencies between such devices as paper plotter, interactive graphics and FR-80 microfilm/microfiche hardware. Another handles the niceties necessary for finer quality pulications work, e.g. superscripting, subscripting, boldface, variable character/page sizing, rotation, the use of multiple character sets (e.g. mathematical, Greek, physics) as well as features to allow the user to design special characters.

BACKGROUND

The National Nuclear Data Center (NNDC) at Brookhaven National Laboratory is designated by the U.S. Department of Energy to provide a wide spectrum of information services covering the entire field of low energy nuclear data to requestors within the United States and Canada(R1). The scope of the services performed by the Center ranges from issuing newsletters and publications to providing technical assistance of a scientific as well as computer nature to conducting seminars on special topics. The NNDC also interfaces with and coordinates similar groups nationally and internationally in the periodic revision of documented and computerized reference data libraries to insure completeness and accuracy of information (Figure 1).

The NNDC has approximately 30 staff members of which 20 are at the scientific or professional level (Figure 2). It maintains its own dedicated computers (KA-10 & PDP-15 subsystem) for support and maintenance of bibliographic and experimental data libraries and servicing of requests. These requests may be answered by retrievals from one or more of the Center's data libraries and may include documents (write-ups, graphs, fiche), computer output listings and/or magnetic tapes generated according to individual requestor specifications. More general needs are often satisfied by one of the Data Center's many publications.

Until recently, the NNDC computer needs were fulfilled by a DEC-10 KA processor with 144K of memory. Periperals included two line printers, card reader, a TU20 tape subsystem (2 7tr and 1 9tr), 6 RP02 disk drives and 4 dectape units. A PDP-15 with CRT display, graph tablet and lightpen functioned as the interactive graphics subsystem, while an incremental-drum mechanical plotter completed the configuration (Figure 3).

The Data Center is presently being upgraded to a KL-10/1091. The dectape units, printers, card reader and plotter are being carried across, as is the PDP-15 subsystem and its developed software. A TU70-71-72 tape subsystem (2 9tr and 1 7tr)and 4 RP06 drives are replacing the prior tape and

disk subsystems, respectively. By the acquisition of this new hardware, the Center intends to meet its objective of providing on-line access to computer codes and to non-confidential data libraries within the next few years.

Design Considerations

The NNDC had three choices for selection as output media: a drum plotter (CalComp-563, 30" width), a 16mm/35mm CRT/microfilm subsystem (initially a CalComp-835 replaced by Information International Incorporated's FR80/COMP80 system, which then afforded microfiche capability) and lastly, a CRT display. Each of these was examined to determine basic physical limitations and individual operating characteristics. Character height-to-width ratio differed as did character formation. Cnaracter generation on the paper plotter and the CRT display are accomplished by stroke definition; on the FR80/COMP subsystem the characters could be mapped into the predefined FR-80 stroke symbol table. (The facility for character generation via user-defined stroking does exist in the NNDC software package.)

The next consideration concerned the Data Center's basic function, that of providing information services to requestors. This implied visual presentations that were clear to the requestor, conveyed concisely the information sought and were of a quality that spoke well for the Center, as a physics community servicer. Therefore, in addition to the normal alphabetic and numeric character sets, provisions had to be made for the inclusion of the Greek symbols which comprise a large portion of a physicist's vocabulary, as well as the mathematical symbols used in numerical analysis and Monte Carlo techniques in order to report those program problem solutions.

The fact that the problem solutions primarily addressed a scientific community meant that certain special effects would be needed in any publication and technical report or newsletter. The physical meaning of a decay scheme or mass chain and associated isotopic representations are conveyed by a specific type of display (Figure 4, R2). The graphics software had to provide superscripting and subscripting features as well as upper and lower case capabilities. Additionally, as seen in this primarily textual report (Figure 5, R3) boldface was required in addition to page rotation, variable page layouts and the resultant recalculation for possible character resizing. (Figure 6, R3). Lastly, this transparency exemplifies the type of axis annotation and multiple x-y data reporting demanded within results of actual experimentally-measured data (Figures 7 & 8, R4).

Finally, consider for whom the graphics package was to be written: a user. User may denote an applications programmer who writes the graphics programs and generally understands the machine internals and the techniques of making a graphics device function. User might also imply the end-user; one who may be a non-computer oriented person, a pure physicist with minimal but self-serving knowledge of a programming language. Finally there are those individuals in the middle of the spectrum, people who do a fair amount of programming and data handling, but are not computer types by avocation.

Consequently, after assembling these factors and compiling the range of features to be included, the philosophical aspects of the design of the graphics system for the NNDC took root. The package must support a singular graphics device or combinations of devices for simultaneous output, yet remain device-independent. This would enable the user/programmer, regardless of personal ability, to easily utilize/implement the system and to concentrate on his particular application rather than the peculiarities of an individualdisplay medium. High quality results had to be achieved with maximum flexibility in number and type of features available. Lastly, and obviously, the 'cost' of the graphics software had to be minimized, cost being measured by speed of programming, ease of implementation into existing codes, maintainability and the ability for future interface of new features (Figure 9).

Approach

Initially three plotting packages, one for each physical device (subsystem), evolved for use on the PDP-10; each had identical FORTRAN-callable software interfaces in terms of subroutine name as well as number and order of arguments. These were entitled LIB40, PLOTS and PLOT15 (Figure 10a).

LIB40 is the PDP-10 FORTRAN-40 Library Subroutine plotting package. Its available routines are described in any one of DEC's FORTRAN Language Manuals. Consequently, its features will not be discussed at any length. These routines are equivalent to those available from CalComp (California Computer Products) and are basically used to produce plots on an incremental drum plotter.

PLOTS is the programming package consisting of subroutines designed to perform the basic x-y plotting functions on a microfilm/microfiche subsystem. Multiple film sizes and fiche reductions are available, the most commonly employed being 16mm and 35mm for film, and 42X or 48X for fiche. Certain routines calculate the plot data and then format the plot data into lines and symbols; still others format

lines and symbols into commands to write the magnetic tape which later drives the plotter.

The PLOT15 (R5) program package and its associated interactive-graphics software (AUX15) are written utilizing a master-slave concept whereby the PDP-10 acts as the master and the PDP-15 is the slave. The PDP-10 initiates all communications between the computers via the DA-10 hardware. (The DA-10 hardware is a basic interrupt-driven buffered device that permits transfer of data on a word basis via the iobus. The 36-bit word from the PDP-10 results in two 18-bit PDP-15 words.) On the PDP-15 low core (0-200 octal) is reserved for the device handler that facilitates the transmission of data between the two computers; this is written in MACRO-15. High core (201-37777 octal) is reserved for the storage of the display file. After initialization, as each input word from the PDP-10 is read, the display file is immediately updated; the display is not stopped during the procedure so the continuous picture is shown. This process of inserting one word at a time is continued until all input words are inserted or the PDP-15'S available core is filled, at which point an error message is sent and any remaining transmission is ignored (R5). Note that basically the PDP-15 is functioning as a buffer and the PDP-10 is loading it and dispatching the final display.

Interrogation of the PDP-15 with respect to raster position of the lightpen cursor, VT-15 pushbutton status, next available core location for the display file, etc. are accomplished through the AUX15 routines. These routines also include features for buffer management of the picture to accomplish subpicturing and selective erasure of the display.

Solution

The programming for rapid implementation of simultaneous plotting on any or all of the three graphics subsystems was instituted in the following sequence:

1) To obtain a complete set of subroutines with unique, but device-identifying names, the routines from the existing systems were either renamed by the addition of a character at the end of the name, shortened or changed to avoid conflict with popular names.

2) A labeled COMMON area was used to define three flags to indicate which devices were 'turned on'; if the device flag was zero, it signified the device is 'off'. The non-zero state designated device 'on' and created output for that unit.

3) A new set of general-purpose subroutines with the identical names and calling sequences as existed in any of the original systems was written to minimize chance of error. Routines not applicable to a device substem were included as dummies to maintain consistency. Internally these routines used the labelled COMMON to decide which actual graphic routine to call, using the subroutine's renamed name.

With the adoption of these conventions the user need only be concerned with initially defining a device as 'on' or 'off'. He then proceeds to program, implementing the appropriate calls to the functional subroutines SYMBOL, NUMBER, PLOT, etc., as if only one device were being used. This system was entitled PLOT3, to symbolize the use of the three devices (Figure 10b).

PLOT3 directs control to a particular device. It 'sits' at the top of the sections of code dealing with the devices and separates them from the bulk of the user-code. This particular sequence of intercommunication, i.e. user PLOT3 device, insures that the routines remain independent of each other but at the same time are able to communicate with each other via deliberately established and organized operands and arguments, thereby avoiding direct user involvement. PLOT3 also contains defensive code to protect the user from himself and the occurrence of certain common errors. Two such examples are automatic advance to eliminate overplots and checks against on,mid-plot reinitialization.

Occupying the same layer as PLOT3 yet distinct from it, since its development is more recent, is FANCY, a collection of functionally-specific subroutines (R7). Through the use of subroutine calling parameters, these provide special feature selection such as titling, additional character set definitions (symbol table position mappings via ASYMBL, symbol stroke definitions via PENMOV, user-defined special character stroking via SPCHR) and axis labeling and scaling capabilities. A command menu and parameter-selection routine can be invoked to handle the actual device initialization and character selection and assembly. Thus the user, in response to a quiry sequence at run time, can use the code to layout his basic publication.

Scenario of an Application

To demonstrate the flexibility of the graphics software, consider the steps taken in assembling and refining the data for one of the NNDC's best-known publications, BNL325, Neutron Cross Sections, Vol. II, otherwise referred to as "The Book of Curves."

The appropriate data in the form of bibliographic as well as data point tables, is retrieved from the Evaluated Nuclear Data files (ENDF) and the CSISRS Experimental Data files. Another program combines these multiple files into a single

file while generating indices of the isotopic and energy range contents of the merged file.

The experimental and evaluated data points are then displayed on the PDP-15 CRT (PLOT3-PLOTD). A physicist then examines the data, certifying the credibility against individual expertise and supplemental resonance parameter data. The data and possibly selected portions of it, are culled and refit to obtain a smooth curve. During this iterative process, picture manipulations and alterations are accomplished via lightpenning and appropriate menu selection (AUX15). Certain portions of the display may be zoomed for closer examination; other subpictures may be deleted entirely. After the curve analysis is completed, 'eyeguides' are generated and final data values extracted, redisplayed and written to an updated data file which is in turn plotted on the paper plotter (PLOT3-PLOT5). (Eyeguides are created for publication as an attempt to delineate the main features of the data. R4) These plots are then evaluated and depending upon those results, either redisplayed, refit and replotted or denoted as finalized for publication.

After all the data files have been analyzed in this manner and it has been determined that the best possible fit for data has been achieved, the resultant pictures are plotted on microfilm (PLOT3-PLOTS). Negatives for use for future printing are made; in a separate run, microfiche can be generated to provide an inexpensive and easily accessible storable medium.

(Note: the PLOT3-subsystem parenthesized expressions are merely notations to show the interrelationship and behavior of the package.)

Conclusions

This design structure originated approximately 8 years ago, and while the software has been modified and enhanced over the years, the basic concept has remained the same. The beauty of the layered design is in its conceptual simplicity (Figure 12). While most graphics and publications packages, especially those involving sophisticated interactive graphics, take considerable amounts of people, hardware and software resources, this is untrue of the NNDC graphics software.

The man-years used to develop and maintain the system have been small and yet the system's functionality and flexibility are tremendous. The NNDC graphics package is not a sophisticated software package, but merely an adherence to the basics of good programming technique. That implies modularity: the consolidation into specific places within the code, all of each particular type of function. Clarity and ease of programmer assimilation are accomplished utilizing significant in-line documentation through the inclusion of comment statements to explain segments of the coded logic. Coupling this with standard FORTRAN usage has enabled sections of the code to be transported across machines. Additional features can be easily interfaced into PLOT3 as additional modules via the FANCY routines. Maintainability has been shown in that the package has accommodated changes in computer configuration with little or no adjustment in the software or on the part of the user and has adapted to changing conditions over the course of time.

In the course of designing publications the basic goal has been to convey the maximum amount of information in the most visually comprehensive manner. Whenever the Data Center has been faced with that type of challenge, the package has permitted higher level routines to be developed with no change in the basic software. Through these means, the NNDC has met its obligations to the scientific community in the quality of services rendered.

References

R1: An Introduction to the National Nuclear Data Center, S. Pearlstein, Director

R2: ENDF/B Fission Product Decay Data, P.F. Rose, T.W. Burrows, August 1976

R3: The Bibliography of Integral Charged Particle Nuclear Data, T.W. Burrows, J.S. Burt, March 1978, 2nd Edition

R4: BNL325, Neutron Cross Sections, D.I. Garber, R.R. Kinsey, Vol. II., Experimental Data Plotted Against Eyeguides/Evaluated Curves, January 1976, 3rd Edition

R5: PLOT15 System - Version 72-1: D.E. Cullen

R6: PLOT3 System - Version 72-1: D.E. Cullen, W.H. Kropp

Acknowledgements

The author wishes to thank Bruce Stype of the Brookhaven Graphic Arts Department for creating and reproducing the illustrations needed for this paper and also to William Kropp for supplying historical information on the previous plotting packages and final proofreading of this document.

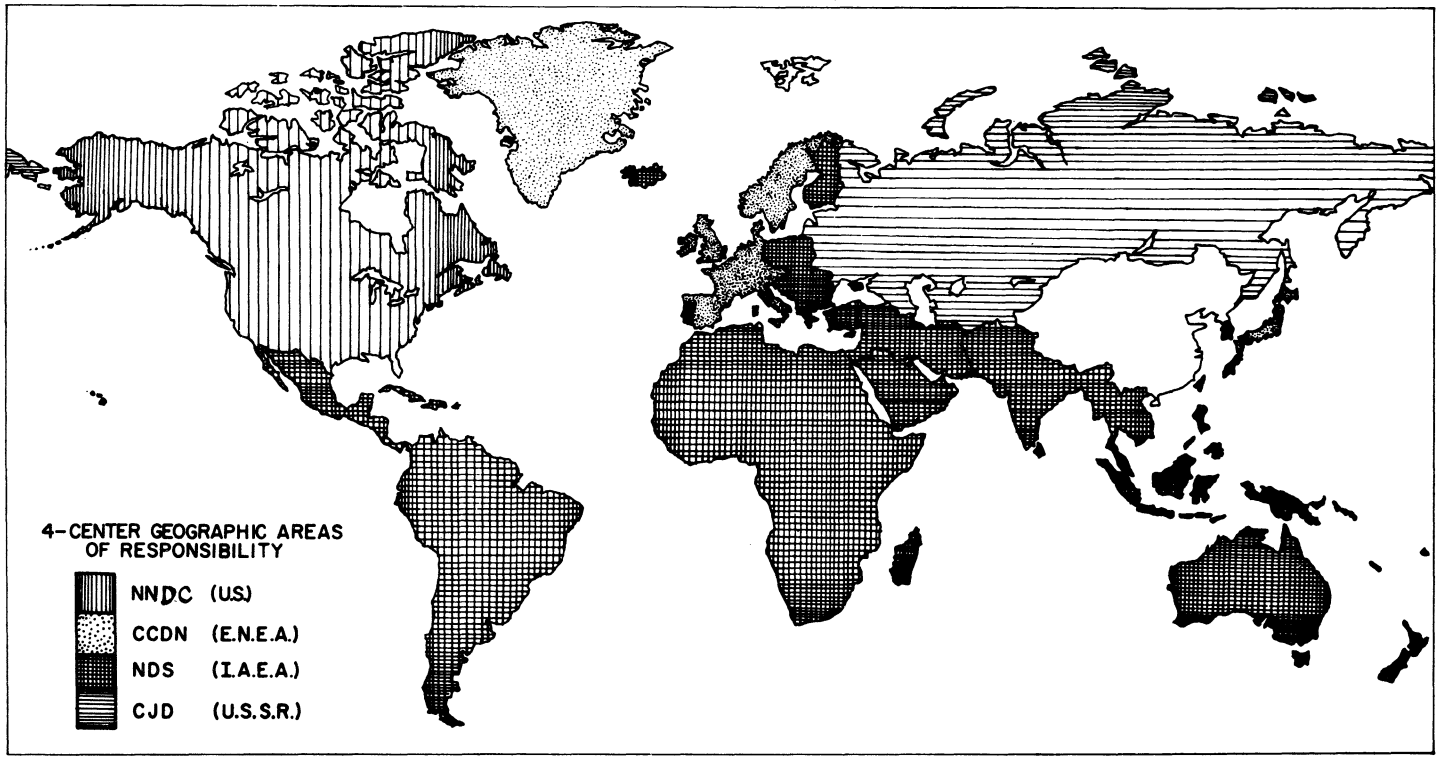


FIGURE 1

Neutron Cross Section Data is collected, stored, and disseminated according to an International agreement involving 4 centers. Requests for Neutron Data should be directed to the center responsible for your geographical area.

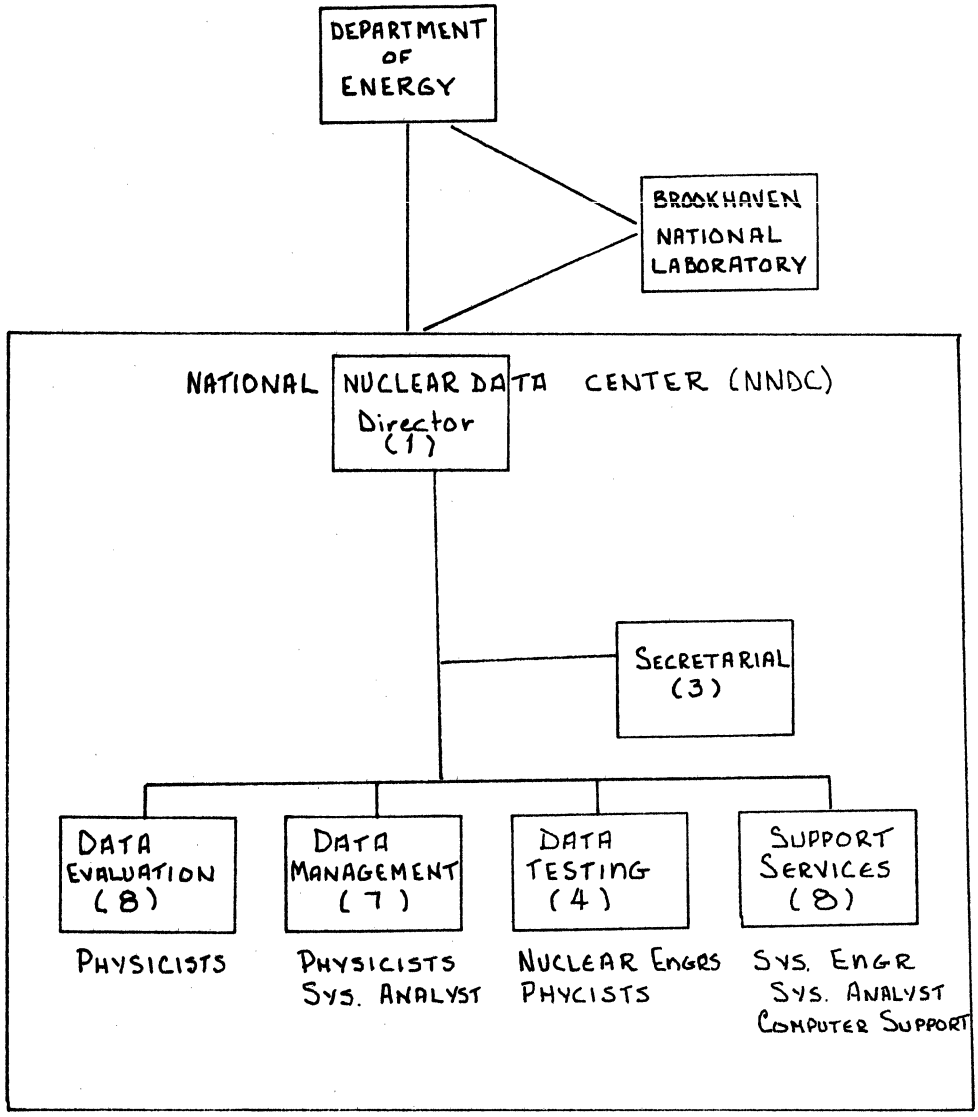

DEPARTMENT OF ENERGY
BROOKHAVEN NATIONAL LABORATORY
NATIONAL NUCLEAR DATA CENTER (NNDC)
Director (1)
SECRETARIAL (3)
DATA EVALUATION (8)
PHYSICISTS
DATA MANAGEMENT (7)
PHYSICISTS
SYS. ANALYST
DATA TESTING (4)
NUCLEAR ENGRS
PHYCISTS
SUPPORT SERVICES (8)
SYS. ENGR
SYS. ANALYST
COMPUTER SUPPORT

FIGURE 2

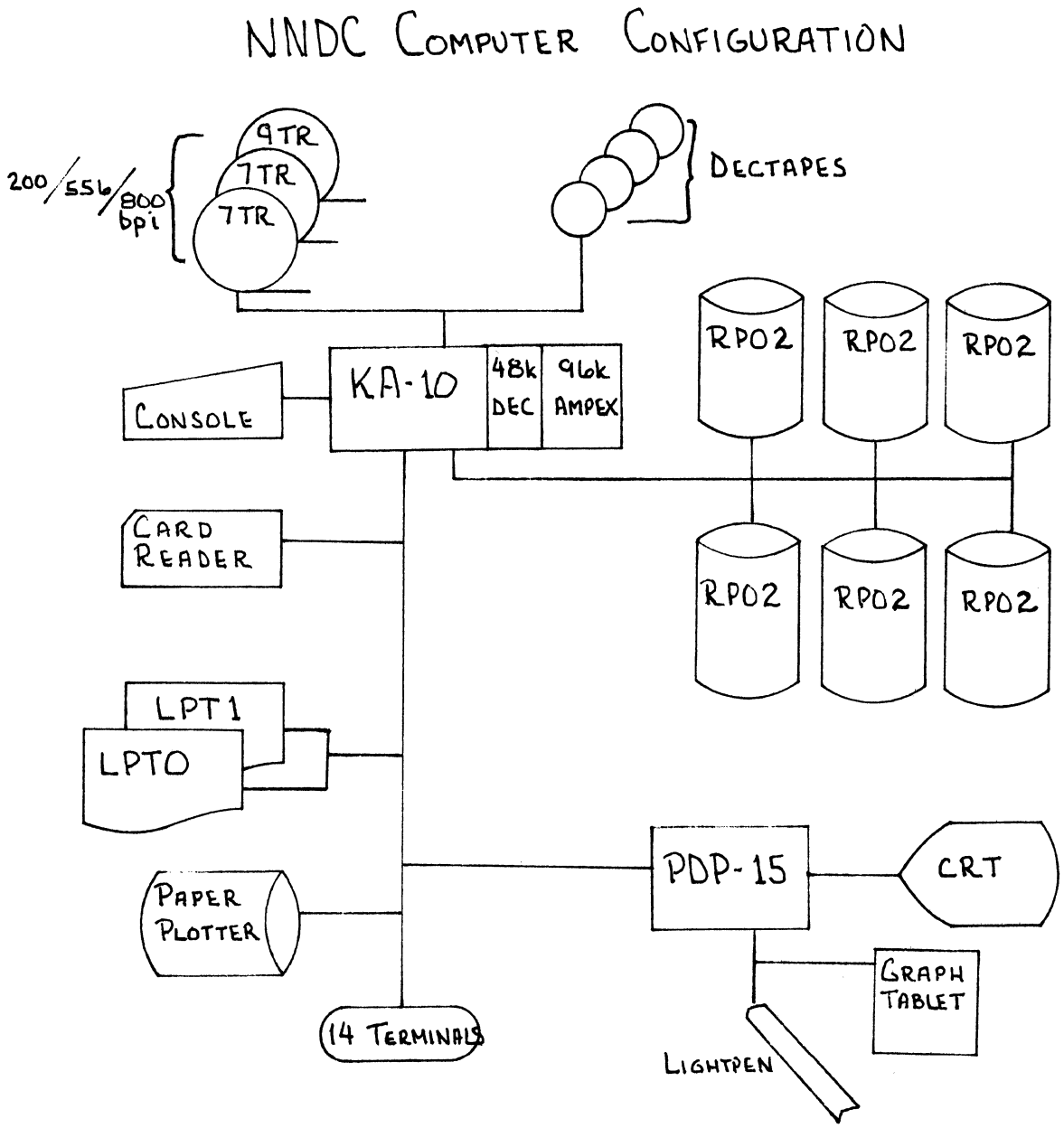
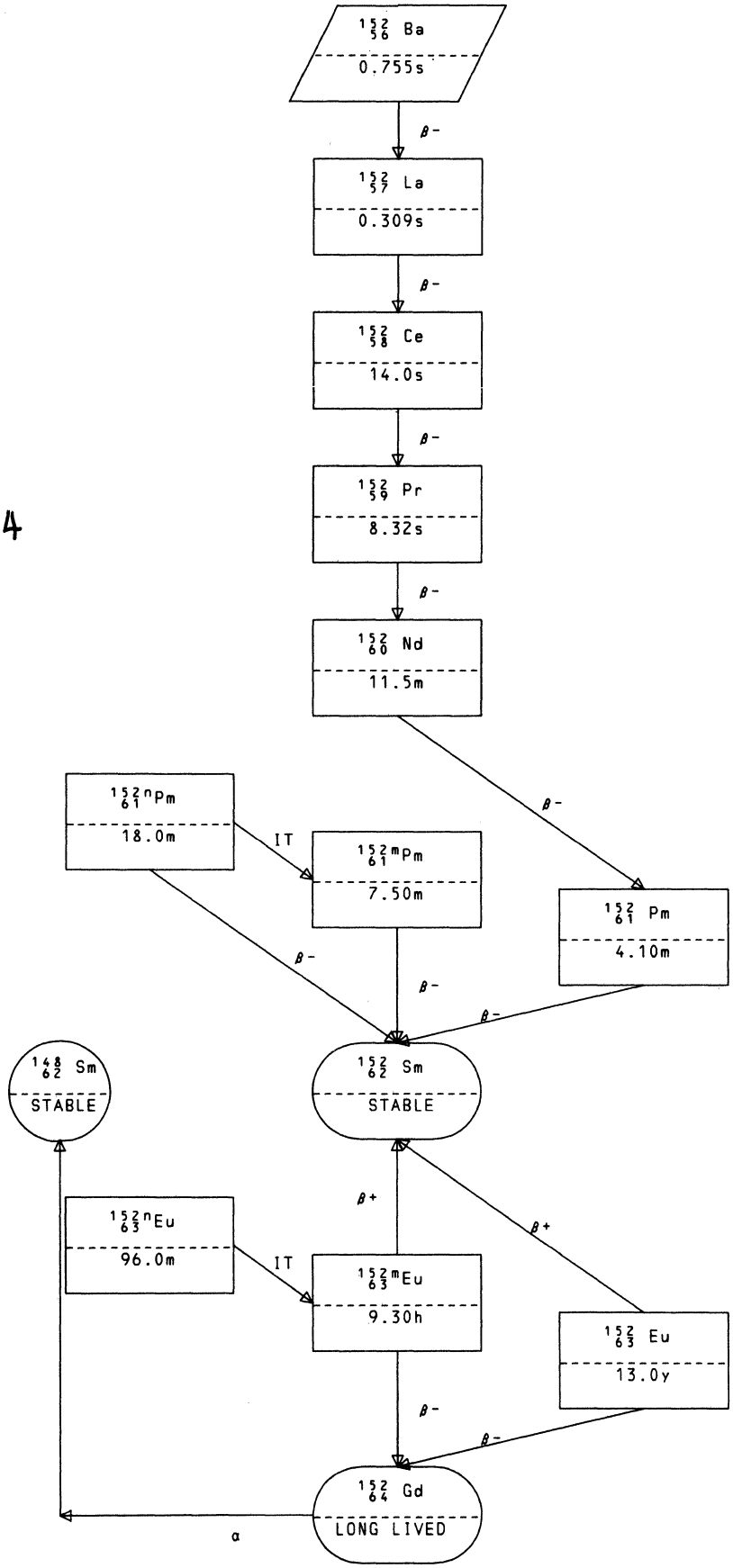

NNDC COMPUTER CONFIGURATION
200/556/800 bpi
9TR
7TR
7TR
DECTAPES
RP02
RP02
RP02
CONSOLE
KA-10
48k DEC
96k AMPEX
CARD READER
RP02
RP02
RP02
LPT1
LPTO
PDP-15
CRT
PAPER PLOTTER
GRAPH TABLET
14 TERMINALS
LIGHTPEN

FIGURE 3

FIGURE 4



152-0

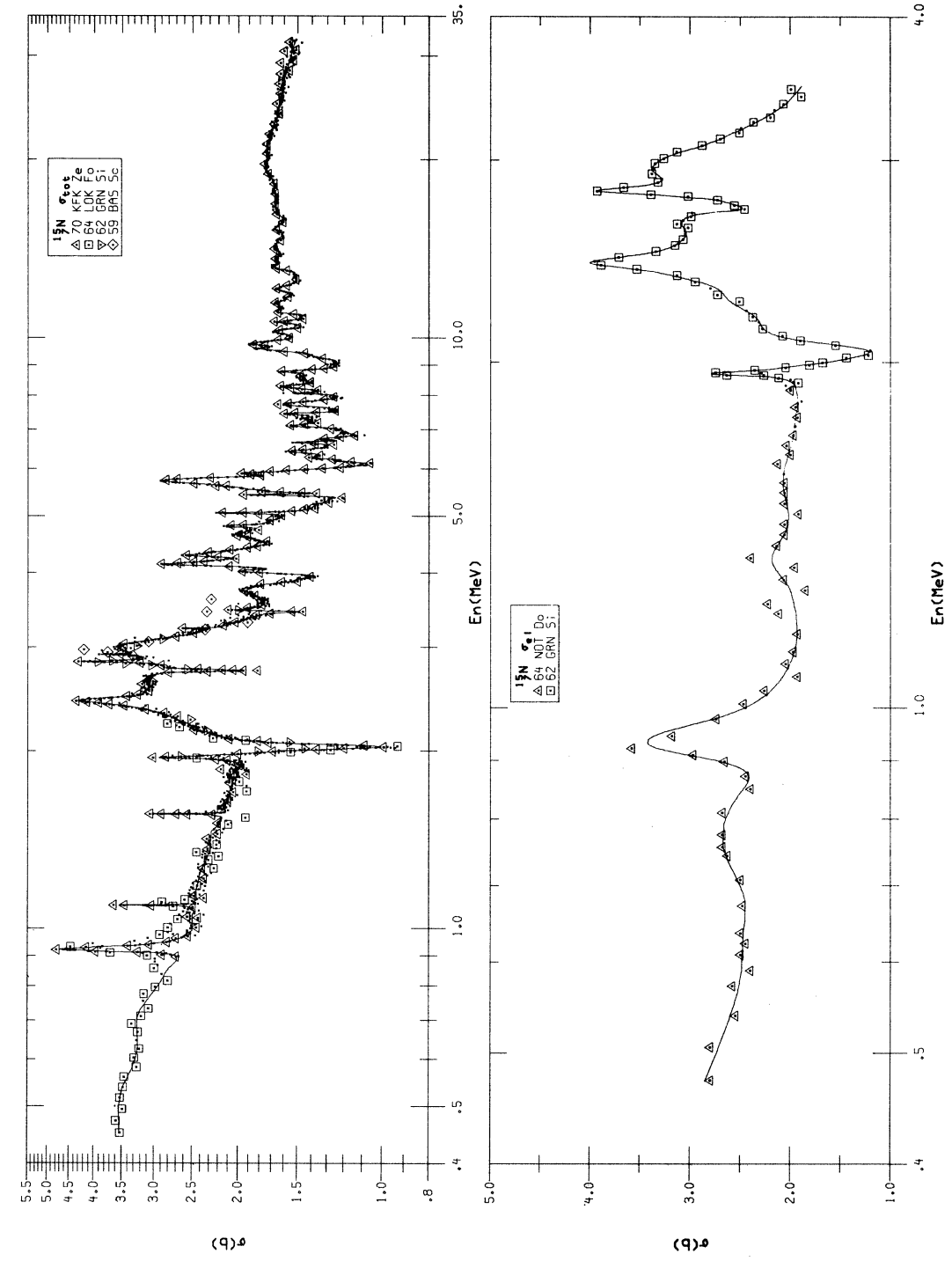
No.	Lab	Work Type	Reference			Emin (MeV)	Emax (MeV)	Author, Comments
			^{63}Cu(p,n)^{63}Zn σ(E)					
1032	BNL	Expt J	PR/C 9	1819	574	4.0+0	2.5+1	Colle+ TBL. CURV. ACTIVATION METHOD.
		Expt 4	EXFORB0057.002		77	4.0+0	2.5+1	. 49 DATA POINTS.
1033	ETH	Expt 4	EXFORB0048.002		77	4.2+0	6.3+0	. 9 DATA POINTS.
1034	ORL	Expt P	ORNL-2910	25	460	4.2+0	5.8+0	Johnson+ CURV. 4PI NEUTRON DET.
1035	ROC	Expt 4	EXFORB0051.		77	4.2+0	6.6+0	. 10 POINTS.CPX DATA CORRECTED BY COMP
1036	CLA	Expt 4	EXFORB0060.002		77	4.5+0	1.1+1	. 15 DATA POINTS.
1037	HRV	Expt 4	EXFORB0054.004		77	5.0+0	9.9+1	. 16 DATA POINTS.
1038	MUN	Expt J	NP/A 198	625	70	8.7+0	1.6+1	Hille+ TBL. CURV. STCKD FOILS.
		Expt 4	EXFORB0058.002		77	8.7+0	1.6+1	. 8 DATA POINTS.
1039	ANL	Expt J	PR/C 7	1410	473	1.5+3	1.2+4	Steinberg+ TBL. CURV.
		Expt 4	EXFORB0031.002		77	1.5+3	1.2+4	.7 ENTRIES.
			^{63}Cu(p,n)^{63}Zn σ(E)xFactor					
1040	HIR	Expt J	JIN 39	1923	77	3.0+1	5.2+1	Noma+CURV.RAT ZN61/ZN63.STCKD-FOIL+ISOL
			^{63}Cu(p,γ)^{64}Zn cumulativeσ relative product yield					
1041	KFI	Expt J	JP/G 2	365	76	3.1+0	3.3+0	Fodor+ CURV. 2KEV STEPS. GE(LI)
			^{63}Cu(p,γ)^{64}Zn σ(E)					
1042	CAL	Theo R	OAP-422		875	TR	1.2+1	Woosley+ CURV. HAUSER-FESHBACH.
			^{65}Cu(p,absorption) σ(E)					
1043	IJI	Revw J	FCY 6	827	75	6.8+0	1.5+1	Nemets+ TBL.
		Revw J	SJPN 6	325	76	6.8+0	1.5+1	. TRANSLATION OF FCY 6,(4),827(1975).
			^{65}Cu(p,total) σ(E)					
1044	IJI	Revw J	FCY 6	827	75	6.8+0	1.5+1	Nemets+ CURV. DEPENDENCE ON N AND Z
		Revw J	SJPN 6	325	76	6.8+0	1.5+1	. TRANSLATION OF FCY 6,(4),827(75).
			^{65}Cu(p,α)^{62}Ni σ(E)					
1045	CAL	Theo R	OAP-422		875	1.1+0	8.1+0	Woosley+ CURV. HAUSER-FESHBACH.
			^{65}Cu(p,4n+p)^{61}Cu σ(E)					
1046	HRV	Expt 4	EXFORB0054.008		77	4.5+1	9.9+1	. 12 DATA POINTS.
			^{65}Cu(p,3n+p)^{62}Cu σ(E)					
1047	HRV	Expt 4	EXFORB0054.007		77	2.5+1	9.9+1	. 17 DATA POINTS.
			^{65}Cu(p,n+p)^{64}Cu σ(E)					
1048	HRV	Expt 4	EXFORB0054.006		77	9.8+0	9.9+1	. 18 DATA POINTS.
1049	BNL	Expt J	JIN 38	23	76	1.3+1	2.5+1	Colli+ SINGLE AND STCKD FOILS.
		Expt 4	EXFORB0059.002		77	1.3+1	2.5+1	. 19 DATA POINTS.
1050	BNL	Theo C	77BNL	491	77	1.5+1	7.0+1	Divadeenam. CURV. GDH+PREEQUILIBRIUM.
1051	MCG	Expt W	MEGHIR+	66	1.9+1	8.5+1	Meghir+ TBL IN PR 144,962(1966)	
		Expt 4	EXFORB0016.002		076	1.9+1	8.5+1	. 19 ENTRIES.
1052	ORL	Expt J	PR 99	718	55	2.2+1		Cohen+ TBL.
1053	ORL	Expt 4	EXFORB0050.012		77	2.2+1		. 500MB. REL. UNC.=15%. CALIB. UNC.=15%
1054	MCG	Expt J	JIN 35	361	73	2.3+1	1.0+2	Newton+ TBL. CURV. ACTIVATION.
		Expt 4	EXFORB0023.002		77	2.3+1	1.0+2	.19 ENTRIES.
1055	MCG	Expt J	CJC 55	3609	077	3.5+1	8.5+1	Galinier+TBL.MONITOR FROM JIN 35,361(73
			^{65}Cu(p,n+p)^{64}Cu σ(E)xFactor					
1056	ORL	Expt J	PR 99	718	55	2.2+1		Cohen+ TBL. CURV. RATIO(P,NP)/(P,2N).
			^{65}Cu(p,inelastic)^{65}Cu partial σ(E)xFactor					
1057	IJI	Expt J	YF 24	461	976	2.0+0	3.0+0	Krivonosov+ CURV. 771KEV LEVEL.
		Expt J	SNP 24	239	77	2.0+0	3.0+0	.TRANSLATION OF YF 24(3), 461(1976).
			^{65}Cu(p,4n)^{62}Zn σ(E)					
1058	HRV	Expt 4	EXFORB0054.010		77	3.4+1	9.9+1	. 11 DATA POINTS.
			^{65}Cu(p,3n)^{63}Zn σ(E)					
1059	HRV	Expt 4	EXFORB0054.009		77	3.0+1	9.9+1	. 11 DATA POINTS.
			^{65}Cu(p,2n)^{64}Zn σ(E)xFactor					
1060	ORL	Expt J	PR 99	718	55	2.2+1		Cohen+ TBL. CURV. RATIO(P,NP)/(P,2N).
			^{65}Cu(p,n)^{65}Zn product yieldxFactor					
1061	WMU	Exth J	PR/C 15	1592	477	2.2+0		Bernstein.NDG. CAL. GEOM. CORR TO E(TH)

FIGURE 5

Target	Inc	Emin (MeV)	Emax (MeV)	Lab	W	Author	No.
			α partial σ(E)				
^{6}Li	d	1.0-1	1.0+0	ANL	E	Elwyn+	2262
^{12}C	^{14}N	6.7+1		TAM	M	Stokstead+	4791
			α compound-nucleus σ(E)				
^{3}H	d	NDG		LAS	D	Hale+	2240
^{3}He	d	NDG		LAS	D	Hale+	2245
^{4}He	p	NDG		LAS	D	Hale+	21
^{12}C	^{3}He	3.0+1		BRK	E	Pape+	2686
^{12}C	^{14}N	8.7+1		TAM	M	Stokstead+	4792
^{12}C	^{14}N	8.7+1		TAM	M	Stokstead+	4793
^{12}C	^{14}N	8.7+1		TAM	M	Stokstead+	4794
			α direct-interaction σ(E)				
^{12}C	^{3}He	3.0+1		BRK	E	Pape+	2687
^{51}V	^{16}O	7.7+1		GRE	E	Chevarier+	5240
^{50}Cr	^{14}N	7.1+1	1.0+2	GRE	M	Billerey+	4867
^{55}Mn	^{12}C	7.7+1		GRE	E	Chevarier+	4595
^{56}Fe	^{16}O	7.7+1		GRE	E	Chevarier+	5260
^{60}Ni	^{12}C	7.7+1		GRE	E	Chevarier+	4607
			α relative σ(E)				
^{7}Li	p	1.6+0	1.2+1	UPP	E	Sandhu+	37
^{12}C	^{12}C	9.0+0	2.8+1	SAC	R	Papineau.	4482
			α σ(E)xFactor				
^{3}H	d	5.0-1	2.0+1	LAS	D	Drosg.	2241
^{3}H	t	6.1-2	1.9-1	CCP	E	Serov+	2630
^{6}Li	p	7.5-2	3.1-1	CAL	R	Tombrello.	33
			α partial σ(E)xFactor				
^{7}Li	d	1.9+0	2.1+0	IJI	E	Grantsev+	2281
			α spectrum average σ(E)xFactor				
^{6}Li	d	1.0-3	1.0+0	ANL	T	Elwyn+	2263
^{6}Li	d	1.0-3	1.0+0	ANL	T	Elwyn+	2265
			α thick target yield				
^{2}H	t	1.9-1		NBS	E	Fleming+	2626
			α spectrum average thick target yield				
^{cmp}Al	p	SCR		RIC	T	Walton+	273
			Z = 3 σ(E)				
^{12}C	^{12}C	4.5+1	2.0+2	TAM	E	Namboodiri+	4506

Target	Inc	Emin (MeV)	Emax (MeV)	Lab	W	Author	No.
			Z = 3 σ(E)				
^{45}Sc	^{22}Ne	8.4+1		GRE	M	Billerey+	5756
^{51}V	^{16}O	7.7+1		GRE	E	Chevarier+	5246
^{55}Mn	^{12}C	7.7+1		GRE	E	Chevarier+	4597
^{56}Fe	^{16}O	7.7+1		GRE	E	Chevarier+	5261
^{60}Ni	^{12}C	7.7+1		GRE	E	Chevarier+	4608
^{197}Au	^{12}C	1.3+2		YAL	E	Eyal+	4686 +
^{197}Au	^{16}O	1.4+2	1.7+2	YAL	E	Eyal+	5504 +
Many	^{12}C	NDG		GRE	M	Billerey+	4727
Many	^{14}N	NDG		GRE	M	Billerey+	4931
Many	^{16}O	NDG		GRE	M	Billerey+	5544
Many	^{22}Ne	NDG		GRE	M	Billerey+	5766
			Z = 3 partial σ(E)				
^{12}C	^{14}N	8.7+1	1.6+2	TAM	M	Stokstead+	4838
			Z = 3 spallation σ(E)				
^{12}C	p	4.5+1	1.0+2	MRY	E	Mathews+	119
			Z = 3 binary σ(E)				
^{16}O	p	2.0+1	1.0+2	INU	T	Compani-Tabrizi+	166
			Z = 3 spectrum average σ(E)xFactor				
^{cmp}O	p	0.0+0	9.9+9	MRY	M	Roche+	155
			^{6}Li product yield				
^{9}Be	p	5.0-2	2.0-1	KFI	E	Szenlpetery.	56
^{10}B	^{7}Li	2.4+1		HEI	E	Kohler+	4304
^{18}O	^{18}O	9.1+1		PAR	E	Detraz+	5592
			^{6}Li σ(E)				
^{4}He	α	6.0+1	1.4+2	MRY	E	Mathews.	3196
^{6}Li	d	2.3+0	6.0+0	TNL	E	Gould+	2266
^{12}C	^{14}N	8.7+1	1.6+2	TAM	M	Stokstead+	4795
			^{6}Li partial σ(E)				
^{4}He	α	4.7+1	5.0+1	MSU	E	King+	3197
^{6}Li	p	3.5+0	9.0+0	TNL	E	Gould+	34
^{9}Be	p	4.6+0	5.5+0	KTO	E	Yasue+	57
^{12}C	^{14}N	6.7+1		TAM	M	Stokstead+	4796
			^{6}Li compound-nucleus σ(E)				
^{12}C	^{14}N	8.7+1		TAM	M	Stokstead+	4797
^{12}C	^{14}N	8.7+1		TAM	M	Stokstead+	4798
^{12}C	^{14}N	8.7+1		TAM	M	Stokstead+	4799

FIGURE 6

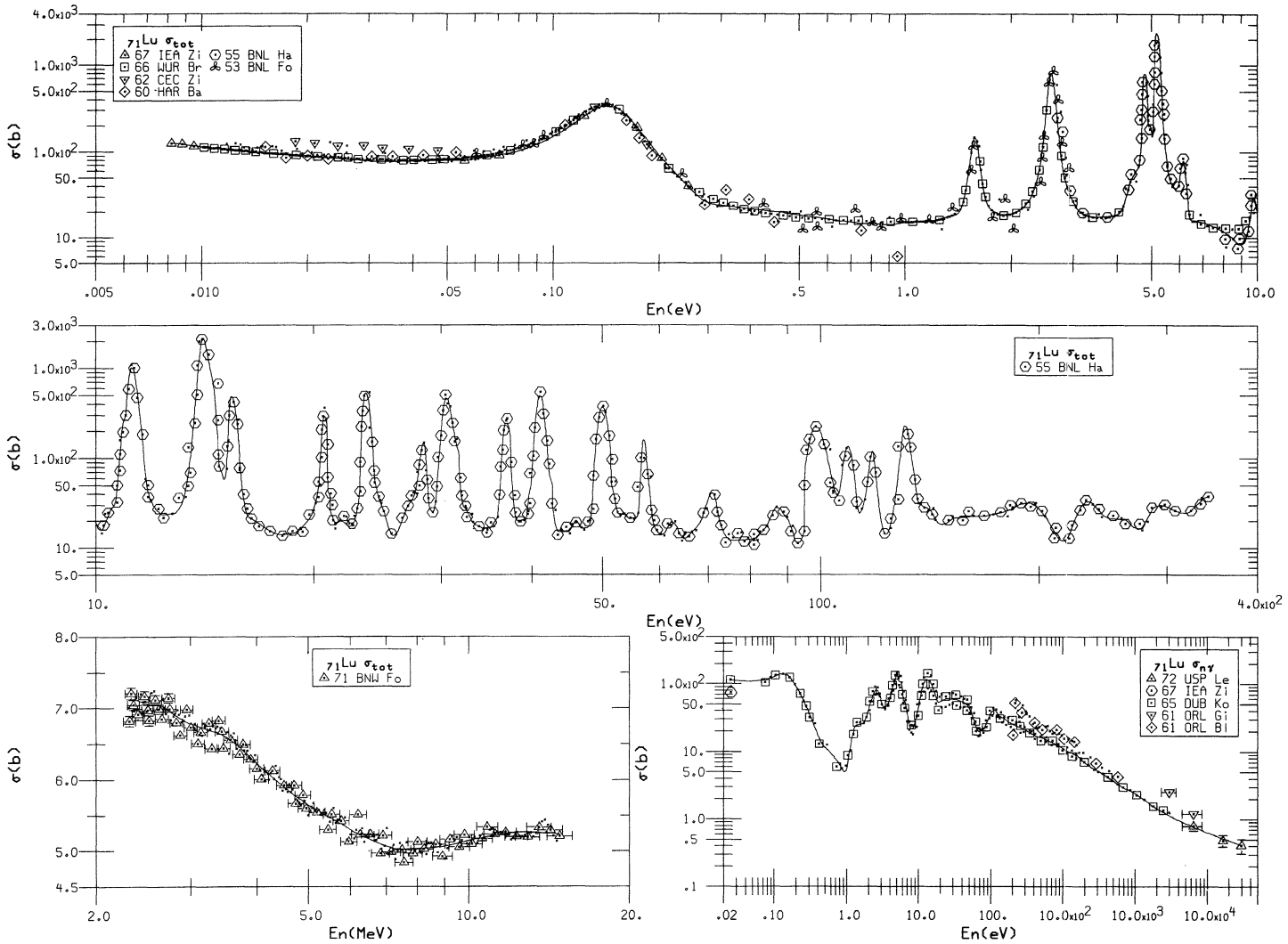


15N
σ_{tot}
σ_{el}

36

FIGURE 7

FIGURE 8



71Lu
σtot
σnγ

DESIGN CONSIDERATIONS

DEVICE SELECTION (3)

CHARACTER SET(S)
Determination
Definition

INFORMATION SERVICES - SCOPE

FEATURES -

USER-TYPES

END RESULT

FLEXIBILITY

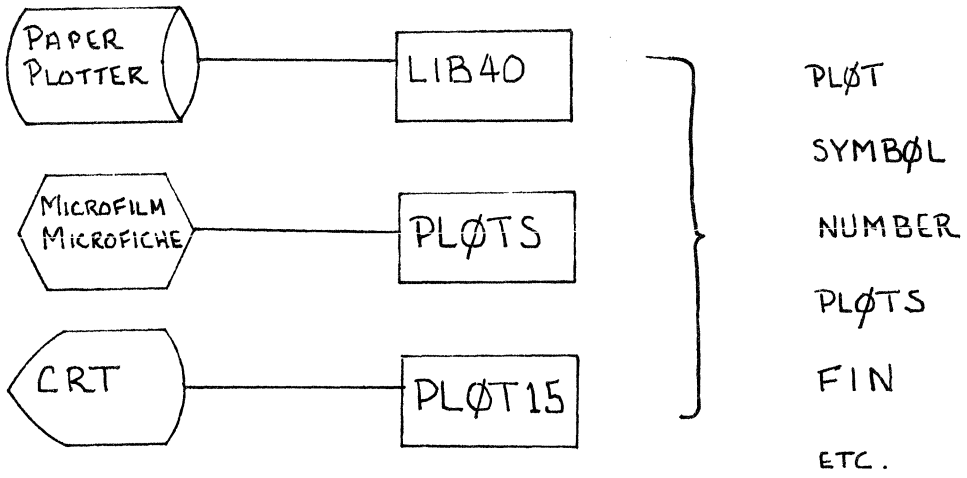
MAINTAINABLE

RELIABLE

EXPANDABLE

MINIMAL COST

FIGURE 9



(a)

PLØT3	PLØT5	PLØT8	PLØTD
PLØT	PLØTS	PLØT8	PLØTD
SYMBØL	SYMB5	SYMB8	SYMBD
NUMBER	NUMB5	NUMB8	NUMBD
PLØTS	PLØTS5	PLØTS8	PLØTSD
FIN	FIN5	FIN8	FINDD
CALCMP	CALC5	CALC8	CALCD
package	paper	film/fiche	Display - CRT

(b)

FIGURE 10

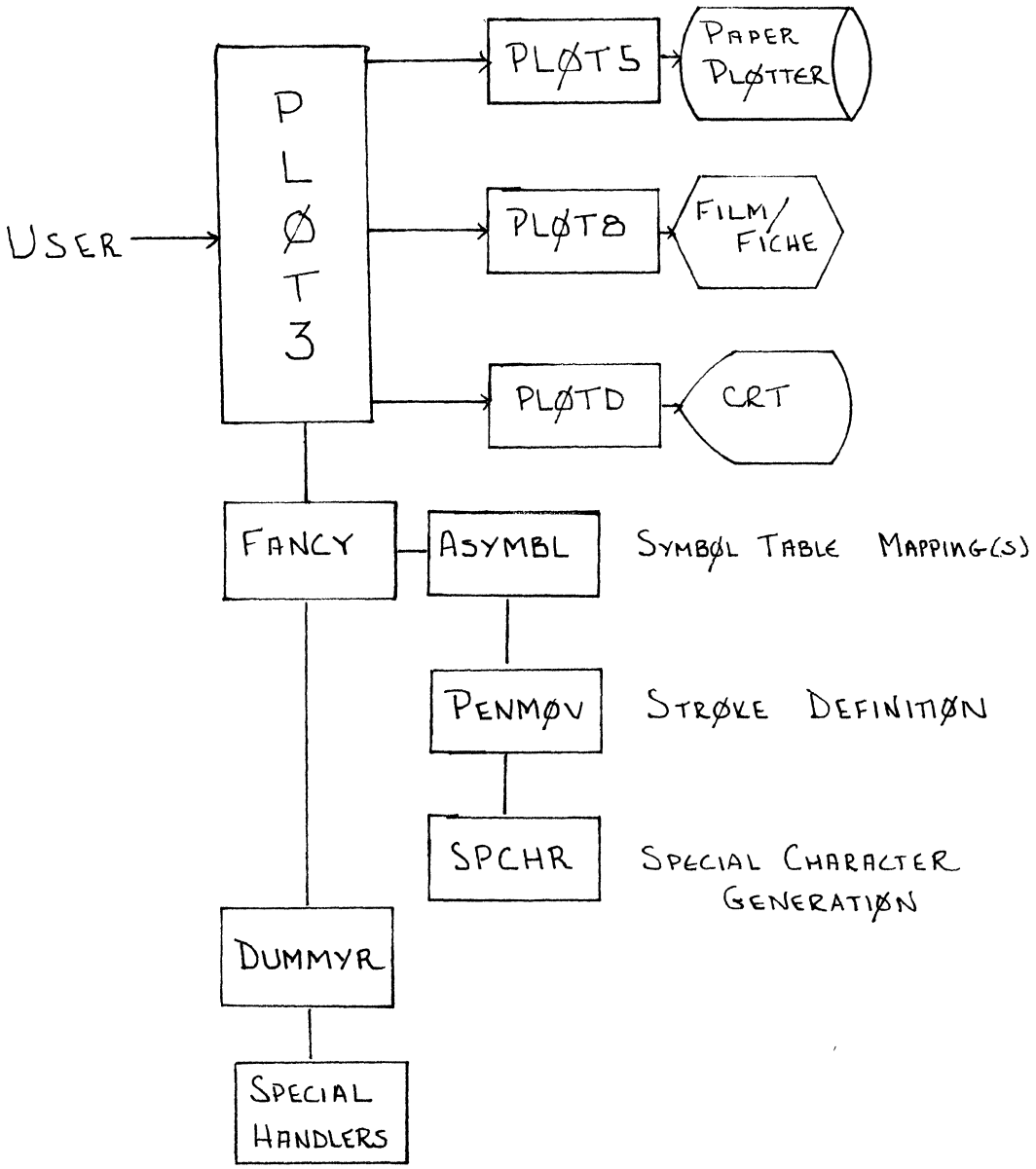

USER
PLØT3
PLØT5
PAPER PLØTTER
PLØT8
FILM/FICHE
PLØTD
CRT
FANCY
ASYMBL
SYMBØL TABLE MAPPING(S)
PENMØV
STRØKE DEFINITIØN
SPCHR
SPECIAL CHARACTER GENERATIØN
DUMMYR
SPECIAL HANDLERS

FIGURE 11

BNL325 Publication Sequence

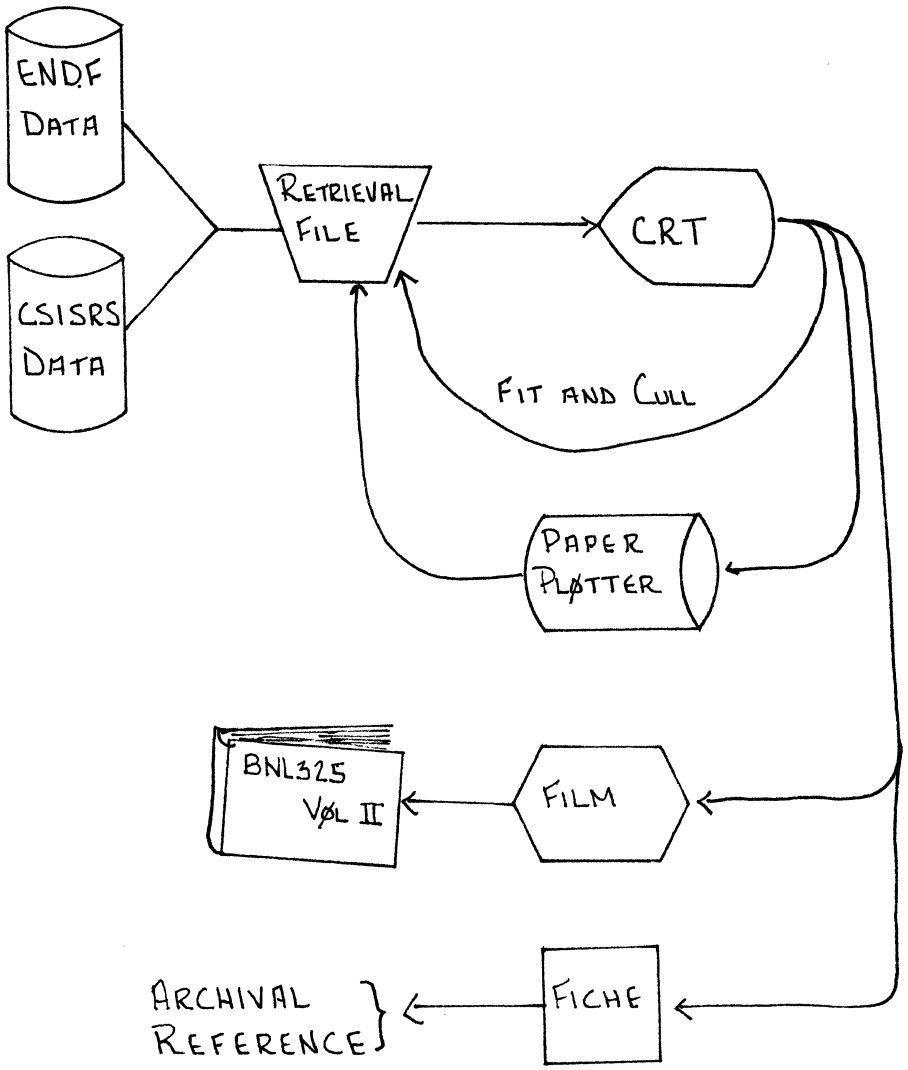


FIGURE 12

A SPECIAL PURPOSE LANGUAGE (STATUS) FOR TEACHING STATISTICS;

SOME OF ITS DESIGN PRINCIPLES, AND VALUES AS AN EDUCATIONAL TOOL

J.C. Turner
University of Waikato
Hamilton, New Zealand

ABSTRACT

STATUS is a language developed primarily to aid in the teaching of statistics and probability, and to point the way to new educational methods. However, it is broad enough to suit a wide class of users, and is open-ended. The main consideration in the design of its syntax is that statements shall be simple to learn and flexible in use. This simplicity, together with the power of its interpretive system, make STATUS a suitable base language from which to develop new forms of teaching statements.

KEYWORDS

Statistical computing; special purpose language; software; teaching; education; STATUS; values; methods.

1. EDUCATIONAL USES OF COMPUTERS

Much has been written on the use of computer languages, interpretive-systems, packages and on-line systems to aid in the teaching of statistics. A good set of references may be found in EVANS (1973).[1] A Conference of relevance to this paper is reported on by BAJPAI (1974).[2]

Whichever method writers experiment with they usually report enthusiastically on the results achieved. Some of the values and advantages mentioned are the following:

... Using the computer it is possible to bypass teaching fine points in grouping and other methods for calculating common indices ... especially true in attempts to introduce students to nonparametric distributions.

... Each student can spend most of his available time on interpreting results rather than calculating them ... not only able to interpret many results but also to do so on basis of information ordinarily too cumbersome for him to acquire.

... Each student can do more work with subject matter connected with his own speciality ... he is thereby much more strongly motivated to learn the statistical concepts.

... He becomes able to process data by a number of relatively complex computational procedures which leads to feelings of accomplishment (rather than frustration, or total failure, as when doing them by hand or desk calculator).

... He is introduced to many more techniques than is customary (many, if not most, undergraduates still learn only classical methods of one and two sample univariate tests, and some simple linear regression and correlation. Is this <u>really</u> enough, now that we have computers?)

... Data can be plotted in a variety of ways, and studies of distributions, and tests, can be made by graphical means.

... Simulation can be used as an educational tool.

... Familiarity with computing procedures and computers is gained, a valuable educational goal in itself.

2. DESIGN PRINCIPLES OF THE STATISTICAL LANGUAGE 'STATUS'

The language STATUS (STATistical computing Ultra Simple) has been developed at the University of Waikato, New Zealand with all the above teaching possibilities and values held firmly in view. The class of student for which it will be used, in the main, is undergraduate, both mathematical and non-mathematical, in all years. It has already been tested on large numbers of students in this class. The following claims - additional to those outlined above - can be made for it:

... Statements of the language can be used in statistical texts and notes in much the same way as mathematical script and formulae are now. Although not always as economical as mathematical writing, two or three lines of STATUS to define a formulae will often be more memorable and meaningful to a student than a similar mathematical development. This is certainly not the case with programs written in general purpose languages such as Fortran, Algol and APL.

... Each operation with STATUS is simple to learn and may be used with little or no change on both univariate and multivariate data. Thus a student on a second or third course will move easily to the complex calculations needed for multivariate work. Indeed, the intention is that with STATUS he will be introduced to multivariate statistics much earlier than is now possible.

... STATUS is based on matrix manipulation; so a statistics student will absorb, almost painlessly, the symbolism and operations of matrix algebra. He will be able to apply STATUS to the many other areas of mathematics, science and social science etc. where matrices play an important role.

... Although STATUS was designed with computer batch-processing in view, it can be used easily at a terminal, and on-line teaching methods are being devised for it.

3. REASONS FOR PRODUCING STATUS

The author's experiences with attempts to introduce computing into statistics courses (with classes both large and small, mathematics and methods orientated) have led him to the following convictions:

That a lecturer's time and energy are severely limited. Unless he has a regular support of teaching assistants and laboratory technicians, and immediate access to programming aid, he cannot attempt any computerized teaching system which will add much to his own load.

That using a general purpose language is not satisfactory for most courses (BASIC is perhaps the most acceptable). Statistical confusion is merely confounded by discussing computer routines. Consider for example, a student's deepening incomprehension when asked to study a Fortran program to calculate a statistic as simple even as $\bar{x}$; the use of subscripted variables and DO loops is more terrifying than the application of Σ in mathematical notation.

That interactive computer teaching systems are too expensive to be generally possible at the present time, particularly for large classes. An interactive system, one in which the student 'converses' with the computer, aims to usurp many of the traditional tasks of the teacher, freeing him for higher level work. It provides such advantages as instant answers, assessment of work and feedback, and directed learning tasks. But in many senses the computer is driving the student. The success with which it can do so - the self-confidence and independent abilities to probe and inquire which it can impart to the student - depend very much on the quality of the system software. The problem of producing software which is adequate to deal with general teaching situations in interactive mode is a very difficult one. However, it is undeniable that the benefits of using such systems are great, and their use will greatly increase in future, as computing power becomes cheaper and the necessary software becomes available.

That the best hope of putting large scale computers into the hands of large classes of students from a wide variety of disciplines is to produce high level, special purpose languages which can be learned and used very easily. Hence the development of STATUS. A matrix interpretive system was in use at the University, called SMIS (Symbolic Matrix Interpretive System); and it was considered better to build a teaching language onto that than to attempt to build teaching courses round statistical packages such as BASIS (Burroughs), or systems such as P-STAT.

For further discussion on all these points the reader is referred to WALLACE (1969).[3]

4. THE NEED FOR SIMPLICITY

Before going on to a description of the language STATUS, we make some general comments on one of the major themes of this paper - the need for simplicity. It is a matter which all too often appears to be insufficiently considered by computer scientists and software writers.

The emphasis with a special language must be on simplicity, otherwise the educational issues with which the student is engaged will be clouded by the difficulties of using the language. As Wallace says, there is a danger of converting statistics courses to programming courses (the introduction of too much mathematics into statistics courses is a remarkably similar danger). However if the statements of the language are devised with sufficient care, and symbolism kept to a minimum, program segments can be freely interspersed with teaching notes (in the same way that mathematical formulae are, and perhaps with more effect) and so will be assimilated with the subject matter.

The language must be flexible enough for equivalent results to be produced in a variety of ways. And the lecturer must be able to insert new statements into the system with comparative ease. He will then be able to set projects which are germaine to his course, and demanding and realistic for the students.

Apart from the simplicity of learning, writing and remembering a language, the ease by which it may be used at the Computer Centre must be considered. There appears to be a 'Computer Manager's Syndrome' which attacks many managers - it causes them to devise complicated input procedures and job control systems, and insists that users be coded to the nth degree. For a special purpose language to be of real use in teaching, this kind of tyranny must be swept away. With STATUS a student has no special control card to supply. He writes a 'start' statement, with his name and course number, and that is all; an 'end' statement is available but not essential. For student project work no more than this should ever be necessary.

If all language and operational procedures can be kept as simple as possible, both lecturer and students will confidently exploit the system, and achieve statistical objectives far more comprehensive than those of normal courses.

5. BRIEF DESCRIPTION OF THE STATUS LANGUAGE

Statements

Every statement in the language consists of an *opcode* followed by up to three *matrix names*, followed by up to four *numbers*. Additional information is sometimes inserted, as a bracketed '*argument*', after the opcode.

Example: FREQ DATA FRQ 12

This statement causes a grouped frequency table, with 12 group intervals, to be formed from the data stored in matrix DATA. The table is stored in FRQ.

An *opcode* is a 3- or 4-letter word, coding an operation which the computer has to carry out. It is usually evident from the opcode what the operation is to be. Examples are SUM, MEAN, FRQ, POWR, SAMP, LOAD, PRNT, ANOV, CORR. Familiarity with about twenty of these is quickly gained; this is sufficient for first-course students to carry out sophisticated statistical exercises, with real or simulated data.

The matrix *names* are made up by the program writer. They are used for locations in computer memory of input data, or results computations. No distinction between real and integer data is needed.

In general the *numbers* may be integers or decimals, used to specify such things as matrix orders, powers, numbers of frequency groups, etc.

Programs

The editing and punching of programs is enormously simplified by the fact that all is in free format. One or more spaces between items delimit the items. Further, statements are usually very short, and several may be punched on one card (separated by semi-colons).

Input/Output problems are reduced to a minimum. For student (and much general) work it is not necessary to give the user full control of his printed output. Provided that comments can be interspersed freely with printed matrices, there is no need of facilities for setting out tables, headings and text in precise positions on the printout. Much time is wasted on such niceties.

A student has no job control cards to wrestle with. His program must begin with a 'start' statement (e.g. *STRT J.C.TURNER*; even the name is optional) and that is all. No 'end' statement is needed.

Illustrative example: The following complete program illustrates the simplicity, and a few of the opcodes, of the language.

Program	Explanation
STRT J.C.TURNER DEMO.	Essential start card; text after STRT is optional.
LOAD TEMP 30 5	Instructs computer to read data (e.g. temperatures) into a matrix named TEMP of order 30x5
21.3 14.6 18.7	Data for TEMP, stored row-
23.4 22.0 17.9	wise. It is punched in
28.1 25.3 ..... 24.5	free format, using as many cards as necessary
* COMPUTE FIVE MOMENTS	An asterisk signifies a 'comment' which will appear in the program printout
STAT TEMP MOMS	The statistics ($\bar{x}$, s, m_2, m_3, m_4) will be stored as a column-vector in MOMS (name supplied by user)
PRNT MOMS	The statistical vector will be printed out, with heading MATRIX MOMS. Elements are printed with 5 decimal digits (format F14.5); rows and columns are numbered for ease of reference
FREQ TEMP FRQ 12	A grouped frequency table, with 12 groups, is computed and stored under name FRQ. If the opcode is extended thus: FREQ (EXPLAIN), a neat tabular printout be given automatically
HIST FRQ	A suitably scaled histogram of the TEMP data will be printed out.

Program	Explanation
* SUPPOSE NOW THAT THE * NUMBERS IN TEMP ARE * THIRTY * RECORDINGS OF A * 5-VARIATE * EXPERIMENT	Comments
CORR TEMP R PRNT R 1 CORRELATION MATRIX	The 5x5 correlation matrix R will be computed and printed out. The 1 in the print statement signifies that a 'heading' follows, to be printed immediately before the matrix R
EIGN R VALS VECS PRNT VALS, VECS	The eigenvalues and vectors will be computed and stored in VALS, VECS; they are then printed out
* SUPPOSE NOW THAT THE * DATA * IN TEMP IS FROM AN * UNREPLICATED * 2-FACTOR EXPT * REQUIRING ANALYSIS * OF VARIANCE	Comments
ANOV TEMP 30 5	The analysis of variance computations will be carried out. A standard anova table will be printed. Sums of squares, degrees of freedom and mean squares will be stored under fixed names SS, DF, MS respectively, for further calculations if desired.

Using semi-colons the whole program can be punched on about 7 cards (excluding data cards). The program causes a great deal of computing to be carried out, and the manner of writing it is simple and easily remembered. In teaching, of course, such a hotch-potch program would never be presented. Opcodes and statements are available for illustrating statistical concepts and methods in whichever order, at whatever depth, the lecturer cares to introduce them. From their very first lesson, and in each subsequent lesson, the students learn to write complete programs, of increasing sophistication. A simple example of this process follows.

6. EXAMPLE OF TEACHING PROCEDURE

Consider some of the difficulties in teaching the concept and computation of the arithmetic mean. A lecturer is faced with these in his first or second lecture.

Teaching Problems

Given a sample of 20 observations from an experiment we have to explain the symbols, mean, computation and uses of

$$\bar{x} = \left(\sum_{i=1}^{20} x_i\right)/20.$$

Time has to be spent explaining subscripts and the sigma operator, both of which are quite difficult concepts for most beginners. And, as mentioned above, to show how a computer obtains $\bar{x}$, using FORTRAN with a subscripted variable and DO loop, would deepen the confusion.

STATUS approach

Introducing the sample as a column-vector X, the STATUS algorithm (program) for producing $\bar{x}$ in XBAR is

```
SIGM  X     SUMX        ... stores Σx in SUMX
DIV   SUMX  20   XBAR    ... divides by 20 and stores
                             x̄ in XBAR
```

This can be given first, before introducing $\Sigma x/n$, its mathematical equivalent. It may also be worthwhile to introduce the alternative algorithm

```
MULT  IR    X    SUMX
DIV   SUMX  20   XBAR
```

where IR is a 1x20 row-vector of 1's. This gives another view of $\bar{x}$; and prepares the way for use of $X'X$ as a sum of squares - and for $X'X$ later as the sum of squares and cross products matrix when X is an nxp matrix.

If the means of 12 (say) samples are required, then X is written as a 20x12 matrix, each column being one sample. Both the above algorithms, operating on X, will produce in XBAR the row-vector $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{12})$. Thus the student can be set a project in lesson 1 to study, and report on, sampling properties of the mean. His writing of computer programs will be quick but not automatic; each statement has to be thought about from a statistics point of view. And when he receives his printout he will have further calculations to do by hand (e.g. find ranges), and conclusions to draw and write about.

Once an algorithm in STATUS is learned, a single statement may be available to replace it. Thus for the arithmetic mean the statement MEAN X XBAR does the job, producing a vector of column means from the matrix X. This enables teaching to proceed in small steps or large ones, depending on the standards and objectives of the course. Note that even a small step in STATUS is a big one compared with its equivalent in FORTRAN. And whenever possible it will be one with direct statistical significance.

Mathematics formulae must be introduced too, but often this will best be done after algorithms have been used by the students. (*KNUTH*, the well-known computer scientist, has stated his belief that a mathematician has not fully understood a mathematics theorem if he cannot state it in algorithmic form. There is much truth in this.)

7. SPECIAL FACILITIES

A number of facilities which add greatly to the power of the language will now be described briefly.

Use of Disk Storage

A lecturer must be able to store sets of data on disk, and students to use these sets easily in their projects. STATUS provides these facilities thus:

Storing Statement

```
FILE  MATX
```

MATX: any matrix name, of a set of data loaded by the lecturer in the usual manner

FILE: opcode for filing MATX on disk

Accessing Statement

```
DISK  MATX
```

MATX: any matrix mane, of a set of data already stored on disk

DISK: opcode for causing MATX to be copied from disk to core memory

Data matrices are just as easily removed from disk, using the opcode PURG (for 'purging'). To prevent unlawful or careless purging, the device of writing FILE(LOCK) MATX will 'lock' the matrix on disk; an unlocking device is of course available - but it is not given to students!

Simulated Sampling

A wide range of statistical functions may be sampled by means of the standard statement:

```
SAMP (distribution type)  M  m  n   a  b  ...
```

SAMP: the name of the distribution function must be written here.

M m n: an mxn matrix is supplied

a b ...: values of distribution parameters as needed

No 'seed' need be specified for the generation of pseudo-random digits, although the one currently in use can be inspected and changed if desired by means of an opcode SEED.

Distribution types available are UNIFORM, EXPONENTIAL, NORMAL, BINOMIAL, POISSON, GEOMETRIC, CHISQ, F and T. Others can be added as teaching needs dictate. The matrix M of sampled values is immediately available for use in studies of probability distributions, sampling theories and simulations.

Matrix Handling

There are many facilities of the language which make it a very powerful one for matrix handling. Some of these are:

Unconditional branching (using GO and DEC)
Looping (several methods; the simplest uses LPTO)
Conditional branching (using IF, with relations .NE.,.EQ.,etc.)
Creating matrices (creating, spanning, removing inserting, etc. of special matrices, sub-matrices, diagonals)
Permuting on columns

Of special mention is that *vectors* of matrices may be stored and accessed simply. An opcode may be set to operate on just one of the matrices in the vector, or on all of them.

8. SCOPE OF STATUS

The current state of the language is that it contains over 100 subroutines; about 40 opcodes carry out matrix operations and 40 are specific to statistical work. In a first course in statistics, knowledge of about 25 opcodes is very rapidly gained, and these are sufficient for many teaching purposes (and general use). Of the many students tested, with or without previous computing knowledge, whether symbol-oriented or not, very few had difficulty in using it freely from the first lesson. Eleven teaching units have been written, to be issued in three sets. The titles are:

Set 1: *Matrix handling, Basic statistics, Sampling; probability distributions, Statistical inference;*
Set 2: *Various input and output facilities, Multivariate analysis (1), Branching and looping; creating matrices, Analysis of variance;*
Set 3: *Multivariate analysis (2), Multivariate analysis (3), Multivariate analysis (4).*

A twelfth unit deals with SMINT and PUTIN, the procedures for inserting new opcodes into the language. A set of opcodes for Operational Research projects is also included in the language. The opcodes are LINP (linear programming), QSIM and QCAL (simulation and calculations on queue systems), and INV (inventory systems).

9. THE FUTURE

A number of statements for time-series analysis, and for a variety of non-parametric methods will be introduced shortly.

Three proposed additions to the facilities of STATUS are:

(1) the ability to compute a 'higher arithmetic expression' - that is one composed of matrices and matrix operations - with a single statement;

(2) the ability to store subroutines temporarily:

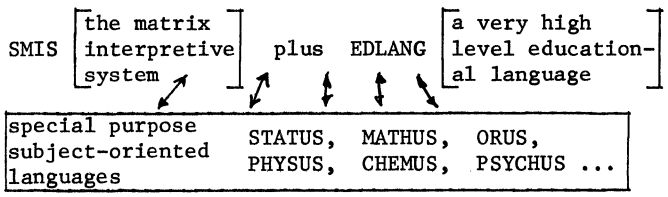
(3) to simplify student use, mark-sense and porta-punch cards are being designed for use in a PDP 11/34 Cafeteria system.

Teaching experiments are being planned for 1979 onwards which will enable different approaches to statistics teaching to be taken, for courses both mathematics and methods oriented, and their results to be compared. Using STATUS and adding to it when necessary, attempts will be made to travel further and achieve a wider range of objectives than normally. Clearly, if more project work is to be expected of a student - however simply it can be programmed for computer - then less of his time can be directed to examination of purely mathematical aspects; this is where cuts will no doubt have to be made. However, if the project work leads to better grasp of statistical principles, more application, more analysis and more evaluation of results, this must be to the good for the majority of our students.

If one is convinced that a special purpose statistical language such as STATUS is going to provide benefits of real educational value, one wonders whether an even higher level language (a super language!) should not be developed for educational purposes. When planning to use a computer to help with a process, it is easy to fall into the trap of merely programming the process as it exists now, rather than considering the equation

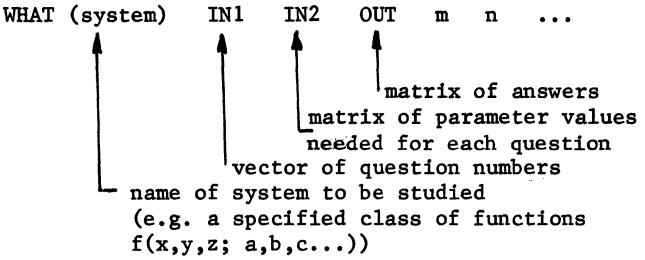
'computer plus process' = 'new process' ;

one should ask what kinds of 'new process' are possible, before beginning programming. For example, the following sketch of a super educational language might be what we should aim at.



The subject languages would operate independently with SMIS (for local teaching and general purposes), or through EDLANG if certain high level teaching processes were being attempted. EDLANG would enable computations and analysis of data (literal and numeric) to be carried out which answered general questions covering a spectrum of subjects. For example, instead of asking a student to study and use one statistics function at a time, could we not provide him (via the computer) with a schema of information which enabled him to understand families of such functions, used in different contexts. Could he be enabled to study systems of equations (linear, nonlinear, differential), and model and examine processes (discrete and continuous), evaluating effects of changes on them; to ask questions about them and persuade the computer to answer them; and to synthesise and report on his findings. If this could be made possible, then univariate studies would give way quickly to multivariate ones, linear to nonlinear, functions to function spaces, and so on.

The type of statement which might exist in EDLANG (pure speculation at the moment) is



The available questions a student might ask would be of the type "What happens if ...?" These would have to be carefully specified in relation to each system. Other opcodes, such as LOOK, HOW etc., would allow for plots to be asked for, and other kinds of question to be made.

ACKNOWLEDGEMENTS, ETC.

A matrix-handling system called SMIS was originally produced at the University of California, Berkeley (1967). The simplicity of its use gave the author the incentive to devise the STATUS language. Full credit must be given to W.T. Rogers, University of Waikato, who has been responsible for large scale changes, simplifications and additions to the interpretive system, and has helped the author considerably in the writing of the statistical subroutines.

STATUS is currently running on PDP 11/70(RSTS) and Burroughs B6700 computers. It is written entirely in FORTRAN, and requires little work to modify it for other machines. For further details of computing aspects see ROGERS AND TURNER (197?)[4], or write for information to J.C. Turner, University of Waikato, Hamilton, New Zealand. Prospective users outside New Zealand please note that an initial charge of $750 will be made for a tape of STATUS, and a further $35 per year if the newsletter and update service is required.

REFERENCES

[2] BAJPAI, A.C. (1974) (Editor). Proceedings of the Conference held under the auspices of NCC/CAMET, Mar 1974. Int.J.of Math.Ed.in Sc. & Tech. 5, Nos. 3 and 4.

[1] EVANS, D.A. (1973). The influence of Computers in the Teaching of Statistics. J.R.Statist. Soc.A, 136, pt 2, 153-190.

[4] ROGERS, W.T. and TURNER, J.C. (197?). STATUS: a Statistical Computing Language. (in preparation for the Journal of Statistical Computing and Simulation)

[3] WALLACE, D.L. (1969). Computers in the teaching of statistics: where are the main effects? In Statistical Computation (Milton, R.C. & Nedler, J.A. editors), pp 349-361. New York: Academic Press.

ENGLISH STRANDS

Eric Leventhal
Riverdale Country School
Riverdale, New York

ABSTRACT

English Strands are on their way to becoming a reality. This paper describes the efforts to date that have lead to a working segment of what will eventually be a complete English drill and practice package.

Educational software has been developed at Riverdale Country School since 1973. Our constant goal has been to ease the burden of the classroom teacher by having the computer provide some individualized drill work. This paper will present a brief history of our activities and in some detail describe our work done to date on English Strands.

Ideally any package should be simple to use; great efforts must be made so that both the students and the teachers need know as little as possible about the computer. The purpose of computer assisted instruction (CAI) is to help the teacher in instructing the student. The teacher must be able to enroll students in the drill programs and to produce comprehensive reports about individual students or entire classes, without the aid of a computer expert. The software must be easy to use so that the teacher is not discouraged from using the computer. The software must also be easy to modify so that feedback from the teacher can be used to make improvements in the programs and in the curriculum.

The curriculum of any CAI package is crucial to its success. Through years of experience we at Riverdale have learned that no matter how well written the individual programs are, without a complete curriculum they are useless. From our experience, in the Math Strands, the format we have found best for drilling is the division of each curriculum into different topics or "strands". Each strand is then divided into a series of levels which may correspond to increasing difficulty or simply to different rules. The students may then be drilled on a series of exercises from various strands, or in exercises from an individual strand.

The development of drilling programs for Mathematics was begun five years ago by Riverdale personnel in conjunction with a school from which we were renting computer time. The original curriculum for the Math Strands was based on the work of Dr. Patrick Suppes at Stanford University. The first version of the Math Strands was written by Michael Fulop, a tenth grade student, in the summer of 1974. A year later work commenced on English Strands with a vocabulary program, but because of the massive amounts of data required it has never been used extensively. Each vocabulary word must have a definition, an example, a hint, and seven drills; although the learning strategies seem effective, it will take a long time to produce, enter, and edit all of the curriculum material necessary for large scale use.

Although the vocabulary program is not currently used, a program which was developed for use with VOCAB and other English programs, called ENGMOD, a general purpose data enterer, editor and printer, is still used whenever variable length data storage is required in our CAI programs.

During 1976 and 1977 the Math Strands were revised and upgraded as feedback from teachers and students pointed out inadequacies and possible improvements. (For further information on the Math Strands please refer to the presentation by David Solomon at the San Diego DECUS Conference in 1977.) The English Strands on the other hand, has remained unused. The problem has not been lack of programmers, because very enthusiastic and capable student programmers are always available, but has been finding willing and qualified individuals to develop the curriculum.

An effort was begun in the spring of 1978 to design and develop a set of English CAI programs. The success of Math Strands has convinced us that the strands concept is sound and should be adhered to. From past experience it was decided to keep the actual volume of data at a manageable size and to find some logical way to divide the curriculum so that it need not all be developed at one time. The lesson we learned from previous English efforts was that the teachers who would be using the final CAI programs must be involved in designing and and implementing the curriculum. The different needs of teachers at different grade levels suggested that we develop two different approaches to the use of the computer in English. One approach, aimed

at the elementary grades, stresses the general language arts curriculum, while the other approach, for the high school is oriented toward specific topics.

A paragraph format has been devised for the curriculum at the elementary level. Starr Snead, a fifth grade teacher and a specialist in reading and learning disabilities, and Phil Snead, a school psychologist and former fifth and sixth grade teacher, have developed this curriculum over the summer; Phil Terman, an eleventh grade student, wrote the program. These paragraphs incorporate a highly flexible battery of exercises in a series of one hundred fifty paragraphs. The curriculum is developed from the analysis of several elementary school texts. The added complexity and difficulty in writing the material is more than compensated for by the higher level of interest and the stronger motivation for the young children to use the program, because of the coherent paragraph form.

Each of the one hundred fifty paragraphs consists of four or five sentences, each sentence dealing with a particular strand: (1) Vocabulary, (2) Spelling, (3) Punctuation, (4) Usage. Each sentence has a simple, average and difficult form of the question. A combination of these sentences is chosen according to the student's recorded proficiency in the particular strand. From these sentences a cohesive paragraph is composed. This paragraph is printed, and multiple choice questions are asked about each strand. The student's record is updated based on performance during a session. It is also possible, if a student is weak in one strand, to drill the student in only that one strand.

David Nicholson, a high school English teacher, has worked in conjunction with Eric Leventhal, a twenfth grade student, to implement English Strands. These strands concentrate on specific skills; therefore, no paragraphs are used. Instead, the emphasis is placed on college-board type drills. The high school curriculum is organized into six strands. These are: (1) Punctuation, (2) Syntax, (3) Grammar and Usage, (4) Diction and Style, (5) Spelling, and (6) Vocabulary. Because of our past experiences with developing English curriculum, only the punctuation and syntax strands have been tackled thus far. Each strand is divided into a series of substrands. In the case of punctuation some of the substrands include quotation marks, hyphens and dashes, and parenthesis. Each substrand is composed of one or more levels; each with one rule and a set of drills. Three types of approaches are available: the student may answer assorted drills from various strands at the level which the student is currently on in the strand; or the student might answer questions from only one strand in which case the student will advance from one level to the next or the student may do drills from one specific rule in a particular strand. This last method is used to exercise skills in which the teacher has determined the student needs extra work.

This logical structure is implemented through two sets of variable length ASCII data files. The ENGMOD file structure and routines are used; this has greatly simplified the debugging of the program. One file, the index file, is a two dimensional array: strand number by level. The current English programs allow for eight strands and any number of levels. The index contains a list of pointers to the other file, the drill text file, as well as the criteria for promotion from each level to the next one. The individual level consists of a pointer to the text of its rule, its standard for advancement, the number of the level to advance to, and a set of up to four instructions each with a list of pointers to the drills. The first line of the entries in the data file is an I.D. line. This identifies the entry in terms of its type (rule, instruction or drill), the strand, the substrand, and the level it belongs to. This structure allows a great deal of flexibility in the arrangement of the levels and allows cross checking in order to verify the integrity of the data base.

Each entry in the data file is comprised of five sections: the I.D. line, the main text, the multiple choice question, the conditional print, and the answers to the multiple choice question. Any section, except the I.D. line may be omitted. The I.D. line is as described above. The main text may either contain pure text, as in the case of rules and instructions, or may have embedded questions. The embedded questions format is mainly used in the punctuation strand where the text is printed and the terminal waits for the student to type in a punctuation mark or answer. If the student's answer is correct then the next part of the sentence is printed; if the answer is wrong then the student is asked to try again. (The student is given two tries.) The multiple choice question and answers may be used alone or in conjunction with the main text to strengthen the student's understanding of a rule. The conditional print lines are optionally displayed according to the student's answer to the multiple choice question. These lines might be a hint, encouragement, or the corrected version of a sentence.

While writing the two different English Strands this summer and autumn, we have tried to keep the two sets of programs as much in common as possible. The very different drill formats have made it necessary to separate the data files, but both sets of files are in ENGMOD format. Because ENGMOD was written and debugged before this summer neither of the programmers had to spend time writing new routines or additional data utility programs. Although the reporting requirements for the fifth and tenth grade English Strands are

different, there are sufficient similarities to warrant combining them into one file. The student enrolling program for Math Strands has been generalized and now will enroll students in English Strands as well.

The English Strands actually consist of nine programs; some are shared with other CAI programs. The programs include: ENGMOD and VTMOD, the variable length data enter, edit, and list programs; ENROLL, the student enroll and delete program; STRAND, the program which is used for students to begin a CAI session; ENGREP, which generates reports on the students' progress in English; ENG5, the fifth grade drill program; ENGDRL, the tenth grade drill program; and two utility programs for the tenth grade strands data files. The two utility programs are ENGCHK, which verifies the integrity of the index and data files, and ENGTBL, which lists information about the rules and drills available. ENGCHK checks every level of one strand and compares the I.D. line of the rule to that of each instruction and drill which are also used by the level. All differences are listed along with the level number, rule number, and instruction or drill number. This cross check, although not perfect, has been quite effective at locating typing errors in the index file or in the entering of data items.

ENGTBL lists the levels of a strand so that a teacher can see what level corresponds to the current classroom work, or is able to tell a student what to type in order to get special help on some rule the student may be having difficulty with. ENGTBL lists the level number and an expanded version of the I.D. line (e.g. "Rule for Punctuation and Commas A and B" instead of "R2;A&B,1"), ENGCHK lists the text of the rule and/or lists the basis for advancement to the next level, what the next level is, and a list of the instructions and the drills for the rule according to the wishes of the teacher. The text of the instructions and drills may be listed using ENGMOD, but these are only needed to correct errors.

Manuals are being prepared for all Strand programs. The source programs have been carefully documented, line by line, and include all subroutines, functions, and variables. Every data file has a format (.FRM) file describing it. This practice is necessary if the programs are ever to be updated or patched after they have been written. Even though it is time consuming to keep the documentation up to date, the benefits are well worth the time spent. In a school environment, the improved ease of a new programmer understanding a program is greatly appreciated because each student programmer will, at most, only be at school for a few years.

Students can use English without having to know much about the computer, all a student must do is to type "ENGLISH". If the student is logged off then LOGIN will chain to STRAND on the system library account. If the student is logged on then the CCL command "ENGLISH" would run STRAND. In addition to the general session, the student may request work on a specific strand (e.g. "English Punctuation" where "Punctuation" may be abbreviated to any length; as long as it cannot be confused with another strand) or on a specific strand and level (e.g. "English Punctuation 7"). STRAND then asks the student his I.D. number and name. These are checked to be sure they are valid and match. If they are, STRAND then chains to either elementary school or high school English Strands, depending on which one the student is enrolled in. The session then begins at a level based on the results of the previous sessions; however, if the student specifies a level, the session uses that one instead. Either way, the score is recorded at the end of the session and a score summary is printed on the terminal for the student. The summary includes a list of all the strands and levels worked on in a particular session, the number of drills done for each, and the number of drills done correctly.

The teacher must log on to the system in order to work with English, but this can be done from any account. The teacher can run: ENROLL, to enroll new students or delete students; ENGREP, to receive a report on the progress of an individual student, or an entire class; ENGTBL, to list the rules, texts and drills used; and ENGMOD to enter or edit the rules, the instructions, the drills or the index. For security reasons, each of these programs have a password.

In the future, as experience is gained with the English Strands, modifications will be made. The main foundation for any change will not be to increase efficiency or to decrease storage space, but to best meet the requirements of the students and the teachers. The curriculum will continue to expand as the other strands curriculum are written and entered. Teachers, as they begin to see the possibilities, are becoming enthusiastic. The success of the English Strands depends upon the ease with which it can be integrated into the classroom and used as an adjunct to the more traditional learning procedures. At Riverdale we feel that English Strands will be able to live up to these difficult requirements.

A PROPOSAL ON THE FUTURE DIRECTION OF COMPUTER ASSISTED INSTRUCTION

Bruce G. Alcock
Riverdale Country School
Riverdale, New York

ABSTRACT

There is a critical need for establishing guidelines that will foster development and implementation of computer assisted instruction on a wide scale. Presented below are three points which are intended to greatly expand the scope of development projects as well as increase the number of potential users.

For the past few years, the power of the computer in the educational environment as a learning tool has not only become evident but it has also been applied in practical every day use. This has resulted in a belief among computer education specialists that in order to expand activities beyond isolated local pockets of success, a national effort is necessary. This is a proposal for one possible approach.

This summer the House of Representatives Committee on Science and Technology held hearings on computer uses in education and what could be done to further the all too few efforts currently being made. They are expected to recommend the creation of a presidential appointed committee to study the problem further. This past August an organization called the National Educational Computing Consortium was formed to: (1) Influence direction of educational computing. (2) Conduct special studies and projects. (3) Provide information to various societies. (4) Hold meetings. There has also been talk about the creation of a national clearing house through which all CAI development would be coordinated.

For the past six years, personnel at the Riverdale Country School have been working on developing and implementing curriculum on the computer. In some cases this has involved modification of existing curricula but there have also been attempts at developing the curriculum itself. Some projects have failed, or are still in progress, but others have been extremely successful. Based on this we now feel that there is a lack of experience in developing material for the elementary/secondary school market that is evident in all proposals we have seen to date. Furthermore, our feelings dealing with more general issues are reinforced on the college level through our involvement with DECUS (Digital Equipment Computer Users Society) which has brought us in contact with schools of higher education that are producing and using successful material.

The first fact that becomes painfully clear is the void which exists between various institutions that are developing material. This void could also be described as a lack of communication, or more appropriately, a lack of channels through which communication is fostered. For the past five years, only TICCIT and PLATO have received any real publicity effort and neither of these projects is useful to the user who has a computer system and would like to implement some type of computer assisted instruction. The only lines of communication that do exist are through user groups and these tend to be very insular. Even so, this provides some sort of awareness that there are other installations attempting similar projects.

The elementary/secondary school environment is especially affected by the lack of knowledge on availability of material. Computer personnel at this level often are math or science teachers doing computer work on a part-time basis. They do not belong to any of the computer-oriented professional societies, and do not know about user groups or other schools using computers that could provide information on using the computer and teaching programming.

A central clearing house to distribute software is necessary. While the activities of CONDUIT are already engaged in this activity, possible expansion of this organization has to be seriously considered. More importantly, the existing activities and future potential need to be publicized not just among the computer education people but among all educators as a source of existing low cost material. The programs distributed by CONDUIT theoretically may be run on any system, and may even be adaptable to the microcomputer market.

The computer manufacturers user groups can provide another good source for interchange of material, but they often are not geared to the needs of educators. Some form of incentive must be developed including, if need be, the subsidizing of education oriented user groups.

The most difficult aspect of developing CAI material is the curriculum itself. Lesson materials must be presented in a format which lends itself to computer implementation. Teachers currently derive their curriculum content through the use of textbooks which for the most part are linear with little or no branching. The student reads a chapter and answers questions at the end of the chapter. This work is corrected in class and the next day the teacher prepares a student for the next homework assignment by introducing the next section. While this is an oversimplification, it is obviously impossible for a teacher with twenty students to hand out twenty different homework assignments, especially if that teacher has four or five different classes. If the student is weak in one area, the problems should be easier in that area and harder for those skills in which the student has strength.

A computer curriculum is designed with a single student in mind, since the student and the terminal are on a one-to-one basis. Furthermore, the branching capabilities that make possible easier or harder exercises, additional explanations, or skipping material the student is familiar with, are concepts familiar to the systems analyst, but totally alien to the curriculum specialist.

Projects, including workshops and articles in general educational journals to educate the educator on how to write new course material using the above mentioned features, should be sponsored by the National Science Foundation as well as National Institute for Education. Bringing the computer into the mainstream of education is not an easy task, but is essential to extending the use of the computer. Currently, only the computer education specialist is able to use the new technology and create material for it. Funding which will bring knowledge on how to use the computer to the majority of educators still unaware of how to use the potential will alleviate fears about computers as well as make it possible to have an increase in development of curriculum for the computer.

Exchange of material has been an issue which the commercial market has solved through the use of COBOL, and the scientific community through FORTRAN. BASIC is supposed to be the language of education, but is rapidly spreading to small systems for business and other applications. The minimal BASIC (for which a standard now exists) does not have enough power to produce good software - it does not even include text manipulation facilities. The proliferation of education (CAI) languages includes TUTOR, COURSEWRITER, PILOT, etc., all of which have strong supporters. This complicates the picture since it is impossible to exchange software. Perhaps the attempts at standardization and interchange. are being made at the wrong level. It is difficult to present the exact same lesson on a graphics CRT terminal and on a paper terminal of 72 characters width. Graphics cannot be used on the printer and text must remain on the screen long enough for the student to read and understand it.

The common denominator is the curriculum. This is the most difficult and expensive portion of CAI development. Implementation left up to individual institutions can take many forms including graphics when the equipment is available. The implementation language, as well as method, should be such that the system is used most effectively. Optimization is especially important in a market where hardware dollars are limited, and programming talent in the form of students is relatively inexpensive. Computer people not familiar with this special environment would tend to have education follow the patterns set by business where equipment expense can be justified by reducing operating costs, as well as having a tax advantage, while trying to keep expensive programming costs down.

Developing curriculum material is too expensive to be undertaken as an in-house project with no external funding. Moreover, a great deal of good talent lies fallow because of this cost factor. The federal government through NSF and NIE must be persuaded to initiate a program that will make it possible for qualified institutions to form the lesson material that is so desperately needed. But in doing so, some organization must keep track of progress of all the projects for two reasons. The first reason, accountability, means that the work will be completed according to the terms under which the grant was made (not to control the effort); the second, publicity, will encourage developers since each project has to be given national coverage. Motivation will be increased and sharing of ideas will also increase. Coverage, not restricted to the computer education people, but among all educators will make them aware of the effort that is taking place and what material will be available in the future. They can then plan to use the computer in the not too distant future.

Terms of the grant must include making the final product available to all other educational institutions at distribution cost. It would be more appropriate, however, for an expanded CONDUIT or a national clearing house to take on this task of distribution. Individual institutions would then be able to implement the existing curriculum packages quite rapidly on an array of different processors. Computer manufacturers then would find it profitable to buy the best CAI packages, provide support and sell them to schools that do not want to develop their own software.

While there are no guarantees that the software part of this plan would be a success, consider that a private school (elementary/secondary level) took one summer to implement a version of the Stanford Math Strands curriculum. Certainly, at this point, private enterprises would find it profitable to write the programs and this must not be discouraged. With the curriculum in the public domain, anyone could pick it up and do whatever they wanted to with it.

From the lessons of TICCIT and PLATO we should learn that any further money must be spent on the software and not hardware, and the software funds should go toward content not more languages. Any money spent for system software should be directed toward networking of microprocessors. Distributed processing does not mean isolated processing, especially in the educational environment where central records on student performance permit mobility on the part of students.

Holding national meetings is not a bad idea but if it only caters to a small group of specialists it is not going to help expand CAI development. A national conference that will specifically address itself to involving the non-computer educators is necessary. But perhaps, the first effort should be made by computer educators to attend the regular educational meetings.

The main points that must be stressed are:

1. Funding of a distributor - CONDUIT - is the easiest way since they already have the expertise. But whoever takes on the task of distribution must include the elementary/secondary school market, as well as higher education, and be able to distribute curriculum packages.

2. Develop new methods for curriculum presentation, and inform the entire education industry of these methods. Do not restrict creating CAI to a small group of specialists. The programming and curriculum work are two separate areas that should be attended to by two separate groups, without losing communication between the two parties.

3. Funding of local curriculum development. Emphasis has to be placed on complete packages, and keeping the entire education industry informed on what is being done. Curriculum packages will fall into the public domain, but specific implementations can be sold commercially. The profit incentive has been lacking thus far, and this will adequately reduce initial investment on CAI.

It should be noted that in each of these points, involvement of the entire education community is an important part of the strategy for without the cooperation and help of the typical classroom teacher, nothing lasting can be accomplished.

Forming a policy for the nation on computer education will have to include input from many organizations. It will have to go into much greater detail than has this paper. The three major points listed above form the nucleus around which such a policy could be built.

ELECTRONIC MAIL SYSTEM

RICHARD ANDREOLI and JERALD MELNICK
DIGITAL EQUIPMENT CORPORATION
ONE IRON WAY
MARLBORO, MA

ABSTRACT

A combination of office automation, word processor and communications system, this system gives the user the ability to generate, edit, electronically deliver and file memos, messages or documents. Other features include the ability to keep personal calendars, generate large documents and develop and edit graphic displays.

INTRODUCTION

"ELECTRONIC MAIL" has recently been used to label many different means of transmitting information electronically. Telegraph, store and forward message switches, and communicating word processors are among the most widely used today. Facsimile document transmission is attracting a large amount of new interest and is also typically described as "ELECTRONIC MAIL". This method employs a page scanner which converts images into raster and then transmits the signal (usually over telephone lines) to a decoding device which then prints the image.

EMS-11

The ELECTRONIC MAIL SYSTEM (EMS) discussed here is a DEDICATED-DOCUMENT based Mail System operating under DSM-11. Presently, 45 terminals are connected to a PDP-11/55. Terminals are owned or shared by users and access to the system is protected by requiring a user specified password when logging in. In this configuration, all EMS users have access to a common data base. Messages (memos and documents) are transferred with a minimal time delay and the system provides many additional aids to automate office work, as well as supporting a number of marketing functions. In addition, a telephone link to the Corporate Message Service (CMS - a store and forward message switch) allows EMS users to send messages from their terminals to many parts of the world.

OBJECTIVES

The objective of EMS is eliminate the large amount of paper generated in the operation of an office. The system provides an effective and dependable means of communication, as well as an aid in editing, copying, filing, and scheduling of events. The benefit is better utilization of human resources and a more efficient way of communicating.

DESCRIPTION

The system is comprised of four subsystems:

1. CALENDAR KEEPER
2. MEMO HANDLER
3. DOCUMENT HANDLER
4. GRAPHICS HANDLER

A fifth subsystem exists which allows the SYSTEM COORDINATOR to maintain the lists and profiles of all EMS users, and update distribution lists for convenient memo distribution. The routines are accessed by choosing a selection from a list of available options or MENU. The following is a brief outline of the routines within each of the subsystems.

CALENDAR KEEPER

The objective of the CALENDAR KEEPER is to create an efficient and organized means of personal calendar scheduling, along with the added capability which finds common unreserved time slots for up to five people.

The CALENDAR KEEPER consists of five sections which are displayed as choices on the "CALENDAR KEEPER MENU" when the routine is first accessed. These sections are:

1. DISPLAY CALENDAR
   Displays any individual's calendar for any specified date.

2. SCHEDULE
   Permits the scheduling of events or activities on the users's personal calendar.

3. TIME SLOT SCAN
   Searches the calendars of up to five persons for a desired number of contiguous unreserved time slots.

4. AVAILABLE TIME SLOT
   Displays calendar of up to five persons and allows the scheduling of common unreserved time slots.

5. CALENDAR DIRECTORY
   Lists all of those individuals who currently have a calendar.

MEMO HANDLER

The objective of the memo handler is to provide an efficient means of drafting, editing, copying, filing, and distributing memos.

Capabilities of the MEMO HANDLER feature:

1. Automated distribution of INTERNAL mail with negligible time delay. Hardcopy production is available, along with mail address label printing for all EXTERNAL mail.

2. Complete editing capabilities.

3. Automated filing and memo retrieval by "SUBJECT SEARCH".

The MEMO HANDLER consists of six sections which are displayed as choices on the "MEMO HANDLER MENU" when the subsystem is first accessed. These sections are:

1. GENERATE MEMO
   Permits drafting, editing, and sending of a memo.

2. SEND MEMO
   Allows the user to send memos from the UNSENT file.

3. READ MEMO
   Displays any memo contained in a user's files.

4. MEMO AND FILE MANAGER
   Creates user defined files, deletes files, deletes memos, and files memos.

5. SEARCH MEMO SUBJECT
   Searches through all or selected memos for memos with a specified subject.

6. LIST DIRECTORIES
   Lists all EMS users, all available distribution lists, all individuals within a specified distribution list and all distribution groups in which a specified individual may be found.

DOCUMENT HANDLER

The objective of the DOCUMENT HANDLER is to provide an efficient means to draft, store, edit and print documents.

The DOCUMENT HANDLER is particularly useful for generating reports or plans which require periodic updates and printing. The document may also be forwarded to the EMS users along with an attached FORWARDING MESSAGE.

The DOCUMENT HANDLER consists of six sections which are displayed as choices on the DOCUMENT HANDLER MENU when the routine is first accessed. These sections are:

1. GENERATE DOCUMENT
   Permits document drafting with page and line format controls.

2. EDIT DOCUMENT
   Allows editing of documents.

3. PRINT DOCUMENT
   Prints hardcopy of selected documents.

4. DELETE DOCUMENT
   Deletes any unwanted documents.

5. DOCUMENT DIRECTORY
   Lists all documents which belong to the user.

6. FORWARD DOCUMENT
   Allows the user to send a copy of the document along with a cover letter to any EMS user.

GRAPHICS HANDLER

The purpose of the GRAPHICS HANDLER is to provide the user (without programmer intervention) with the ability to create their own graphs. This will be performed by the user answering questions pertaining to data manager routines and graphic manager routines.

The DATA MANAGER MENU consists of the following five sections:

1. ENTER NEW DATA FILE
   Permits the setting up of a data file consisting of X and Y point values.

2. EDIT EXISTING DATA FILE
   Allows the editing of a data file.

3. DISPLAY EXISTING DATA FILE
   Displays on terminal or hardcopy a users data file.

4. DELETE DATA FILE
   Deletes any unused and unwanted data file.

5. DATA FILE DIRECTORY
   Lists all data files which belong to the user.

The GRAPHICS MANAGER MENU consists of the following five sections:

1. GENERATE NEW GRAPH
   Allows the user to generate a graph using up to two data files, setting scales and labeling the graph.

2. EDIT GRAPH
   Allows the editing of a graph: such as, displaying the graph as a histogram or graph, change scaling and/or labeling, etc.

3. DISPLAY EXISTING GRAPH
   Displays any graph which belongs to the user.

4. DELETE GRAPH
   Deletes any unwanted graphs.

5. GRAPH FILE DIRECTORY
   Lists all graphs which belong to the user.

FUTURE CONSIDERATIONS

This project was originally started with a small group of managers and secretaries operating with 20 terminals on an 11/40. The first system was written in MUMPS-11 and later converted to DSM-11. As previously mentioned, 45 terminals are now connected to an 11/55 and a directory of 300 users is now on file. The link to the CMS system has greatly improved the flexibility and usefulness of EMS. Approximately one third of all memos written on EMS are now transferred for distribution through the Corporate Message Service. Future plans include receiving messages from CMS.

Efforts are now being made to add the many new ideas and features suggested by users. The design and implementation of a network of DSM-11 based Electronic Mail systems is currently underway. Each node will be responsible for switching its own outgoing mail to the appropriate system. Communication between nodes will take place via dial-up lines. Directory and system information will be passed in a similar fashion, allowing directory updates at each node.

Finally, a link will be available to all WORD PROCESSING SYSTEMS (WPS-8). WPS-8 files will be transferred to and from the EMS DOCUMENT HANDLER for temporary storage or transmission to users and other WPS-8 systems.

COMPUTERIZED FINANCIAL ACCOUNTING: JOURNAL ENTRIES THROUGH FINANCIAL STATEMENTS

Clairmont P. Carter
Babson College
Babson Park, Mass. 02157

ABSTRACT

Transaction analysis is often considered to be the key to understanding financial accounting. However, because of the many mechanical, bookkeeping processes involved between transaction analysis and the preparation of financial statements, it is quite possible for students to fail to clearly see the financial statement impact of specific business transactions. Through the use of a financial accounting computer package it is now possible for students to concentrate on transaction analysis while the computer handles most of the mechanics and prepares the financial statements. As a result, students can quickly see how their analysis and interpretation of specific transactions directly influence the financial statements. Specifically, there has been developed a financial accounting computer package which requires students to input journal entries only, after which the computer will update the general ledger, prepare a trial balance, and generate financial statements. The computer program is such that each student has a separate set of files in which her or his company input is maintained and from which the financial statements are prepared. The financial accounting computer package includes 12 sets of transactions covering the topics most frequently discussed in a financial accounting principles course, such as, cash, receivables, inventories, fixed assets, liabilities, and owners' equity. The transaction sets apply to an unincorporated, service enterprise which evolves into an incorporated, merchandising firm.

INTRODUCTION

The accounting process consists of the following ten steps:

1. The preparation of general journal entries.
2. Posting the general journal entries to the general ledger.
3. The preparation of a trial balance before adjustment.
4. The preparation of adjusting journal entries.
5. Posting the adjusting entries to the general ledger.
6. The preparation of the adjusted trial balance.
7. The preparation of the closing entries.
8. Posting the closing entries to the general ledger.
9. The preparation of the post-closing trial balance.
10. The preparation of the financial statements.

Of the ten steps described above, all but two steps are of a highly mechanical nature. Only Steps 1 and 4, the preparation of journal entries, require detailed analyses. As a consequence, the analysis of business transactions which result in general journal entries and/or adjusting entries is often considered to be the key to understanding financial accounting. However, the need for the other, mechanical steps in the accounting process often makes it difficult for students to clearly see the impact that their analysis of business transactions can have upon the financial statements which result from the complete accounting process. In fact, the mechanics of the accounting process are so cumbersome that accounting is quite often taught on a topic-by-topic basis, in which the accounting effects of such topics are discussed but rarely demonstrated on a complete set of financial statements. For example, in many accounting principles courses it is customary to discuss the effects of different inventory costing techniques and different depreciation

methods but such effects are rarely, if ever, demonstrated on a complete set of financial statements. The bookkeeping process is so cumbersome that it often prohibits such exploration of the financial statement effects of many alternative accounting practices. As a result, students obtain some idea of the impact of alternative accounting practices, but rarely do the students have an opportunity to put such effects into proper prospective by viewing them in relation to the other topics which impact on the financial statements.

To overcome the mechanical nature of the accounting process, to allow students to concentrate on transaction analysis and still be able to see the impact of their analysis upon a complete set of financial statements, a financial accounting computer package has been developed. In essence, the financial accounting computer package requires the students to perform the transaction analysis and the computer performs the mechanical, bookkeeping process, resulting in the financial statements.

ELEMENTS OF THE FINANCIAL ACCOUNTING COMPUTER PACKAGE

The financial accounting computer package may be viewed as three separate elements - a series of business transactions, the computer input, and the computer output - each of which is discussed below.

Business Transactions

The financial accounting computer package has been designed in such a way that the students are viewed as the accountants for a business enterprise. To both reflect the operations of the business and to expose the students to the many different topics discussed in a financial accounting principles course, 12 sets of accounting transactions have been developed. Each set of transactions covers one month's business activity. For each month's activity, it is the students' responsibility to analyze the transactions and, with the aid of the computer, prepare the monthly financial statements.

The 12 transaction sets are designed to cover most of the topics normally discussed in a financial accounting principles course, such as, cash, receivables, inventories, fixed assets, liabilities, and owners' equity. Furthermore, in order to include coverage of the different types of businesses and the different capital structures of businesses, the transaction sets reflect several possible transactions in a company's development, as presented in Exhibit 1.

An examination of Exhibit 1 reveals that the transaction sets do not cover manufacturing enterprises or partnerships. Neither of these topics was included in the transaction sets because such topics are discussed only briefly in most financial accounting principles courses. However, it should be noted that the computer program is flexible enough that the chart of accounts and, thus, the financial statements, may be modified to reflect either topic if an instructor so desires. Of course, such modification would also require a change in one or more of the 12 transaction sets.

As illustrations of the types of events covered in the transaction sets, Exhibits 2, 3, and 4 present examples of transactions effecting the Stone Automotive Company. Exhibit 2 reflects the events of a service company which is a sole proprietorship. The transaction set in Exhibit 2 is the first one presented to the students and, as such, has been designed to be brief and relatively simple, since, for many students, this may not only be their first experience with accounting but may also be their first exposure to the computer. A quick review of Exhibits 3 and 4 reveals that the transaction sets become more complicated as the students progress through the accounting course. Exhibit 3 presents transactions relating to the operations of a merchandising company which is a sole proprietorship and Exhibit 4 presents transactions relating to a merchandising firm which is a corporation. Thus, as illustrated in Exhibits 2, 3, and 4, the students' company evolves from a sole proprietorship, service company to a merchandising corporation. The transaction sets have been designed to reflect such evolution and to show its effects on the financial statements.

EXHIBIT 2
TRANSACTION SET
SERVICE COMPANY, SOLE PROPRIETORSHIP

Transaction Set #1

1/1/78	Jessica Stone organized the Stone Automotive Company by investing $2,000 of her cash. The Stone Automotive Company will provide high quality automobile engine service on an appointment only basis. The service will be provided in Ms. Stone's personal garage using tools owned by Ms. Stone.
1/8/78	The Company purchased, on account $1,350 of automotive supplies which will be used to service various automobiles.

EXHIBIT 1
TRANSACTION SET COVERAGE

Transaction Sets	Business Type	Capital Structure
1-4	Service Company	Sole Proprietorship
5-9	Merchandising Company	Sole Proprietorship
10-12	Merchandising Company	Corporation

1/20/78 The Company borrowed $2,400 from a friend of Ms. Stone by Ms. Stone's signing of a note payable. Beginning on 2/20/78 the note will be repayed in twelve monthly installments of $200. The loan is interest free.

1/25/78 The Company paid for half of the automotive supplies purchased on 1/8/78. The balance is to be paid on 2/5/78.

1/31/78 The Company purchased a small piece of land to be used as a parking lot. The land cost $1,000 and will be paid for on 2/28/78.

EXHIBIT 3
TRANSACTION SET
MERCHANDISING COMPANY, SOLE PROPRIETORSHIP

Transaction Set #5

5/1/78 After a good deal of research, Ms. Stone decided that she would be better off by selling auto parts than by using them to repair cars. As a result, as of 5/1/78 her business changed from auto engine repair to auto parts distribution.

5/1/78 The supplies on hand were examined and determined to be in salable condition. Hint: The supplies should be reclassified as inventory.

5/4/78 Jessica purchased, on account, $5,000 of auto parts.

5/5/78 Ms. Stone paid April's bills received 4/25/78.

5/10/78 Ms. Stone received the cash on all 4/30/78 accounts receivable.

5/13/78 Jessica paid cash for half the auto parts purchased on 5/4/78.

5/15/78 Ms. Stone made her second payment on the bank note.

5/16/78 Because of the volume of business, Ms. Stone hired Linda Fetters as a part-time sales clerk. Ms. Fetters is to be paid a wage of $2.50 per hour.

5/25/78 May's utility bill was $52, the phone bill was $72, and the advertising bill was $105. All three bills will be paid 6/5/78.

5/31/78 May's sales were $4,450, of which $3,000 were cash, the balance to be collected in June.

5/31/78 Auto parts on hand totaled $3,000.

5/31/78 Ms. Fetters was paid for all 20 hours she worked in May.

5/31/78 Ms. Stone withdrew $950 for her personal use.

EXHIBIT 4
TRANSACTION SET
MERCHANDISING COMPANY, CORPORATION

Transaction Set #10

10/1/78 Ms. Stone decided to incorporate her business. After fulfilling all the legal requirements, the result was 10,000 authorized shares of $1 par value common stock. 6,000 of the shares were taken by Ms. Stone in return for her capital interest in the business.

10/1/78 Ms. Stone decided the business needed a truck more than a car. As a result, she sold the car for $1,500. She then paid $2,400 cash for a 1974 Ford pick-up truck, which will be depreciated over four years, using the sum-of-the-years'-digits method and assuming no salvage value.

10/5/78 Stone paid September's bills received 9/25/78.

10/7/78 Stone paid the October rent.

10/10/78 Ms. Fetters subscribed to 100 shares of the $1 par value Stone Incorporated common stock. The subscription price was $5 per share.

10/10/78 The note receivable received on 7/10/78 and discounted on 7/25/78 was settled in full by the maker of the note. Hint: The maker paid the bank and the bank notified Ms. Stone.

10/12/78 Stone paid for the auto parts purchased 9/19/78.

10/15/78 Stone made her seventh payment on the bank note.

10/17/78 Stone increased the petty cash fund by $50.

10/20/78 Auto parts purchased on account with terms of N/30 were:

Part Class	Quantity Purchased	Invoice Price/Unit
Transmissions	22	270.00
Mufflers	10	4.50
Exhaust Pipes	5	74.40
Axles	4	43.50

10/23/78 Ms. Fetters paid for the stock subscribed on 10/10/78.

10/25/78 October's utility bill was $180, the phone bill was $74, and the advertising bill was $215. All three bills will be paid on 11/5/78.

10/31/78 October's sales were $10,250, of which $4,250 were cash, the balance were credit sales under terms of 2/10, N/30.

10/31/78 October credit customers owing $2,000 paid within the discount period.

10/31/78 The 8/31/78 accounts receivable of $350 were collected. $2,000 of the September accounts receivable were collected and $175 were judged to be uncollectible.

10/31/78 A physical inventory revealed the following to be on hand.

Part Class	Quantity On Hand	Current Market Price
Transmissions	18	270.00
Mufflers	20	4.75
Exhaust Pipes	5	74.00
Axles	2	43.00

10/31/78 Petty cash expenditures in October were: $25 for office supplies and $43 for postage.

10/31/78	Ms. Fetters was paid for 250 hours worked and Ms. Stone was paid a salary of $1500.
10/31/78	Truck expenses paid in cash were $100.
10/31/78	October's bank service charge was $5.
10/31/78	Plaston stock has a market value of $12.25/share.

Computer Input

The computer input for the financial accounting computer package is designed around a series of commands which allow students to select the specific computer function they want performed. Exhibit 5 briefly explains the commands available to the students.

EXHIBIT 5
COMPUTER COMMANDS

Command	Purpose
1	Set up all accounts in the general ledger. This command eliminates all old balances which may exist and prepares the general ledger to accept the results of any new journal entry postings.
2	List the chart of accounts.
3	Modify the chart of accounts and the general ledger.
4	Prepare journal entries.
5	List the journal entries.
6	Post journal entries to the general ledger.
7	List the trial balance.
8	List the financial statements: income statement, capital statement, and balance sheet.
9	Exit from the program.
10	Help! List all available commands.

It should be noted in Exhibit 5 that the computer commands are quite similar to the accounting process described earlier. Specifically, the accounting process proceeds from journal entries to financial statements and so does the series of computer commands. Thus, in effect, the series of commands allows the students to perform the accounting process with the aid of the computer.

Once the students have selected their desired computer command, the computer responds with a series of simple questions which must be answered by the students. For example, if the students select command 4, the journal entry command, the computer will ask for (1) the transaction date, (2) an explanation of the transaction, (3) the account number, (4) debit or credit, and (5) the dollar amount of the debit or credit. Each question must be answered before the next one is asked. Furthermore, questions 3, 4, and 5 are repeated until the total debits equal the total credits. Thus, the computer will not allow the students to present it with data which will result in financial statements which do not tie together.

In essence, the financial accounting computer package input is prepared by the interaction between the students and the computer. Through the use of a series of commands, the students indicate the operation they want performed. The computer responds with a series of questions, the answers to which will allow the computer to perform the desired operation. From a practical standpoint, student-computer interaction is most pronounced during the journal entry operation (command 4). It is at this stage that the students must inform the computer of the results of the students' analysis of specific business transactions. Once the computer has been so informed, through the use of the proper series of commands, the computer will generate a complete set of financial statements. An example of the complete input requirements and the output results is presented in a following section of this paper. It should be especially noted that the key to obtaining reasonable financial statements through the use of this financial accounting computer package is the students' analysis of business transactions. If students analyze transactions incorrectly, they will get unreasonable results. The computer will not correct the students' inability to do accounting.

Computer Output

The output of the financial accounting computer package depends upon the computer command selected by the students. Essentially, the output consists of the following: (1) the chart of accounts, which is presented as Exhibit 6, (2) the general journal, presented as Exhibit 7, (3) the trial balance, presented as Exhibit 8, (4) the income statement, presented as Exhibit 9, (5) the capital statement, presented as Exhibit 10, and (6) the balance sheet, presented as Exhibit 11. In addition to the computer output presented in Exhibits 6 through 11, the financial accounting computer package results in a print-out of the students-computer interaction, as presented in the following section of this paper.

EXHIBIT 6
CHART OF ACCOUNTS

Account Number	Account Name
1000	Cash
1100	Accounts Receivable
1110	Allowance for Doubtful Accts.
1120	Notes Receivable
1150	Interest Receivable
1200	Prepaid Insurance
1250	Prepaid Rent
1300	Supplies
1350	Other Current Assets
1500	Furniture & Fixtures
1520	Accum. Deprec.-Furn. & Fixt.
1550	Machinery & Equipment
1570	Accum. Depr.-Mach. & Equip.
1600	Buildings

1620	Accum. Deprec.-Buildings
1650	Automobiles
1670	Accum. Deprec.-Automobiles
1700	Land
1800	Patents
1850	Goodwill
1900	Investments
2000	Accounts Payable
2050	Notes Payable (current)
2100	Interest Payable
2150	Wages Payable
2200	Unearned Revenue
2400	Other Current Liabilities
2500	Mortgage Payable
2600	Notes Payable (long term)
3000	Capital
3100	Revenue & Expense Summary
3200	Withdrawals
4000	Service Revenue
4050	Sales
4100	Interest Income
4200	Other Revenue
5000	Cost of Goods Sold
5010	Purchases
5020	Purchase Discounts
5030	Purchase Returns
5100	Salaries Expense
5150	Rent Expense
5200	Advertising Expense
5250	Utilities Expense
5300	Telephone Expense
5350	Repairs & Maintenance Expense
5400	Insurance Expense
5450	Property Taxes Expense
5500	Interest Expense
5550	Supplies Expense
5600	Amortization of Goodwill
5650	Depreciation - Furn. & Fixt.
5700	Depreciation - Mach. & Equip.
5750	Depreciation-Buildings
5800	Depreciation-Automobiles

EXHIBIT 7
GENERAL JOURNAL

Transaction Date	Explanation	Account #	Debit	Credit
xx/xx/xx	- - -	xxxx	xx	
		xxxx		xx
xx/xx/xx	- - -	xxxx	xx	
		xxxx		xx
Totals			xx	xx

EXHIBIT 8
TRIAL BALANCE
AS OF XX/XX/XX

Account Number	Account Name	Debit Balance	Credit Balance
xxxx	---	xx	
xxxx	---	xx	
xxxx	---		xx
Totals		xx	xx

EXHIBIT 9
INCOME STATEMENT
FOR PERIOD ENDING XX/XX/XX

	Current Month	
REVENUES		
Service Revenue	x	
Interest Income	x	
Other Revenue	x	
TOTAL REVENUE		x
OPERATING EXPENSES:		
Salaries Expense	x	
Rent Expense	x	
Advertising Expenses	x	
Utilities Expense	x	
Telephone Expense	x	
Repairs & Maintenance Expense	x	
Insurance Expense	x	
Property Taxes Expense	x	
Interest Expense	x	
Supplies Expense	x	
Amortization of Goodwill	x	
Depreciation - Furn. & Fixt.	x	
Depreciation - Mach. & Equip.	x	
Depreciation-Buildings	x	
Depreciation-Automobiles	x	
TOTAL OPERATING EXPENSES		x
NET INCOME		x

EXHIBIT 10
CAPITAL STATEMENT
FOR PERIOD ENDED XX/XX/XX

Beginning Balance		xx
Add: investments	x	
net income	x	xx
Deduct: withdrawals	x	
net loss	x	xx
Ending Balance		xx

EXHIBIT 11
BALANCE SHEET
AS OF XX/XX/XX

ASSETS

Current Assets:			
Cash		x	
Accounts Receivable	x		
Allowance for Doubtful Accts.	x	x	
Notes Receivable		x	
Interest Receivable		x	
Prepaid Insurance		x	
Prepaid Rent		x	
Supplies		x	
Other Current Assets		x	
Total Current Assets			x
Plant & Equipment			
Furniture & Fixtures	x		
Accum. Deprec.-Furn. & Fixt.	x	x	
Machinery & Equipment	x		
Accum. Depr.-Mach. & Equip.	x	x	
Buildings	x		
Accum. Deprec.-Buildings	x	x	
Automobiles	x		
Accum. Deprec.-Automobiles	x	x	

Land	x	
Total Plant & Equipment		x
Other Assets:		
Patents	x	
Goodwill	x	
Investments	x	
Total Other Assets		x
Total Assets		x
LIABILITIES & OWNERS EQUITY		
Current Liabilities:		
Accounts Payable	x	
Notes Payable (current)	x	
Interest Payable	x	
Wages Payable	x	
Unearned Revenue	x	
Other Current Liabilities	x	
Total Current Liabilities		x
Long Term Liabilities:		
Mortgage Payable	x	
Notes Payable (long term)	x	
Total Long Term Liabilities		x
Total Liabilities		x
Owners Equity:		
Capital	x	
Expense & Revenue Summary	x	
Withdrawals	x	
Total Owners Equity		x
Total Liabilities & Owners Equity		x

FINANCIAL ACCOUNTING COMPUTER PACKAGE ILLUSTRATED

Many of the elements of the financial accounting computer package are presented in the following illustration of how students would make use of the package to generate March's financial statements for the Stone Automotive Company. At this point in the students' use of the package, January's and February's events have already been processed and the appropriate financial statements have been generated.

Business Transactions

Exhibit 12 presents the March transactions of the Stone Automotive Company.

EXHIBIT 12
STONE AUTOMOTIVE COMPANY
MARCH TRANSACTIONS

3/1/78	Because her garage was getting rather crowded, Ms. Stone signed a one-year lease on her neighbor's garage. The lease calls for monthly rental payments of $250 to be made on the first day of each month. Jessica paid $750 to her neighbor on 3/1/78.
3/5/78	Ms. Stone paid February's bills received on 2/25/78.
3/15/78	Because her friend was experiencing some financial problems, Ms. Stone decided to repay her loan. To accomplish this, Jessica borrowed $2400 from the Chestnut Hill Bank by signing a one-year note. Beginning on 4/15/78, the note will be repayed in 12 monthly installments of $200 plus 12% interest on the declining balance. Using some (or all) of the money borrowed from the bank, Jessica repayed her friend for the money borrowed on 1/20/78.
3/20/78	Ms. Stone withdrew $750 for her personal use.
3/25/78	March's utility bill was $153, the phone bill was $46, and the advertising bill was $89. All three bills will be paid on 4/5/78.
3/31/78	During March, Ms. Stone provided her customers with $1,500 services of which she received $800 in cash.
3/31/78	Automotive supplies on hand totaled $600.

Inasmuch as the students have processed January's and February's transactions, they have the chart of accounts presented in Exhibit 6. Using the chart of accounts and February's balance sheet, the students would analyze March's transactions and prepare the general journal entries presented in Exhibit 13.

Once the students have prepared the general journal entries, they are ready to make use of the computer, as follows:

Computer Processing

To gain access to and make use of the financial accounting computer package, the students must take the following steps:

1. Turn on the computer terminal.
2. Type HELLO 70,1 (or another appropriate access code).
3. The computer responds by requesting the appropriate password, to which the students respond by typing AC200.
4. The computer responds with a statement informing the students that they have gained access to the financial accounting computer package and everything is ready to run.
5. The student responds by typing RUN ACCTG.
6. The computer responds by requesting the students' three digit character code. Note: Each student has a separate set of files in which the company's general ledger is maintained. The students respond to the computer by typing in the code number assigned by the instructor.
7. The computer responds by requesting a command and indicating that command 10 may be helpful to the students.
8. At this stage, if the students have any doubts regarding any of the commands, the students respond to the computer by typing 10.
9. The computer responds with a command listing similar to that presented in Exhibit 5 and asks the students to select a command.
10. Inasmuch as the students have prepared the journal entries presented in

EXHIBIT 13
STONE AUTOMOTIVE COMPANY
MARCH GENERAL JOURNAL ENTRIES

Date	Explanation	Account Number	Account Name	Debit	Credit
3/1/78	Garage Lease	5150	Rent Expense	250	
		1250	Prepaid Rent	500	
		1000	Cash		750
3/5/78	A/P Payment	2000	A/P	200	
		1000	Cash		200
3/15/78	Bank Loan	1000	Cash	2,400	
		2050	Notes Payable		2,400
3/15/78	Loan Payment	2050	Notes Payable	2,200	
		1000	Cash		2,200
3/20/78	Withdrawal	3200	Withdrawal	750	
		1000	Cash		750
3/25/78	Expense Recognition	5250	Utility Expense	153	
		5300	Telephone Expense	46	
		5200	Advertising Expense	89	
		2000	A/P		288
3/31/78	Revenue Recognition	1000	Cash	800	
		1100	A/R	700	
		4000	Service Revenue		1,500
3/31/78	Supplies Expense	5550	Supplies Expense	450	
		1300	Supplies		450
3/31/78	Interest Expense	5500	Interest Expense	12	
		2100	Interest Expense		12
3/31/78	Closing	4000	Service Revenue	1,500	
		3100	E & R Summary		1,500
3/31/78	Closing	3100	E & R Summary	1,000	
		5150	Rent Expense		250
		5250	Utility		153
		5300	Telephone		46
		5200	Advertising		89
		5550	Supplies Expense		450
		5500	Interest Expense		12
3/31/78	Closing	3100	E & R Summary	500	
		3000	Capital		500
3/31/78	Closing	3000	Capital	750	
		3200	Withdrawals		750

Exhibit 13, the students are ready to input such journal entries into the computer. Therefore, the students respond to the computer by typing 4.

11. The computer responds by indicating that this command is for the processing of journal entries. As a result of this command, the following interaction takes place between the computer and the students.

Computer Question	Students' Response
Date of transaction: xx/xx/xx	3/1/78
Explanation	Lease Payment
Account Number	5150
DR of CR	D
Debit Rent Expense for	250
Account Number	1250
DR of CR	D
Debit Prepaid Rent for	500
Account Number	1000
DR of CR	C
Credit cash for	750

Once the total debits equal the total credits, the computer lists the journal entry, as follows:

3/1/78	5150	Rent Expense	$250	
	1250	Prepaid Rent	500	
	1000	Cash		$750

* * * Lease Payment * * *

12. After the journal entry is processed the computer asks if the students wish to delete the entry and start again.
13. If the students respond yes, the journal entry is deleted from the computer files. If the answer is no, the journal entry is not deleted.
14. The computer responds to the students by asking if any more journal entries will be prepared.
15. If the students respond yes, then the process described in steps 11 through 14 will be repeated. If the response is no, the computer requests a new command.
16. After the students have processed all their journal entries, they would respond with command 6, which posts the entries to the general ledger.
17. The computer responds by posting the entries and asking for the next command.
18. The students would probably respond with command 7 which results in the

trial balance.

19. The computer responds by listing the trial balance presented in Exhibit 14 and requests a new command.

EXHIBIT 14
STONE AUTOMOTIVE COMPANY
TRIAL BALANCE
AS OF 3/31/78

Account Number	Account	Debit	Credit
1000	Cash	$1,950	
1100	Accounts Receivable	700	
1250	Prepaid Rent	500	
1300	Supplies	600	
1700	Land	1,000	
2000	Accounts Payable		$ 288
2050	Notes Payable		2,400
2100	Interest Payable		12
3000	Capital		2,300
3200	Withdrawals	250	
4000	Service Revenue		1,500
5150	Rent Expense	250	
5200	Advertising Expense	89	
5250	Utilities Expense	153	
5300	Telephone Expense	46	
5500	Interest Expense	12	
5550	Supplies Expense	450	
		$6,500	$6,500

20. After reviewing the trial balance, the students would respond with command 8, which results in the financial statements.

21. The computer responds by indicating that this command results in financial statements and the following interaction takes place.

Computer Question	Students' Response
Do you want the income statement	yes
" " " " capital statement	no
" " " " balance sheet	no

The students want only the income statement at this point because the capital statement and balance sheet require the preparation of closing entries. Once the students have responded, the computer lists the financial statement requested. As a result of the above command, the computer would list the income statement presented in Exhibit 15.

22. After the income statement is prepared, the computer asks for its next command.

23. The student would respond with a 4 and, using the trial balance obtained in Step 19, would prepare the closing entries needed.

24. The computer would process the journal entries and request a new command.

25. The student would respond with command 6 to post the journal entries.

26. The computer would post the journal entries and request a new command.

27. At this stage the student could request a new trial balance listing, which would be similar to Exhibit 14, or could request the financial statements by choosing command 8.

28. Responding to command 8, the computer would determine which financials are desired by asking the same questions indicated in step 21. At this point the students would want the capital statement and the balance sheet, which the computer would list as presented in Exhibits 16 and 17.

EXHIBIT 16
STONE AUTOMOTIVE COMPANY
CAPITAL STATEMENT
FOR PERIOD ENDED 3/31/78

Beginning Balance		$2,300
Add: Net Income	$500	500
Deduct: Withdrawals	$750	750
Ending Balance		$2,050

EXHIBIT 15
STONE AUTOMOTIVE COMPANY
INCOME STATEMENT
FOR PERIOD ENDED 3/31/78

	Current Month		Year-to-Date	
Revenues:				
Service Revenue	$1,500		$2,300	
Total Revenues		$1,500		$2,300
Operating Expenses				
Rent Expense	250		250	
Advertising Expense	89		128	
Utilities Expense	153		278	
Telephone Expense	46		82	
Interest Expense	12		12	
Supplies Expense	450		750	
Total Operating Expenses		1,000		1,500
Net Income		$ 500		$ 800

EXHIBIT 17
STONE AUTOMOTIVE COMPANY
BALANCE SHEET
AS OF 3/31/78

ASSETS		
Current Assets:		
Cash	$1,950	
Accounts Receivable	700	
Prepaid Rent	500	
Supplies	600	
Total Current Assets		$3,750
Plant & Equipment:		
Land	1,000	
Total Plant & Equipment		1,000
Total Assets		$4,750
LIABILITIES & OWNERS' EQUITY		
Current Liabilities:		
Accounts Payable	$ 288	
Notes Payable	2,400	
Interest Payable	12	
Total Current Liabilities		$2,700
Owners' Equity		
Capital	2,050	
Total Owners' Equity		2,050
Total Liabilities & Owners' Equity		$4,750

29. After the financial statements have been listed, the computer asks for a new command.
30. At this point the students may request a listing of all the journal entries by typing 5.
31. The computer would respond with a journal listing similar to Exhibit 13. The computer would then ask for a new command.
32. At this point the students have completed their transaction set and would exit from the program by typing 9.

SUMMARY AND IMPLICATIONS

The financial accounting computer package described in this paper allows students to interact with the computer and, as a result, generate financial statements through analyzing business transactions and preparing journal entries. Through the use of the package the students can better see the financial statement effects of their analysis of business transactions. Furthermore, the use of the computer package allows accounting instructors to relate the effects of specific accounting topics to a complete set of financial statements, without requiring students to make use of the many mechanical, cumbersome steps of the accounting bookkeeping process. For example, by varying the use of accounting methods among students, it is possible for accounting instructors to examine the effects of alternative accounting methods as in the areas of inventory costing and fixed asset depreciation.

A SYSTEM ACCOUNTING PACKAGE FOR RSX-11M

Gary Bernstein
Carmelo Granja
Alex Brown

BioMedical Engineering Unit
McGill University
Montreal, Canada

1. Introduction

RSX-11M is a multi-user, multi-task, event driven real-time operating system. When installed on a powerful midi-computer such as the PDP-11/70, it can manage the concurrent operation of real time programs, program development, network communications and a host of other facilities whose requests may emanate from any user logged on local or remote terminals and perhaps even another computer's terminal.

Despite all the capabilities available to users, RSX-11M does not provide any accounting facilities nor does it permit any type of performance measurement.

From a management point of view this situation becomes intolerable in a multi-user environment. This is especially true when users must be made accountable for the utilization of the resources available.

This lack of facilities also impedes users who wish to obtain system performance parameters to either compare with other computer facilities or simply to tune a program so as to achieve better performance.

2. Overview

This paper describes the design and implementation of a software system which logs and reports some of the computer resources consumed by users of an RSX-11M operating system. Statistics which are logged on a per-user basis include:

1) Total terminal connect time;
2) Total weighted terminal connect time, where the weighting factor is a function of the terminal's baud rate;
3) Total number of log-ons and log-offs;
4) Total number of disk-block-weeks;
5) Total number of I/O requests (QIO's);
6) Total CPU time (ticks) consumed by user tasks;
7) Total memory demands (kilo-word-ticks) consumed by user tasks.

The only global statistic currently logged is an accumulation of total connect time versus time-of-day which, when printed out as a histogram, provides an indicator of peak times of system usage.

Reports which can be generated by the system include:

1) Terminal connect time profile (histogram);
2) Numbered invoices sorted by user name;
3) Summary report of total resources consumed during the reporting period;
4) Summary financial report;
5) Detailed report of resources consumed by each user.

Utility functions which allow the operator to edit or zero selected portions of the logging file are also provided.

The accounting system has been in use since May, 1977 on a PDP-11/70 based multi-user RSX-11M system. During this time it has been found to be virtually free from error, consistent and accurate without significantly degrading system performance or jeopardizing system integrity. The version described in this report runs under RSX-11M V03.1.

3. General Design Approach

All logging, with the exception of disk block usage, is accomplished by sending message packets containing the statistics to be logged to a logging task, USR-LOG.TSK, which ultimately stores the statistics in a common logging file, USR-LOG.SYS. This method of inter-task communication allows considerable flexibility in system integration, debugging and testing, and allows new components to be added to the system in a hierarchical manner without stopping operation. For example, the first stage coded into the system was concerned with logging terminal statistics. The CPU logging was then added and tested on-line without disabling the terminal logging.

4. Logging of Terminal Statistics

Task 'ACL' (described in paragraph 5) sends messages to task USRLOG each time an individual user accomplishes a successful connect or disconnect request. The contents of the messages sent by ACL include the UIC at the time of connect or disconnect, the receive speed of the requesting terminal and the RSX-11M terminal UCB from which the log-on or log-off request was initiated. Upon receipt of a log-on message, task USRLOG notes the contents of the message in a scratch-pad area of the logging file, USRLOG.SYS. Upon receipt of the corresponding log-off message, task USRLOG retrieves the log-on data from the scratch-pad area and updates the appropriate record of the logging file to reflect:

1) The total time (minutes used by the UIC since the last time the logging file was zeroed;
2) The total weighted time used by the UIC;
3) The number of log-ins and log-offs performed under that UIC.

'Weighted time' is defined as the product of an index of terminal speed and terminal connect time. For example, terminals connected at speeds between 1200 baud and 2400 baud have an associated weighting index of 2. A session of 40 minutes on such a terminal would result in a weighted time of 80 minutes added to the accumulated weighted time in the log file.

In addition to the statistics noted above, at log-off time an elementary profile of connect time versus time of day is updated using the following algorithm:

> Each hour of the day is represented by one of 24 histogram 'bins' stored in the logging file. When a user logs off, the number of minutes used during each hour of connect is added to the appropriate bin. Quick log-ons and log-offs only cause the profile to be updated if the clock has 'ticked' at least one minute. Single precision 16-bit bins allow accummulation of up to 32,761 minutes per bin, which allows 34 days of logging a fully utilized, 16 terminal system before overflow occurs. An example of the profile printed by the reporting task is shown in Fig. 1.

Exception conditions are treated as follows:

1) When the UIC at log-off differs from the UIC at log-on, USRLOG determines the log-on UIC by terminal number and updates the accummulated statistics for the log-on UIC;

2) When the terminal speed index is different at log-on from log-off, the larger of the two indices is used to log weignted connect time;

3) Upon overflow of any of the logged parameters (unlikely) the maximum allowable value is maintained unmodified;

4) Upon system crashes no entries are made in the logging file of users who were active before the crash. Each time the system is rebooted a utility task, INILOG, is run from the STARTUP command file which zeroes the scratch-pad area of the logging file. It should be noted that INILOG could be coded so as to update the log file using the pre-crash information stored in the scratch-pad. However, our philosophy has been to somewhat pacify the irate user who has just experienced a system crash by not charging for the time used before the crash. It should also be noted that down-times scheduled with SHUTUP cause enforced log-offs to be executed for each logged-on terminal and thus do not result in any loss of accounting statistics.

Upon log-off a summary of logged statistics is displayed on the user terminal as illustrated in Fig. 2. 'Connect time' is the time used during the session while 'total time' is the accumulated time used by the UIC during the then-current billing period.

5. On-Line System Parameter Measurements

5.1 Design criteria

The factors that influenced the design of the on-line account logging software package can be summarized as follows:

a) Select a minimum set of parameters to measure on-line usage of system resources on a per user basis;

b) Design a method for measuring the above parameters ensuring a minimum amount of degradation in system performance;

c) Provide a communication facility with a secondary task to pass along the information gathered on-line;

d) Maintain all the desired capabilities on a single task including the ability to maintain executive hooks, thus avoiding having to patch the executive before SYSGEN;

e) Since the task needs to modify the executive on-line, provide facilities for restoring executive and deactivating the task;

f) For accounting purposes provide an off-line facility for accumulating on a per user basis the relevant accounting parameters. For individual users, provision to present on his terminal the information gathered for that session.

5.2 Parameters Measured

After taking into consideration several factors, among them the amount of difficulty in implementation and burden to the executive we settled on the selection of these parameters:

1) I/O count - number of QIO's issued by tasks running on behalf of a logged-on user.
2) CPU time - number of CPU ticks (1 tick = 1/60 second) used by tasks executing on behalf of a logged-on user.
3) Memory demands - number of kw*ticks, i.e. amount of memory in kw's demanded by a logged-on user times the amount of time those demands were in effect (ticks).

The I/O count provides a measure of the amount of work performed by the executive I/O-related modules on behalf of the user. Its measurement is accomplished by means of the intercept technique.

An intercept point (or hook) is placed in the executive module IOSUB. The code of this intercept simply counts the number of QIO requests issued by tasks running on behalf of logged-on users.

The CPU time is a direct measurement of CPU resources consumed by logged-on users. The memory demands parameter contains a built-in penalty for making memory demands while the system is busy servicing other users simultaneously; that is, demands made during "primetime" (system very busy) will stay in effect longer, therefore the resultant kw*t parameter will be larger.

The measurement of these two parameters is accomplished by a combination of intercept and sampling techniques. For CPU time measurements two intercepts are effected in executive module SYSX1. One of these is used for task switching and the other for null code execution. They are capable of maintaining a pointer to the user presently using the CPU. The pointer will be zero if no logged-on user or the null task is in control of the CPU.

The intercepts for measuring memory demands are placed in executive modules REQSB for starting a task and DREIF for stopping a task. The code in these two intercepts maintains the total present memory demand in kw's for each logged-on user.

Given the above parameters provided by the executive hooks, we are then able by means of the synchronous sampling mechanism provided by the 60 Hz system clock, to accumulate CPU-TIME and MEM.DEMANDS*TICKS for each user logged-on.

5.3 Implementation

5.3.1 Account logging task (ACL). ACL is the name of the task which implements sections a, b and c of the design criteria.

ACL is divided into two distinct parts of code. The first part contains all the necessary logic to implement the desired functions which are independent of the executive. The second part consists of position independent code which when hooked to the different executive modules will operate at the executive level. This code, although present within the task, will eventually be placed in the system pool and operate entirely from there.

The major functions of the ACL task are as follows:

1) maintain a section of position independent code and executive hooks which can be inserted when the task is run and removed when the task receives a code from another task to stop;

2) establish a communication protocol with the executive code to permit detection of a user logging on or off (or another task requiring to stop).

3) On detection of a user logging-on to insert a packet of memory on a self-threaded list which is also accessible from the executive code placed in the pool. The TI:UCB of the user is placed in the packet to identify the user. A message with pertinent parameters is also sent to USRLOG, the task used to accumulate information off-line.

4) On detection of a user logging-off to de-allocate from the pool the packet of memory used for on-line account logging of parameters and to send to USRLOG a message containing all the information gathered.

5) On detection of a "stop" message from ACLOFF to restore the executive to its normal state and to de-allocate from the pool the buffers used for code and packets.

NOTES

1. As presently implemented ACL requires the existance of EIS. This restriction can easily be removed however.
2. ACL can run checkpointable on any partition. While waiting for messages it remains in the STOP condition.
3. For reasons of possible executive throughput degradation, the 60 Hz clock must have been selected at SYSGEN as the system clock.
4. To provide stand-alone debugging capabilities, ACL can be assembled with different optional code.
5. ACL should never be aborted. To deactivate ACL the user must run ACLOFF. This task merely sends a message to ACL which permits ACL to deactivate itself properly.

5.3.2 ACLOFF -ACLOFF is the name of the task used to send the stop message to ACL.

5.3.3 ACLEP. The module ACLEP is assembled with task USRLOG and permits accumulation of the account-logging parameters gathered by ACL on a file in the system disk. It also presents on the user terminal the relevant information.

ACLEP needs the presence of the optional floating point unit in the system.

5.3.4 General Considerations. Measurement of the above three parameters for accounting purposes implies that the numbers thus obtained represent actual conditions of system behaviour regardless of individual user form of demands.

In this regard due to technical considerations the following assumptions are made:

1) All users' tasks run at the same default priority, are all checkpointable, are not SLAVE, and have not been started by the RUN directive (or RUN command with /RSI option).

Tasks that run at higher priority than default cannot be round-robinned; if they are not checkpointable they cannot be made swappable; SLAVE tasks are not accounted for since both their TI: and UIC may change dynamically by a RECEIVE directive; and finally the RUN directive (or RUN command with /RSI option) always causes the system to set the TI: device for the requested task to CO:.

2) HEL and BYE, which are used to log users on and off from terminals, are the only means of getting access to system resources.

At present we assume a friendly environment and expect all users to follow the rules given above.

If necessary it should not prove too difficult to make privileged those features that at present could get around the accounting procedure. These would be /PRI=, /SL, /-CP; i.e. always have appropriate defaults at run time for non-privileged users.

6. Measurement of System Disk Parameters

Another statistic logged by the system is the amount of disk storage claimed by each user.

The concept of an online, real-time, disk-storage "monitor" task (i.e. a task that would record, in some fashion, all file system activity: creations, deletions, extensions, etc) was rejected. Such a monitor would present considerable overhead to the filing system; it was decided that the dividends gained from such a precise measurement would not justify the projected decline in system performance.

Therefore, a "sampling" approach was taken. A Fortran program, 'BLK', was created. For each entry in the RSX account file, BLK examines the User File Directory (UFD). For each entry in a UFD, BLK examines the associated file header, from which it extracts the values for number of blocks used and number of blocks allocated. Totals for blocks-used and blocks-allocated to a particular UFD are stored in a random-access file. This file is logically organized into fixed-size "segments", where each segment represents one run of BLK, and contains information about the date and time of sampling, and totals for each UFD sampled during that run. At invoice time, the reporting task, 'REPLOG', examines each segment of the sample file and accumulates a "disk-block-week" total for each UIC encountered.

Samples are taken daily via the RSX scheduling mechanism, i.e. the STARTUP command file installs BLK, then schedules it to be run at 2:00 AM, with a reschedule interval of 24 hours.

The sample file is reset (i.e. all segments are logically deleted) after a set of invoices has been produced, whereupon the cycle begins anew.

7. Report Generation

The utility task, REPLOG, is responsible for generation of summary reports and invoices, and for general housekeeping operations on the data files. The interactive functions are as follows:

1) print summary report of system usage;
2) print terminal usage profile;
3) examine a single account;
4) modify a single account; this provision allows for manual intervention under exceptional circumstances;
5) print invoices; a financial summary is generated following the invoices;
6) zero the logging data file;
7) zero only the scratch area of the log data file, e.g. to reset the current status of the accounting system;
8) zero the terminal usage profile area of the log file;
9) zero the disk block sample file;
10) create new log files;
11) create new disk block sample file.

The task has been written using a "Structured FORTRAN" dialect. A preprocessor exists to translate the "SF" code into ANSI FORTRAN. Some examples of typical invoices and reports are shown in Figures 3 to 5.

Acknowledgement

This work was supported by a grant from the Macdonald Stewart Foundation.

```
>BYE
>
HAVE A GOOD AFTERNOON
10-FEB-78 14:49 TT11: LOGGED OFF
>
  TOTAL TIME ::  1390 MINUTES
CONNECT TIME ::    55 MINUTES
   I/O COUNT ::  883
    CPU TIME ::  2S-49T
 MEM. DEMAND :: 927E+3 KW*TICKS
```

Fig. 2 Log-Off Message Displayed on TI:

```
LOG-ON PROFILE

FROM 09-SEP-77 TO 02-FEB-78

                    % OF TOTAL TERMINAL TIME USED

          10        20        30        40        50        60
           !         !         !         !         !         !
 0 * ( 1.5%)
 1 * ( 1.2%)
 2 * ( 1.1%)
 3 * ( 0.9%)
 4 * ( 0.9%)
 5 * ( 0.9%)
 6 * ( 0.8%)
 7 * ( 0.8%)
 8 * ( 1.1%)
 9 ***** ( 3.7%)
10 ******* ( 6.0%)
11 ******* ( 6.4%)
12 ******* ( 6.4%)
13 ******** ( 7.1%)
14 ********** ( 9.9%)
15 *********** (10.6%)
16 *********** (10.2%)
17 ********* ( 7.9%)
18 ****** ( 5.2%)

19 ***** ( 4.4%)
20 ***** ( 3.9%)
21 ***** ( 4.0%)
22 *** ( 3.1%)
23 ** ( 2.0%)
>
```

Fig. 1 Connect Time Profile

```
MEDNET CHARGES FROM 16-JAN-78    TO 02-FEB-78        INVOICE #001051

NAME:    BROWN,A
ACCOUNT:    103    3

TERMINAL HOURS/CHARGE (@ 50% DISCOUNT)          1.28       0.63
AVERAGE TERMINAL SPEED GROUP/SURCHARGE          3.43       3.12
NUMBER LOG-ONS/CHARGE                              8       0.80
DISK_BLOCK_WEEKS USED/CHARGE                     455.      1.40
   TOTAL.....                                          $ 14.94

-------------------------------------------------------------------

                CHARGES SHOWN ABOVE ARE TO BE PAID.

      CHARGES SHOWN BELOW ARE FOR INFORMATIONAL PURPOSES ONLY
      AND REPRESENT THE PROJECTED RATE SCHEDULE FOR THE PERIOD
      APR-1978 TO MAR-1979. ANY COMMENTS RE THE PROJECTED RATE
      SCHEDULE WOULD BE APPRECIATED.

TERMINAL HOURS/CHARGE                           1.28       4.28
CPU MINUTES/CHARGE                              3.43       4.29
KILO_I/O-REQUESTS/CHARGE                       17.66       1.77
MEMORY DEMAND (KW_HOURS)/CHARGE                25.03       7.51
DISK_BLOCK_WEEKS USED/CHARGE                     455.      1.40
   TOTAL.....                                          $ 16.24

-------------------------------------------------------------------

                      GENERAL MESSAGE

    RSX-11M V3.1 IS NOW INSTALLED AND FULLY OPERATIONAL.

-------------------------------------------------------------------
```

Fig. 3 Sample Invoice

USER LOG REPORT FROM 16-JAN-78 TO 02-FEB-78 11:00:59 PAGE 1

NAME	UIC		TIME USED	WEIGHTED	LOGONS/OFFS	DBWKS
			I/O COUNT	CPU TIKS	MEMORY DEMAND (KWT)	
ARROTT,A	162	1	58	232	2/ 2	16.
			1276.	243.	1462945.	
BERNSTEIN,G	3	101	0	0	0/ 0	1207.
			0.	0.	0.	
BMEU LIBRARY	121	3	22	22	4/ 4	527.
			506.	156.	637353.	
COOPER,J	121	2	634	641	16/ 16	3753.
			12944.	10458.	17206950.	
DAVIES,P	123	5	838	910	10/ 10	2843.
			56521.	22644.	22894872.	
.						
.						
.						
WILLIS,D	141	1	84	265	6/ 6	2473.
			39310.	34053.	5791931.	
YAMAMOTO.Y.L	133	2	98	392	13/ 13	0.
			8943.	1899.	2992333.	

SUMMARY

NUMBER OF ACCOUNTS: 146

TOTAL HOURS USED: 537.867
TOTAL WEIGHTED HOURS: 1604.683
TOTAL # LOGONS: 1109
TOTAL # LOGOFFS: 1058
TOTAL DISK_BLOCK_WEEKS: 207830.

TOTAL I/O COUNT: 0.779273E+07
TOTAL CPUTIKS: 0.437771E+07
TOTAL MEMORY DEMAND: 0.103277E+10
>

Fig. 4 User Log Report

SUMMARY OF CHARGES 16-JAN-78 TO 02-FEB-78 11:00:59

(1) CURRENT CHARGING SYSTEM:

TERMINAL CONNECT TIME CHARGES:	4034.00	
TERMINAL SPEED SURCHARGES:	1067.02	
LOGON CHARGES:	110.90	
SUBTOTAL/AVG HOURLY CHARGE:	$ 5211.92	$ 9.69
DISK BLOCK CHARGES:	638.45	
TOTAL CHARGES:	$ 5850.37	

(2) PROJECTED CHARGING SYSTEM:

TERMINAL CONNECT TIME CHARGES:	537.87	
CPU TIME CHARGES:	1520.04	
I/O CHARGES:	779.27	
MEMORY DEMAND CHARGES:	1434.40	
SUBTOTAL/AVG HOURLY CHARGE:	$ 4271.58	$ 7.94
DISK BLOCK CHARGES:	638.45	
TOTAL CHARGES :	$ 4910.03	

SUMMARY OF RATES

(1) CURRENT CHARGING SYSTEM:

Hourly Rate Connect Time	$ 15.00
Surcharge For Terminal Speed	$ 1.00
Logon Charge	$ 0.10

(2) PROJECTED CHARGING SYSTEM:

Hourly Rate Connect Time	$ 1.00
Rate per CPU Hour	$ 75.00
" " " Minute	$ 1.25
Rate per 1000 I/O Requests	$ 0.10
Rate per KW_Hour Memory Demand	$ 0.30

(3) FOR EITHER SYSTEM:

Weekly Rate per Megabyte Disk Storage	$ 6.00
" " " Block	$0.0031

NUMBER OF INVOICES: 131
LAST INVOICE NUMBER USED: 001153
>

Fig. 5 Financial Summary Report

"RDCL"
REMOTE DEVICE VIA COMMUNICATION LINK

Alexander Brown and Gary Bernstein
BioMedical Engineering Unit
McGill University
Montreal, Quebec

ABSTRACT

This paper describes an RT-11 device driver which passes all filing requests to a remotely-located RSX-11M system over a serial, asynchronous communication link. A task at the RSX end of the link honours the filing request by directing I/O to either:

a) a foreign-mounted RT-11 volume;
or b) an RSX-11M file containing the image of an RT-11 volume.

Data and control messages are transferred between the two processors via a simple communications protocol. The driver may be used as an RT-11 system-device handler, or simply for inter-system file transfers.

INTRODUCTION

Within the community of PDP-11 users one serious impediment to effective distributed processing has been the lack of a reliable, easily-implemented system which allows communication between the RT-11 and RSX-11M operating systems. Described herein is a software package which provides these facilities by allowing the RT-11 user to perform, in a fairly transparent manner, filing operations to a "Remote Device over a serial asynchronous Communication Link" (RDCL). Specifically, the facilities provided by RDCL include:

1) Virtual Terminal - The ability to use the RT-11 system console (TT) as an RSX terminal (TI);

2) Remote Filing - The ability to initiate, from the RT-11 Satellite system, filing operations which perform I/O to a mass storage device located on the RSX-11M Host system;

3) Down-Line-Loading - The ability to load, from a mass storage device on the Host, a version of RT-11 which treats the Host's mass storage as though it were a local system device.

The implications of these facilities are numerous. Economies of scale become immediately obvious when several disk-less PDP-11 Satellite processors are able to communicate with an RSX system configured with a large-capacity system device. For example, it is possible to configure multiple core-only LSI-11 systems from which users are able to down-line-load RT-11, perform normal RT-11 program development activities and then run the application programs so developed. These programs are able to write data acquired during the application back to the Host device using standard RT-11 FORTRAN or SYSLIB I/O statements. By eliminating local mass storage completely, hardware costs are substantially reduced (typically one third of the cost of floppy-based systems is for the floppy), while software functionality and transparency are maintained.

Although core-only RDCL systems provide completely transparent program development facilities by fetching unmodified RT-11 system programs from the remote device, it is sometimes desirable to use RDCL as a complement to systems which have their own local mass storage. For example, in an environment where extensive program development activity takes place on the Satellite, it is advantageous to load the RT-11 system and system programs (which are often heavily overlaid) from a local floppy disk, rather than via the slower communication link. However, the programmer can still benefit from RDCL by using the remote device as he would any other RT-11 peripheral. For example, source programs, load modules, subroutine libraries and data can all be stored and directly manipulated on the larger remote device using standard RT-11 command strings. This helps to alleviate the problems often encountered with the limited amount of storage available for user files after the RT-11 system files have been placed on a floppy disk. Equally important in this mode is the application program's ability to file data on the RSX system, where it can be analyzed, reported, backed up, etc. using the more extensive resources of the multi-user system.

The virtual terminal capability provides a logical connection between the RSX-11M terminal driver and the console terminal of the Satellite processor, making it possible to perform program development on the Host system without having to physically disconnect the terminal from the Satellite. This facility becomes useful, for example, when the RT-11 user wishes to initiate an RSX task to perform analysis on data which has been filed on the Host via RDCL, or to perform the preliminary stages of RT-11 program development (iterations of edit and compile for syntax) on the RSX system. At installations which also run RT-11 as a task under RSX (available from the DECUS RT-11 SIG), all stages of RT-11 program development can be accomplished in virtual terminal mode (or indeed at the Host site itself); the final version can then be copied to the Satellite using RDCL file transfer mode.

The RDCL system should not be regarded as a competitor with DECNET/RT. Many of the DECNET facilities (e.g. task-to-task communication) are not supported. The motivation behind RDCL was primarily to achieve a highly-specific facility which would be functionally transparent to the RT-11 user, and which would neither add a great deal of overhead nor require extensive documentation changes at the RT-11 end. In this respect, RDCL may be considered as a viable alternative for those users who either do not require the extensive functionality of DECNET or cannot meet its additional requirements.

DESIGN OVERVIEW

The final stage in all RT-11 filing operations is always a request queued to the device handler to read or write a given number of words, starting at a given logical block of the device. This device independent approach to I/O makes it possible to easily add device drivers to the RT-11 system without changing the structure of the other operating system components.

The primary component of the RDCL system is an RT-11 device driver (RD) which performs I/O to a communication link, rather than to a UNIBUS device. The parameters of all filing requests received by the RD driver are translated into control-packets which are then transmitted over the communication link to a task running on the Host processor. The Host task, RDH, performs the actual device I/O using RSX-11M block I/O directives, and transmits the data so read (in the case of a READ operation) back to the Satellite over the link in the form of data-packets.

For example, consider the initial request queued to the device driver each time RT-11 programs attempt to read a file from disk. At the driver level, a request is received to read logical block 6 (the directory) of the device and deposit the data stored in that block into a core buffer. When the read operation is complete, the driver notifies the RT-11 file system (USR) which determines from the directory information in memory whether or not the file is resident on the device. Assume now that the same RT-11 disk is placed in a disk drive on an RSX system rather than on the system running RT-11. (This will henceforth be called a 'foreign-mounted RT-11 disk'.) The request to read block 6 of this device is now placed by the RD driver on a telephone line, rather than on the UNIBUS. At the RSX end, task RDH checks the accuracy and syntax of the message received from RD and, assuming all is in order, reads block 6 of the foreign-mounted volume and sends the resulting data back to RT-11 over the link. The RD driver is then responsible for checking the accuracy of the data message received from RSX and passing the data buffer to the USR. Thereafter, all RT-11 operations will proceed as if the data were read from a local device.

The above is a simplified description of the approach used in the RDCL implementation. Figure 1 shows the information flow diagrammatically.

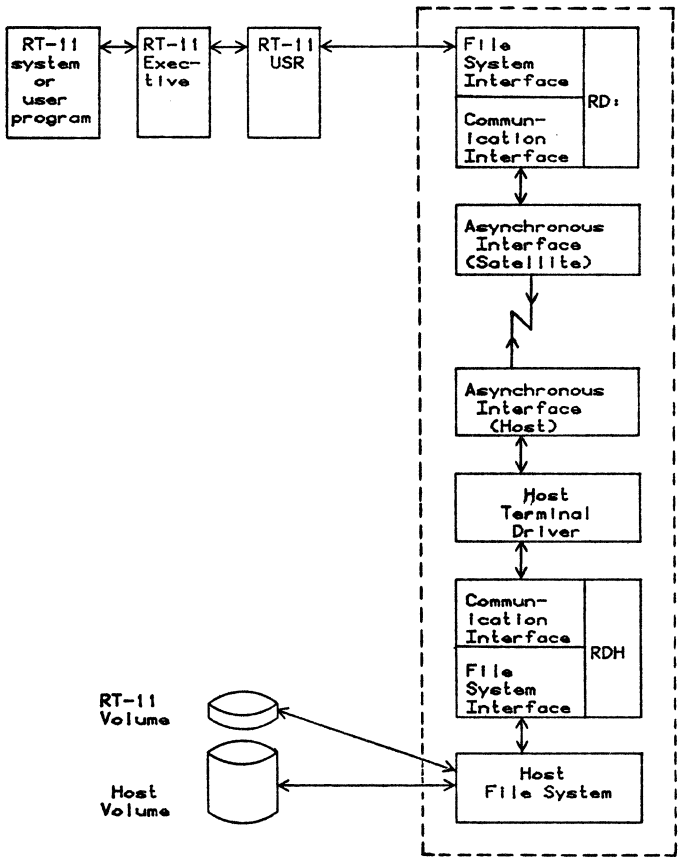


Figure 1

Information Flow in the RDCL System

On the RT-11 system is shown the RDCL driver (RD:) which is responsible for passing USR requests to the RSX task (RDH) running on the Host. The RD/RDH modules, enclosed by dashed lines on the diagram, may be considered together as a completely transparent system device handler by RT-11. All RT-11 operations, including overlaying, running system programs and running user-written programs which perform system level I/O (e.g. FORTRAN 'WRITE') will be functionally transparent although noticeably slower in operation due to the speed restrictions imposed by the communication link. This type of implementation results in a system which is simple to generate, maintain and use, since all the changes are located in a single module, the RD driver. Furthermore, a minimum amount of time is required to educate the RT-11 user. From his point of view, all that has changed is that a 'new device', RD:, has been added to the system, a device which may be accessed in the same fashion as the other mass-storage devices described in the RT-11 manual set.

VIRTUAL DISK

A further aspect of the functionality of the system is related to the implementation of the Host task, RDH. With very minor changes, RDH is able to access a file located on the RSX system disk rather than a foreign-mounted RT-11 volume. This file is organized such that it contains an image of an RT-11 file structured device. Thus, for example, the seventh block within the file contains the same directory information as the seventh block of an RT-11 volume. RDH can be configured such that it will honour an RD request to read the seventh block of a disk by performing a "read-virtual-block" operation on the seventh block of the file.

This feature makes it possible to configure several Satellite processors, each able to perform filing operations to independent files ('virtual disks'), all of which are located on the Host system disk. When the Host system device is large enough, this mode of operation becomes attractive by reason of the fact that it is not necessary to reserve individual disk drives for each of the communicating Satellites. Additionally, the size of each virtual disk can be tailored to provide only as much storage as is required by each Satellite.

VIRTUAL-TERMINAL UTILITY

As mentioned previously, the virtual-terminal utility, 'RDTALK', allows a user at the Satellite to utilize the resources of the Host system in the same fashion as if he were sitting at a terminal directly connected to the Host.

Among other operations, RDTALK acts as the mechanism for establishing logical connection and disconnection of the RD driver to the RDH task. Initially, RDTALK establishes the Satellite terminal as a virtual terminal to the Host. The RT-11 user then is able to log onto the RSX system and initiate the running of the RDH task which, as noted above, is responsive to several control-type messages which emanate from the Satellite. If RDCL is being used as a complement to local storage on the Satellite, no further operations are required; i.e. RDH will always act upon READ/WRITE control messages transmitted by RD. However, using RDTALK, the Satellite user may enter other messages directly from the Satellite console which will be recognized by RDH. For example, to down-line-load a version of RT-11, the Satellite user types the characters 'BOO' on the console. Other possible control messages are 'DSC' (disconnect) and 'RPT' (report).

In configurations which support local mass storage at the Satellite, 'RDTALK' is run from the local device. Down-line-loading of RT-11 on a core-only machine is accomplished by executing RDTALK from read-only-memory (ROM). Preliminary versions of this special ROM bootstrap have already been built for UNIBUS PDP-11 systems; the design of LSI-11 versions is in progress.

MODIFIED RT-11 MONITOR

The version of RT-11 which is down-line-loaded differs in two ways from the DEC-distributed RT-11 operating system:

1) The DEC distribution of RT-11 includes several monitors, all of which are basically the same except for the system device driver which has been linked as part of the monitor. Thus, for example, RKMNSJ.SYS differs from RXMNSJ.SYS only to the extent that the RK driver is linked in the former case and the RX driver in the latter. In the case of the RDCL version of RT-11, the RD driver is linked to the executive, thereby allowing system device I/O requests to default to the remote device.

2) The first part of every RT-11 monitor file consists of bootstrap code which, when initiated, reads the remainder of the monitor into memory from the system device. The RDCL monitor is linked to a modified bootstrap which allows the monitor to be read from the remote device.

Thus, to create the RDCL version of RT-11, it is necessary to have available the source modules of the RT-11 executive and

link these to the bootstrap and driver modules provided as a part of the RDCL distribution.

COMMUNICATIONS PROTOCOL

Host-Satellite synchronization is accomplished using a simple ACK-NAK communication protocol with asynchronous clocking for determining time-outs. Error checking on data packets is implemented using a Cyclical Redundancy Check algorithm (CRC-16). When errors are detected, retries on the data packet are made. When the number of retries exceeds the RDCL defined limit, the operation is aborted -- the driver at the Satellite takes a "device-error" exit to the monitor, while the Host task returns to the "wait state", i.e. wait for a control message. The protocol has intentionally been kept simple in order to minimize system overhead.

USER INTERFACE

The user at the satellite interacts with the system as he would with RT-11,

```
.R RDTALK                              <Initiate
                                         conversation
                                           with
RDTALK -- X01.1                              Host>
Enter Virtual Terminal Mode

>HEL RDRS                              <log on>
PASSWORD:
        RSX-11M BL21    MULTI-USER SYSTEM
GOOD AFTERNOON
30-JAN-78 12:19 LOGGED ON TERMINAL TT10:

 .                                     <any
 .                                       RSX
 .                                         command>

>RUN RDH                               <init communications>
?                                      <RDH prompt>
RDTALK -- Exit                         <<^P><CR> causes
                                         return to
.R PIP                                     RT-11>
*RD:/E
30-JAN-78
RKRDMN.SYS    52
PIP    .SAV    14
 .
 .
*^C

.R FORTRA
*RD:TEST,TEST=RD:TEST
*^C

.R LINK
*RD:TEST=RD:TEST,SY:FORLIB
*^C

.RUN RD:TEST
 .
 .
STOP --

.R RDTALK                              <talk
                                         directly
                                           to Host
RDTALK -- X01.1                              again>
Enter Virtual Terminal Mode

?DSC                                   <request the Host
TT10  --  STOP                           comm. task to stop>
>BYE                                   <log off>
HAVE A GOOD AFTERNOON
30-JAN-78 12:25 TT10: LOGGED OFF
>
RDTALK -- Exit                         <<^P><CR> causes
                                         return to
.                                          RT-11>
```

FIGURE 2. - SAMPLE TERMINAL SESSION

with the additional steps of connecting to and disconnecting from the Host. Figure 2 is a sample terminal session which illustrates the typical interaction possible from an RT-11 system with local mass storage.

When RT-11 is bootstrapped from a remote-device, explicit references to device RD: are unnecessary; i.e. RT-11 defaults to SY:, which invokes the RD driver linked into the RDCL version of RT-11.

Similarly, user programming is transparent. For example, a portion of a program which will file data into a random access file located on the remote device would be:

```
      .
      .
   CALL ASSIGN(1,'RD:MYFILE.DAT')
   DEFINE FILE 1 (LREC,NREC,U,IVAR)
   WRITE (1'1)IOLIST
      .
      .
```

HOST TASK TRANSPORTABILITY

The Host task has been coded in a high-level structured language, 'Structured Fortran' (SF). The SF translator produces standard ANSI FORTRAN source code which can then, in turn, be compiled by most FORTRAN-IV compilers with little or no modification. There is some RSX-11M system-specific code contained in RDH, but an attempt was made to modularize the design sufficiently so that this code could be easily identified and changed.

The possibility exists for implementing the Host task on systems other than RSX-11M (e.g. RSX-11D, TOPS-10, etc). The operating system characteristics which would be necessary are:

1) The equivalent of the RSX-11M terminal driver directive to read and pass-all-bits to the task with no echo;

2) The ability for the filing system to perform logical and/or virtual block I/O on mass storage devices;

3) Asynchronous timer support.

HARDWARE REQUIREMENTS

a) Satellite

- PDP-11 (or LSI-11) processor capable of running RT-11 ;
- clock to be used by RD: for timeout control ;
- asynchronous serial interface (e.g. DL-11W) to be used for communication to the Host ;

therefore achieve maximum speed,

(4) the interdependence of the send and receive activities. The nature of received messages affects the nature of transmitted ones,

(5) the existence of a priority-ordered list of send messages.

Factors (1) and (2) suggest that the send and receive activities be interrupt driven. At the same time, factor (4) means that there are critical sections of code where shared data regions are accessed. Unfortunately, RT-11 does not provide any special mechanism such as semaphores for controlling access to shared data. An alternative is the use of interrupt lockout during critical sections. This is acceptable as long as the duration of the interrupt lockout is less than the minimum time span between interrupts. Access to shared data has been minimized as much as possible. The only shared data structures that remain are the pointers and counters for six queues and two sets of flags associated with reset and negative acknowledgement conditions; all of these are accessed for very short periods of time.

Factors (3) and (5) required a straightforward mechanism for scanning the list of send message flags and initiating the transmission of the highest priority message. An approach involving the use of a clock-driven scan of this send list is conceptually clear - it avoids multiple layers of send interrupt processing - and involves little overhead. Further, the scan interval can be easily adjusted to fit the transmission time of the shortest possible message. A clock is also needed to provide retransmission and link failure timeout conditions. The first of these timeouts occurs if transmitted messages have not been acknowledged for a certain length of time. This timeout is a function of maximum message length and link speed. The second timeout occurs after a fixed interval if no messages are received on the link. In this situation, the current state of the link is stored in a file and an exit to the RT-11 monitor is performed.

Message Transmission: The sequence of events associated with the transmission of a message is the following:

(1) after a timeout, the timer activates the scheduler;

(2) the scheduler scans the send list flags. If one is set the flag is cleared and the appropriate message is prepared. Flags are set by the timer, the interface subroutines or the receive side of the processor, according to the current status of the system. Data messages are handled slightly differently. There can be more than one data message waiting at any one time to be transmitted but only one of each type of control message. Therefore, a queue is needed to store waiting data packets. The scheduler detects the presence of a waiting data packet by checking the counter associated with this queue rather than a flag;

(3) the prepared message is passed to the transmit side of the device processor; while the message is being transmitted, the scheduler is inhibited.

From the above discussion, we see that the act of transmitting messages has no effect on the receive side of the protocol processor, apart from disabling interrupts during critical sections. This separation of activity has two advantages; it improves clarity and it minimizes shared data. Total separation is impossible; factor (4) requires that the receive side affect the transmit side of the monitor. However, this effect is limited to the setting of flags and the adjusting of pointers and counters. One example is the adjustment of the retransmit queue upon reception of message acknowledgements. The receive side itself is driven completely from receive interrupts. In fact all the processing of received messages is performed in a nested series of completion routines called from receive interrupt service routine. This approach presents no problem as long as the time to receive the shortest packet is longer than the time to process a packet header (which has a fixed length).

Message Reception: The following sequence of events is associated with message reception:

(1) upon reception of a packet, the header CRC is computed and the packet is discarded if it is bad;

(2) if it is a control message, the processor's state variables are updated and the send list adjusted;

(3) if it is a data packet, the state variables are updated and the data region's CRC computed. If the packet is valid it is passed to the message processor via an interface subroutine. Otherwise, it is discarded.

BUFFER MANIPULATION

The size and number of buffers is the choice of the message processor using DDCMP. This permits maximum flexibility and promotes efficient use of resources. The only size constraint is a maximum data region size of 2^{14}.

The buffer queues used in DDCMP are one-way linked lists. Each list is characterized by a headpointer, tailpointer and counter. Two separate sets of buffers for transmit and receive were used for the following reasons:

(1) Because a steady outflow of data messages is dependent on a steady inflow of acknowledgements, buffers must always be available to the receive side of the device processor. This is most easily achieved by dedicating a set of buffers to reception.

(2) Having a linked list of send buffers, as opposed to a ring buffer, has the advantage of permitting the message processor to return buffers to DDCMP in an order different from that in which they were obtained.

Send buffers: Three sets of buffers are used for transmission:

sendfree queue:	list of buffers available to the message processor for filling.
send queue:	list of buffers containing data messages waiting to be transmitted.
retransmit queue:	list of buffers containing transmitted but unacknowledged data messages.

The send buffers carry only numbered data messages. Control messages are formed when the need arises and are not stored. The flow of buffers is the following: empty buffers are passed to the message processor where they are filled and returned to the send queue. Upon transmission, they are placed on the retransmit queue until an acknowledgement returns them to the sendfree queue.

Receive buffers: The receive buffers are used for the reception of both data and control messages. Three sets of buffers are again used:

receivefree queue:	pool of free buffers.
numberedmessage queue:	list of received data messages waiting for validation.
receive message queue:	list of valid data messages waiting to be passed to the message processor.

In addition, there are three buffers within the reception process itself. They are the 'presentbuffer', 'previousbuffer' and 'nextbuffer'. The presentbuffer is the one currently being filled with incoming characters, the previousbuffer is the one most recently filled and the nextbuffer is the one to be filled next. Because the processing of control messages and data messages with bad headers takes less time than does the reception of the shortest possible message (8 bytes), the buffers containing these messages can be reassigned immediately as the nextbuffer without having to pass through the receivefree queue. This is path (1) in Figure 3. Data messages with good headers and bad data follow path (2), returning directly to the receivefree queue. Valid data messages pass through the receivemessage queue and return via the message processor to the receivefree queue (path (3)).

If the message processor does not introduce any significant bottlenecks in the system, efficient transmission and reception is possible using only six buffers: three for transmit, where one buffer is being filled, the other is being transmitted and the third is waiting for an acknowledgement, and three for receive, where one is being filled, the other is being validated and the third is being emptied.

INTERFACE TO MESSAGE PROCESSOR

The five interface subroutines allow any message processor to utilize the DDCMP processor in a clean straightforward manner. These Fortran-callable routines are used to initiate and terminate activity at the DDCMP level, to obtain and deliver buffers for transmission in an asynchronous manner and to access the contents of received messages.

Initialization: All DDCMP-level counters and flags are initialized by the initialization routine DDINIT; the number and size of buffers is specified and the communication link is started up via the START-STACK sequence.

Sending Messages: To avoid core-to-core transfers, the subroutine GETBLK makes available to the message processor the actual buffer used for transmitting the message. Once the buffer is filled it is returned to DDCMP via SNDBLK.

Receiving Messages: If the same approach were used on the receive side, there would be the possibility that they would not be returned to DDCMP after the message was obtained. To avoid this possibility, a core-to-core transfer is used by RECEIV to move the contents of a received buffer to a core array. The message processor must arrange to call RECEIV on a regular basis to ensure that received messages are picked up. Otherwise, a bottleneck may occur on the receive side of the DDCMP processor.

Termination: The subroutine DDEXIT is called to terminated DDCMP activity and to exit to the RT-11 monitor. The message processor must perform all of its own exit operations before calling DDEXIT.

INTERACTION WITH RT-11 MONITOR

The current implementation of the protocol processor runs under the RT-11 single-job monitor. All interrupt service routines, through the .INTEN programmed request, run in system state. This greatly reduces available stack space, so subroutine calls internal to DDCMP pass all parameters in registers. Under the SJ monitor, the 60HZ clock interrupt serves no practical purpose, so it is rerouted to provide the needed DDCMP timer facilities. If the F/B monitor were used, the DDCMP timer could be made to run as a F/B clock initiated completion routine.

XFR - A FILE TRANSFER UTILITY PROGRAM

One current application of the DDCMP communications module is the full-duplex transfer of ASCII and binary files between processors. XFR is a Fortran program which handles all the higher-level duties associated with this task. Among its activities are the following:

(i) interact with the user to determine the sequence of actions to be performed. The user must specify the type of file and the direction of transfer. Only one file is eligible at any given time for transmission in a particular direction.

(ii) perform all I/O to storage devices; because ASCII files are record oriented while binary files are block oriented, the two types are treated differently.

(iii) break down files into packets and vice versa and transport the packets to and from DDCMP.

(iv) provide the higher-level message protocol for interaction with the other processor. This protocol is needed to initiate and terminate transfers and to report errors

(v) maintain a log of link activity over extended periods of time. XFR shares with DDCMP a common block of variables which reflect the number and types of messages which are sent across the link. Upon exit, DDCMP stores this information in a file; XFR stores the same information in a second file with the difference that the second file contains a running account of all activity since the file was created; the DDCMP created file stores only the latest link usage information. In this way, both current and long-term link activity can be monitored.

CONCLUSION

An assembly-language module implementing a point-to-point full duplex version of DDCMP and running under the RT-11 SJ monitor has been developed. This module, including a line driver for a DL-11 serial interface, is less than 1.8K words in length, excluding buffers. The module has been designed to be general in nature, with a clean straightforward interface to any higher-level module using it. This permits its use in a wide range of applications. One such use is the transfer of files between processors.

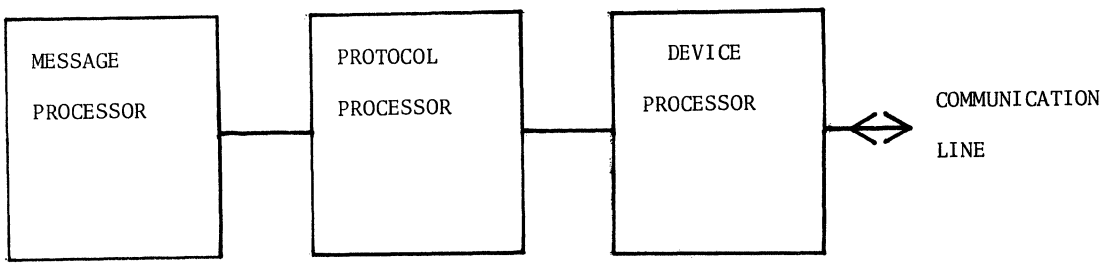


Figure 1. LEVELS OF PROCESSOR NEEDED FOR COMMUNICATION

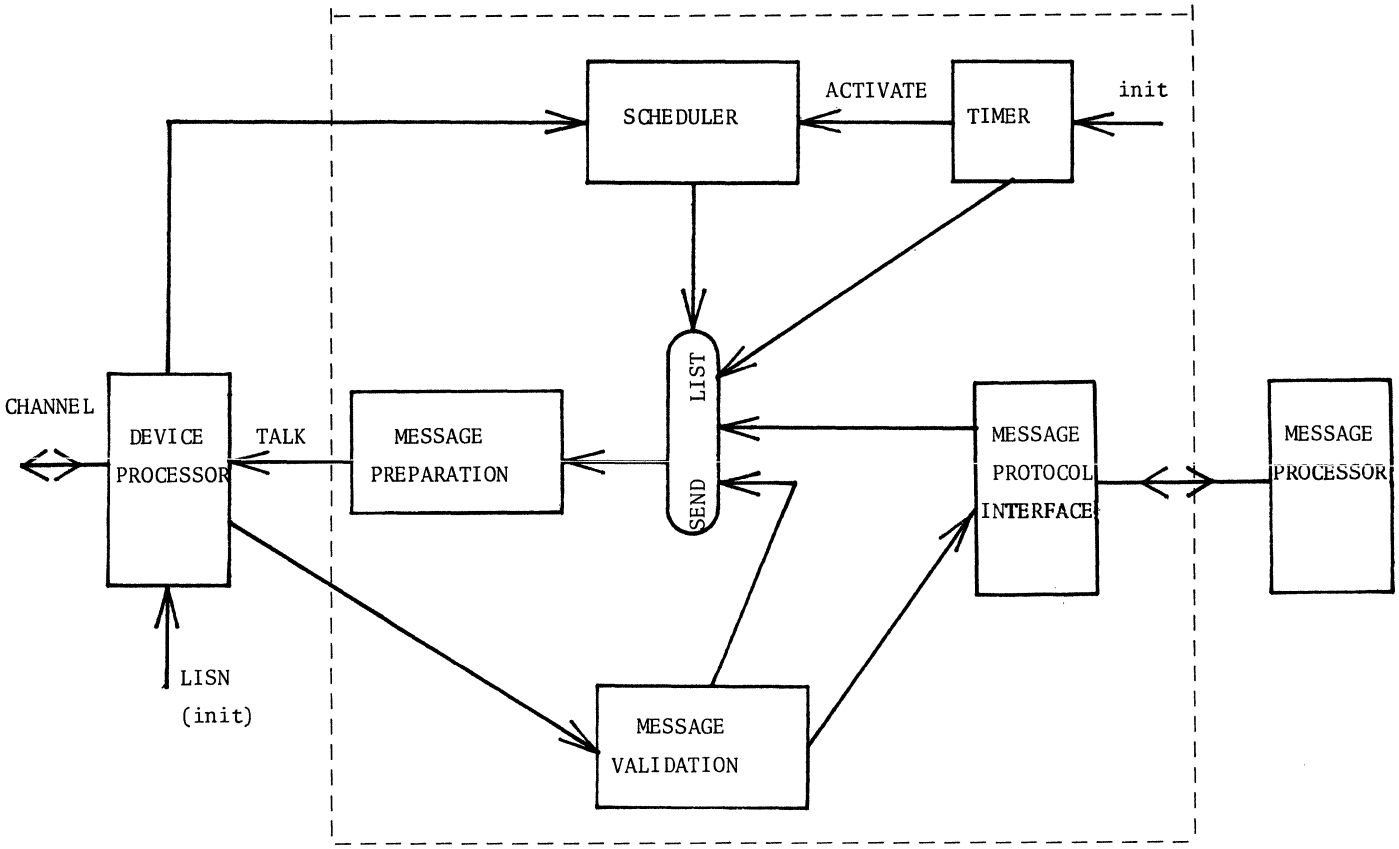


Figure 2. FUNCTIONAL DESCRIPTION OF DDCMP PROCESSOR

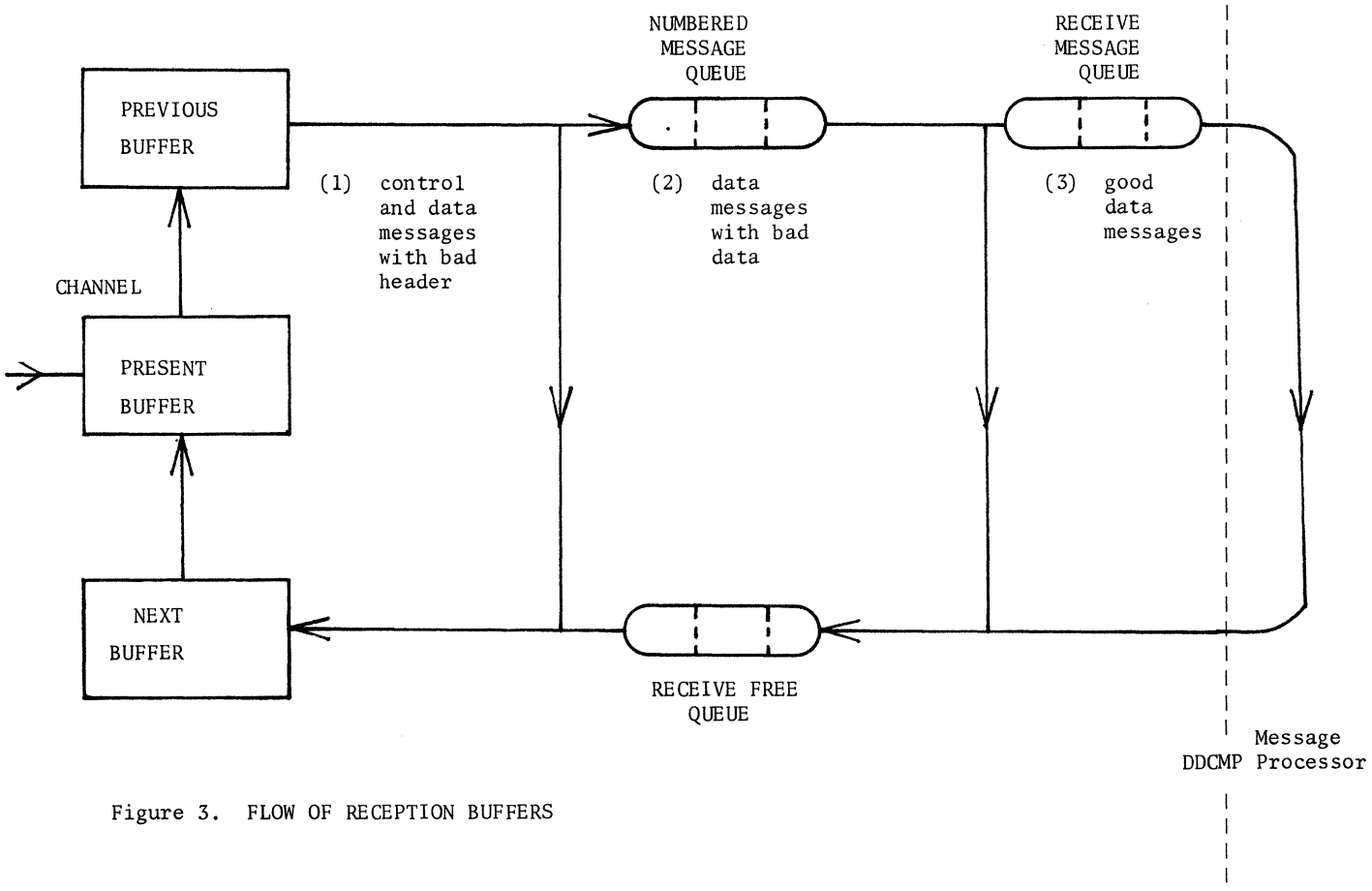


Figure 3. FLOW OF RECEPTION BUFFERS

A MULTI-DETECTOR PULSE-HEIGHT ANALYSIS SYSTEM

Chris P. J. Kelly, Damon Stafford and Alfred J. Hulbert
Inhalation Toxicology Research Institute
Lovelace Biomedical and Environmental Research Institute
P. O. Box 5890
Albuquerque, New Mexico 87115

ABSTRACT

Investigators at Lovelace ITRI are currently using a unified Multi-Detector Pulse Height Analysis (PHA) system which permits the same software package on a PDP-11/T34 to be used for analysis of nuclear counting spectra collected from a wide variety of detectors. The analysis software described includes interactive graphics on storage scope terminals and specialized spectral analysis routines tailored to individual detectors or experiments. The detectors include Germanium-Lithium high resolution gamma detectors, Silicon-Lithium alpha particle detectors, Phoswich (Sodium Iodide-Cesium Iodide) X-Ray detectors, beta scintillation and large Sodium Iodide counters. Applications include neutron activation analysis, analytical radiochemistry, personnel monitoring and analysis of biological specimens from radiobiological studies. A companion PHA package on the ITRI central computer, a PDP-11/70, is functionally equivalent at most levels, with identical user commands, but with enhanced capability for more extensive analysis and hardcopy graphics.

INTRODUCTION

At the Inhalation Toxicology Research Institute (ITRI), pulse height analysis (PHA) has long been an important research tool. As real time computing has expanded, researchers have asked for more computer assistance in the collection and analysis of PHA spectra and this is the system designed to meet those needs.

DESCRIPTION OF PROBLEM

While designing the PHA software, it became clear that the various users of the system would have different needs in counting and analysis, which depended upon many factors. First, the emission characteristics of the radioisotopes (alpha, beta, gamma and X-ray emitters were all in use). Second, the type of application (e.g., personnel monitoring, analytical chemistry and emissions characteristics). Finally, what was to be done with results (observed, discarded or saved for use in Information Storage and Retrieval systems).

The program had to meet these needs, and provide for future expansion and adaptation to other computers. Two PDP-11/T34 processors were scheduled for use in collection and analysis of pulse height data. All detectors had to be accessible by any user, collection and analysis had to be independent processes, and some specialized analysis and output routines needed restrictions on use. The file structure and access scheme provided a key to our solution, and a modular program structure completed the answer.

This flexible system allows each detector to run independently of others, and since some detectors have very long counting times (ten or more hours), this means many simultaneously active detectors. We found that the costs of that many stand-alone systems, and the problems of continually setting up multichannel analyzers for different collection arrangement (and the errors created thereby) were prohibitive, and justified the development effort.

HARDWARE ARRANGEMENT

The data collection is performed by a CAMAC-oriented data acquisition system, connected to a PDP-11/T34 or 11/45 mini-computer. Figure 1 is a schematic of the system, with detectors on the left. Each detector provides its own power supplies, preamps and amplifiers, and puts out pulses between 0 and 8.192 volts in amplitude. These signals are sent to a multiplexer which buffers input to a high-speed analog-to-digital converter (ADC). Tag bits added by the multiplexer note from which detector the pulse originated, and digital information is sent to a CAMAC input module. The CAMAC dataway also contains a multiple clock module which collects elapsed and live time data. The Camac Branch control is handled by a Microprogrammed Branch Driver (MBD-11), a fast microprocessor which resides on the PDP-11 Unibus, and processes data for DMA transfer to the PDP-11 memory.

Control programs in the PDP-11 allocate buffers in core, one for each active detector, and upon termination of a collection, write the data and associated header to a disk file.

FILE STRUCTURE AND ACCESS

In order to allow all users access to the recently collected data files, a special user code was created which gave all users ('world' in the DEC terminology) full access privledges, and stored all data and most common libraries in that user file dictionary. The collection program puts out completed spectra into a master data file (Figure 2) onto which the detectors are logically mapped by a definition file. This definition file also serves as a symbolic mapping, as each detector is given a unique six-character name by

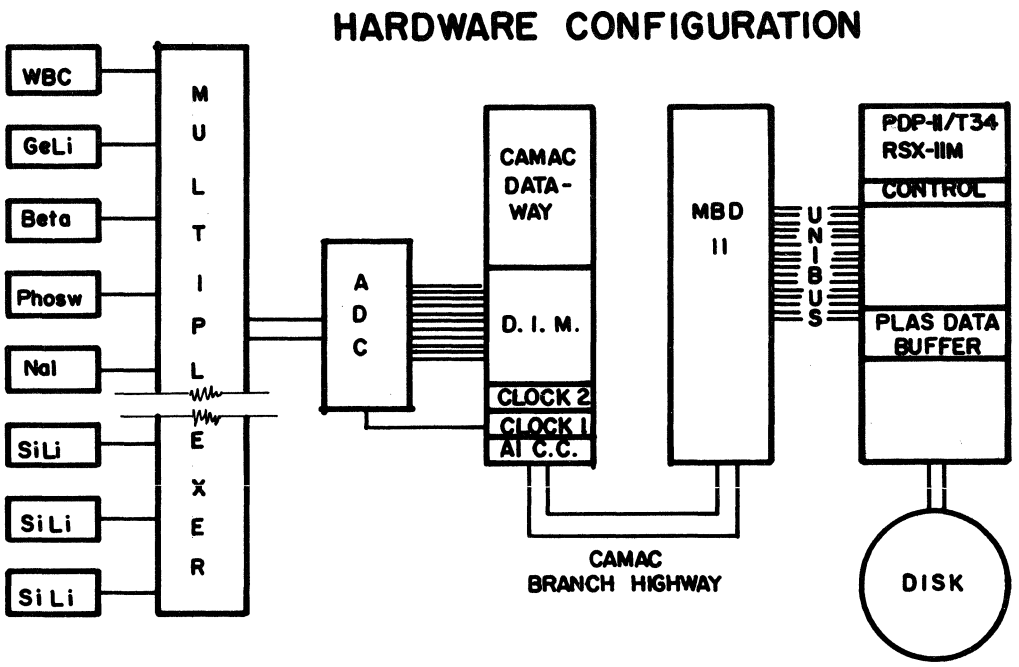


Figure 1

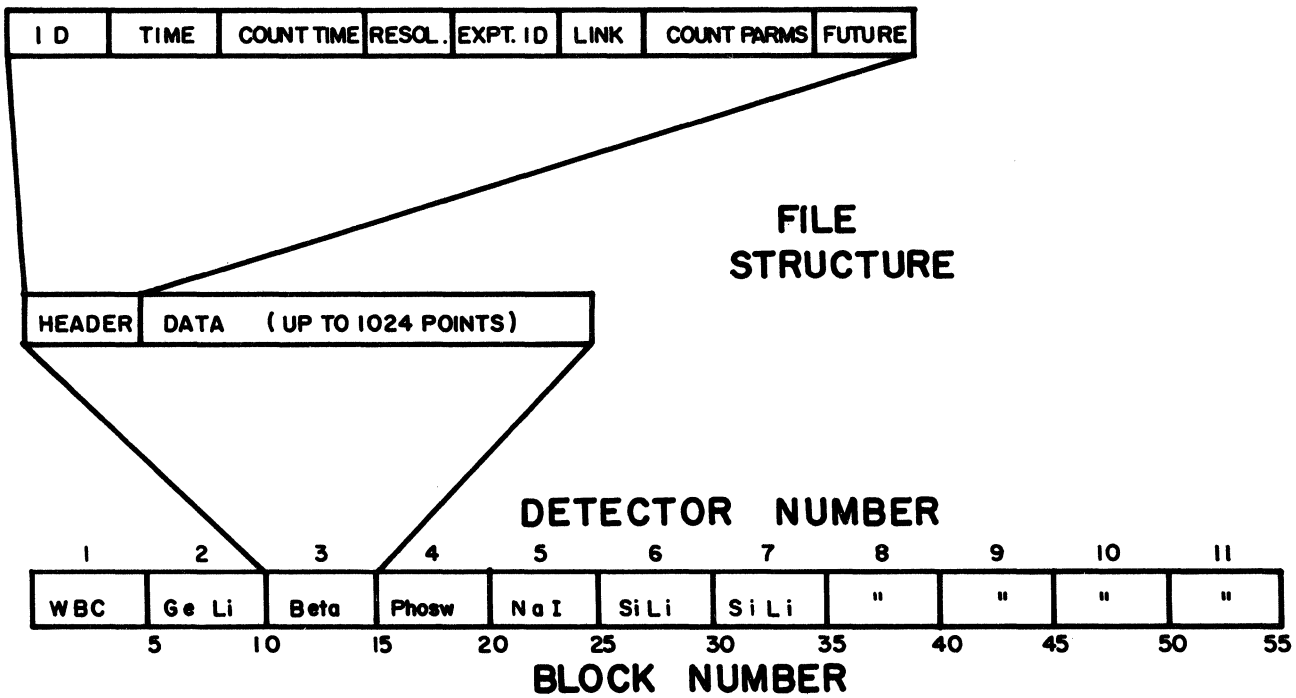


Figure 2

which the user can refer to that detector. The data file is a direct access file 240 blocks long, mapped into a maximum of 48 detectors (five disk blocks per detector). All detectors have the same structure to their files, with the first block reserved for a file description header, and the remainder for data, up to 1024 channels of integers.

The header of the file contains the following information:

1. Detector ID - six-character symbolic name of detector.

2. Detector Number - order in which detector data are mapped into the most recent data file.

3. Time of Day - date and time spectrum was collected.

4. Counting Time - livetime and elapsed time (and therefore deadtime) of collection, and time for use in calculations after deadtime corrections are performed.

5. Resolution Information - energy (KeV) corresponding to channel one and to the highest channel in the spectrum.

6. Number of Channels in this count.

7. Experiment ID - user entered identification of data.

8. Unique ID - for sorting of files.

9. Clock Frequency - of collection control clock.

10. Deadtime Correction Indicator.

11. Background Subtraction Indicator.

12. Linkage Indicator for multi-detector counts.

13. Collection Termination Code (to indicate type of termination, e.g., overflow, timeout).

14. Reserved space for future expansion.

The file is read by calls to in-house multiple-block I/O subroutines which perform a direct access read without the use of large intermediate buffers, conserving memory and speeding execution of I/O.

This arrangement is convenient because the latest data are always found in the same file, and the user is able to access any of the detectors simply by knowing the detector ID name. Only minimal instruction in the operating system of the computer and the PHA program is required. The file system additionally hinders storage of worthless or poorly defined data files (forgotten and wasting the limited mass storage) by writing the most recently collected data on top of older data for the same detector. This forces users to make use of data before collecting additional spectra, a benefit to the system and the users. A user collecting a series of 10 minute counts is able to start a new collection before analyzing the old, since it is only upon termination that older data are erased, and older data are read by the analysis program keeping a complete copy in core during analysis. This overlap of collection and analysis greatly speeds the counting process when many samples are involved.

GENERAL ANALYSIS

The program is interactive with the user, with any or all options selected at will and in any order. This adds to the flexibility of analysis, since any feature may be repeated or dropped as necessary.

Upon startup, the program asks what detector the user is interested in, and calls the mapping subroutine to error check and to set pointers to the detector requested. Data are read into common arrays, and the input file closed. At this point, a subroutine begins to analyze header information, and a summary is printed on the user terminal and on a logging file on disk. Since analysis is usually performed on graphics terminals, the logging file provides a semi-permanent storage of analysis results in addition to an audit trail of functions performed during analysis.

A header analysis subroutine performs deadtime correction based upon data collected by two clocks (live time and elapsed time) at count time, and notifies the user of the magnitude of deadtime and the new sum of the data.

This deadtime correction is performed based upon the termination code, because the integer data type places restrictions on maximum counts per channel, and collection programs will terminate a collection if data value in any channel approaches this value. In this case, the live time of the count will be used with no change in data, while in other cases data will be scaled to match requested count time.

After this brief analysis, control is passed back to the main routine, which asks the user for a two-letter command, which will determine the next function to be implemented. Options include the following:

1. AA - Alpha Radiochemistry Special Analysis
2. AB - Beta Spectrometer Special Analysis
3. AG - Ge(Li) High Resolution Analysis
4. AN - Human Whole-Body Counter Special Analysis
5. BG - Generation of background library
6. BS - Subtract latest background from data
7. EX - Exit from analysis with line printer list of analysis
8. GR - Interactive Graphics
9. LG - Update whole-body counting libraries
10. RC - Semi-automatic recalibration of scales
11. SA - Save current data in disk file
12. TC - Convert data to counts per minute
13. VP - Hardcopy Graphics

Each of these functions have interesting features, and because of the modular design of the program, more may be added at any time with ease. Here the discussion will center on a few of the more interesting functions.

Alpha Radiochemistry Analysis

This function is tailored to the needs of highvolume radioanalytical chemistry of low-level alpha emitters, and is a front-end to an Information Storage and Retrieval system. In this sense, subroutines will do some minor analysis and calculations, and then reformat the data for entry into the ISR system.

When the user calls this function, the program requests names of materials expected in the sample, and reads a library containing some characteristics of the materials. Using this library information, the spectrum is examined and peaks in expected areas are counted and checked for shape and actual energy correspondence. The program then calculates actual activity in these samples using count time, and stores the information together with ID data in a free-field data file which is automatically named for the Julian date and fraction of day on which the spectrum was collected.

The user needs only to transfer this file to the central computer for use in the ISR system. Throughout the PHA program, extensive error checking of input data is performed to ensure accuracy in results.

Whole Body Counting Analysis

The Human Whole-Body Counter is used for personnel monitoring, and periodic checks are performed to ensure safety of laboratory personnel. Sodium iodide detectors are sensitive medium-resolution devices, and are used to produce a 512 channel spectrum in a standard 10 minute counting time. A library containing spectra of the radioisotopes used at ITRI is stored on disk, having been counted on the whole-body counter using standard sources.

When a spectrum has been collected, the user calls the WBC analysis function, which requests names of the library spectra he wishes to compare with the sample. Five spectra are used in any one run. These library spectra have unique ID names by which the user refers to them.

After reading all library spectra into an array, a spectrum fitting matrix inversion routine fits the libraries to the sample using a least squares technique and prints a list of suspected quantitites of each library radionuclide.

Interactive Graphics

A useful and widely used feature is the interactive graphics mode of analysis, as it provides a flexible and powerful means of computer assisted examination of pulse height spectra.

Using 17 single-letter commands, the user is able to perform a relatively exhaustive examination of the data, leaving only the most complex and specialized functions to other subroutines. Functions implemented are as follows:

1. A - Integrate user designated area of display.
2. B - Rescale to previous window.
3. C - Connect data points with a solid line.
4. D - Display computer generated best fit spectrum (generated by the Whole-Body counter Analysis routines).
5. E - Identify point at crosshairs.
6. G - Display spectrum with logarithmic Y axis.
7. H,L - High and low limits on new window (Zoom). Allows magnified view of data.
8. M - Multiply all data by user input number.
9. O - Return to original window (display all data).
10. R - Replot in same window.
11. S - Smooth data (five-point least squares parabolic fit).
12. T - Integrate entire spectrum.
13. X - Exit from Graphics.
14. Z - Set point at crosshairs to value of zero.
15. Errors - Bell and question mark appear.

Interactive Graphics is performed on storage screen terminals with crosshairs, using in-house plotting subroutines for control of the special graphics mode. Since all data generated by the user in graphics mode is erased from the screen upon exit, all functions called are recorded on the disk logging file which is spooled on exit from the PHA program.

Hardcopy Graphics

An option is available to create a paper graph of data, using either a high resolution electrostatic printer/plotter or the impact matrix line printer/low resolution plotter. The user requests hardcopy graphics and the routine asks for a plot title, and calls in subroutines to generate intermediate binary files. Functions diverge now depending upon which computer is used. On data collection computers (PDP-11/T34) the pass-two plot generation is accomplished by a slave task, with output appearing immediately on the impact matrix printer/plotter. On the central computer, the binary file is saved for use by the pass-two program at a later time, with output on the electrostatic plotter under control of the computer operator.

SUMMARY

This Pulse Height Analysis system, designed for a multi-detector, multi-user environment has been in use now for approximately one year and has met the

primary needs of users for more rapid and complete analysis of spectra. Additionally, it has provided the starting point for more specialized techniques, having removed from the experimenter many of the burdens of manual data analysis. Flexibility of design makes the system easily adaptable to changes or additions of detectors and specialized analysis functions.

ACKNOWLEDGEMENTS

Work performed under U. S. Department of Energy Contract Number EY-76-C-04-1013.

The authors thank Drs. R. O. McClellan, M. B. Snipes, S. H. Weissman and J. A. Mewhinney and Mr. F. Barr for reviewing this manuscript.

We regret the announce the recent death of Mr. A. J. Hulbert.

A MULTI-USER, MULTI-DETECTOR PULSE HEIGHT ANALYSIS/GAMMA CAMERA DATA COLLECTION SYSTEM USING CAMAC AND PLAS

Damon Stafford, Chris P. J. Kelly and A. J. Hulbert
Inhalation Toxicology Research Institute
Lovelace Biomedical and Environmental Research Institute
P. O. Box 5890
Albuquerque, New Mexico 87115

ABSTRACT

A system which collects data independently from many pulse height analysis detectors is described. Connection to the computer is via either parallel or serial CAMAC branches and Microprogrammed Branch Drivers (MBD 11). Computers are either PDP-11/T34's or a PDP-11/45 with RSX-11M Ver. 3.0 as the operating system. Some gamma camera collection functions are implemented. These are being expanded. Users run a simple startup task from any terminal to provide required parameters. A central monitor task (CONTRL) then error checks the request and sets up PLAS buffer areas. A third task (COLLEC) communicates the request to the MBD which actively controls data collection. The only task which must be core resident during collection is COLLEC which is relatively small in size.

REQUIREMENT

Nuclear data collection at the Inhalation Toxicology Research Institute (ITRI) has been an integral part of our research support technology for many years. Control of collection and entry of data into the computer was, until recently, very cumbersome and inefficient. In some cases no computer processing of data was done because of the time involved in data entry. New research projects required computer processing simply because of the volume of data to be collected. Many more individuals with diverse backgrounds would be involved in data collection so it was desirable to simplify the collection process as much as possible. The decision was made to automate the collection process.

Pulse height analysis (PHA) includes quantitating the number of radioactive emissions from a substance over a wide energy range. Our requirements for PHA vary significantly with the specific project. One project, low level alpha radiochemistry, will have 32 collection detectors on-line and possibly simultaneously active. Storage requirements for projects vary from one 128 word buffer to two 16K word buffers per detector. Collection times vary from a few seconds to many hours. Count rates vary from a few counts per hour to 50,000 counts per second. Gamma camera collection is primarily a PHA in a three coordinate system. For most applications the collection problem is the same as the two coordinate system.

CAMAC has become the ITRI standard for data collection. Parallel branches are planned to most areas which require PHA collection services. A serial branch will serve other areas which are low count rate projects. Currently eleven detectors are in operation on one parallel branch. The gamma camera is presently in the testing stage. The serial highway will be operational soon.

Individual stand-alone systems were rejected for most applications because of the large number of systems required. A data interface would still have to be developed for transferring the data to the primary analysis computers.

SYSTEM OVERVIEW

Capabilities

A researcher can start a collection from any terminal connected to the same computer as the detector. It can be scheduled to run for a specific time duration. It will stop automatically if the number of counts in an energy range exceeds 32000. The researcher can at any time stop collection and save the accumulated data.

Actual elapsed time may be slightly different (less than a second) than the time requested by the researcher. A free running clock times each collection so that elapsed time is known accurately. Analysis programs use this to compensate for deviations. Resolving time of the ADC is usually more significant. It is monitored and is then compensated for by analysis routines.

Usage

Collection of data from any detector proceeds as follows. The sample is placed in the detector. The researcher goes to any terminal connected to the same computer to which the detector is connected. A collection startup program is run. OMNI is the standard startup routine which works for all detectors except gamma camera (Figure 1). Some projects have startup tasks asking only one question. The computer will return a message to the user's terminal saying that collection was started (assuming there were no errors) and a collection termination message when collection is finished. The sample may then be removed from the detector and the process repeated for another sample. All collection system events are also logged to the console terminal.

Because disk storage space on the data collection computers is limited, the system requires that

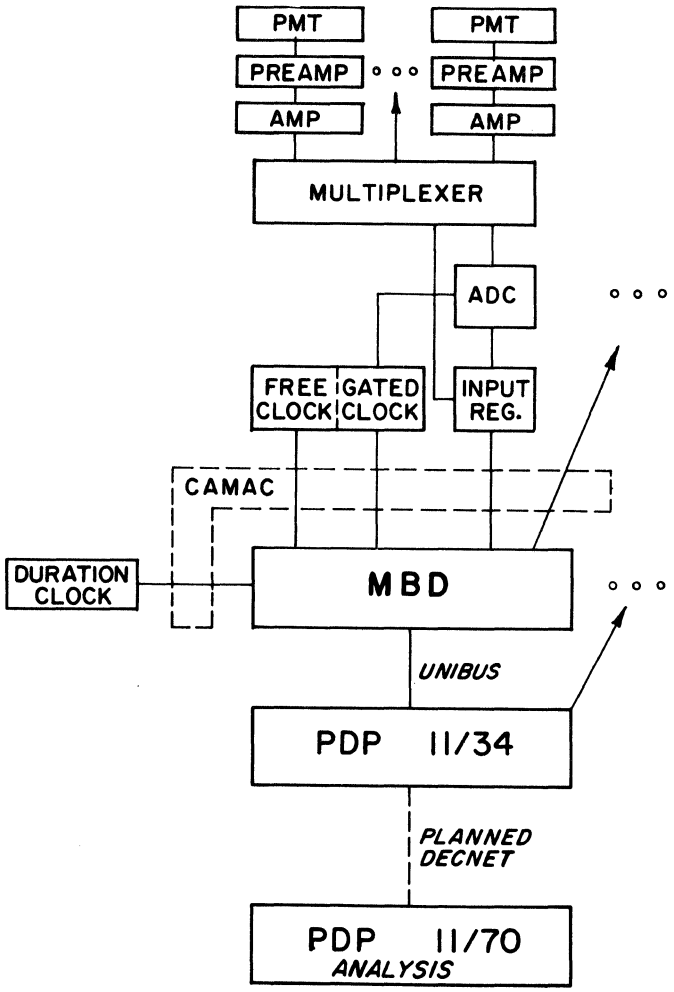
```
 RUN $OMNI
OMNI:  24-APR-78 07:26:53
OMNI:  ENTER THE DETECTOR ID>JELLY1
OMNI:  ENTER THE DURATION (HRS,MIN,SEC)>0,1,0
OMNI:  ENTER X, WHERE 2 XXX IS THE NUMBER OF CHANNELS>12
OMNI:  ENTER COMMENTS>THIS IS THE BIG DEMO
>
CONTRL:  24-APR-78 07:27:51 JELLY1 Data collection started
CONTRL:  24-APR-78 07:28:55 JELLY1 Timeout collection finish
```

Figure 1. Collection startup with system replies.

researchers specifically make a permanent copy of the data if it is desired to save it. Otherwise successive collections on the same detector will destroy the current data. An analysis program is available, which has graphics capability, to aid in that decision. That same analysis program will make a permanent copy of the data if required. The data can then be moved to the central computer, a PDP 11/70, for extensive analysis.

Electronics (Figure 2)

The heart of the system is one of two PDP 11/T34 computers or a PDP 11/45. Each system has several terminals available to users. Each PDP 11 has at least one Microprogrammed Branch Driver (MBD). The MBD is a fast (350 nanosecond instruction time) 16 bit 4K word microprocessor. It is programmable and has eight separate register sets. The MBD interfaces a CAMAC parallel branch highway containing one to seven crates to the PDP 11. It performs all realtime control functions. Other non-PHA realtime functions may be active simultaneously with PHA. Detectors, consisting of a photo-multiplier tube (PMT), a preamp and an amp, send information through a multiplexor (which can handle up to 16 detectors) to an analog to digital converter (ADC). The voltage level from the detector is proportional to the energy of the photon striking the scintillation material in front of the PMT. This is converted to a digitized value from zero to a maximum set on the front panel of the ADC. This number and the multiplexor channel number are put in an input register in the CAMAC crate. A CAMAC LAM (look at me) occurs and the MBD will begin its processing. A dual timer module is supplied with each input register. One timer measures elapsed time. The other measures time the ADC is unavailable for new conversions.



PHA COLLECTION ELECTRONICS

Figure 2

Software

All ITRI's realtime systems use the RSX-11M operating system. Included in each system are two ITRI written drivers which synchronize actions of the MBD with the PDP 11. Facilities are available whereby PDP 11 user programs can load MBD memory with programs or data, start execution of MBD programs and receive asynchronous system traps (AST's) for MBD caused interrupts to the PDP 11. To make more efficient usage of memory and to make the logic of individual programs simpler, a multi-task collection system was designed. All PDP 11 code is written in Fortran, the only higher level language available. The MBD code was written utilizing a set of assembler macros.

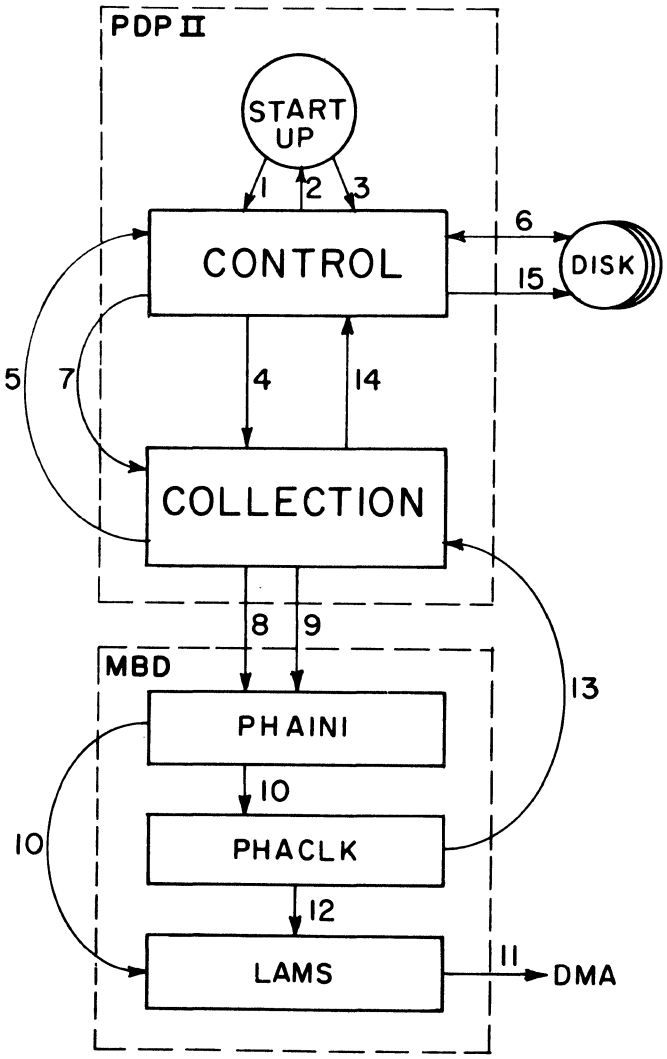
The central monitor and resource allocator is CONTRL. It is always active but checkpointable. No operations with critical response times are performed in CONTRL. It receives input requests from other tasks and finishes each particular function before processing the next request. COLLEC interfaces (at a user level) the PDP 11 collection tasks with the MBD. It initially loads the MBD with its programs. It awakens the MBD when new collections are to be started and receives information about collections that have finished. It is non-checkpointable. Nothing is dependent upon response time.

The investigator communicates with the system through any one of several startup tasks. These tasks ask the appropriate questions of the investigator's specific needs. They are small and easily written. All use the same subroutines for communicating with the collection system. All communicate directly with CONTRL.

Programs that reside in the MBD control all data collections. These programs have quick response time requirements. The lowest priority program controls communication with the PDP 11 concerning new collection starts and system shutdown. The middle priority program controls actual collection timing. It informs the PDP 11 when a collection has finished. Any one of a series of high priority programs process incoming data ready LAM's from detectors. From the time a LAM occurs until the MBD is available to process a new event, roughly 20 microseconds elapse.

System Flow

Parenthesized numbers in the following text refer to labels in Figure 3. To start a collection an investigator runs a startup task. All required questions are asked and the answers error checked before any interaction with the rest of the system occurs. The maximum information required of the investigator is the detector name, collection duration, buffer size, and a comments field for sample identification. The startup task then sends a message to CONTRL requesting buffer space (1). If the detector is not already in use, CONTRL will allocate a buffer in a PLAS region. A "send by reference" will be executed so that the startup task can attach itself to the buffer region (2). The startup task zeroes the buffer and fills in required header information. This includes the time and date and a computer generated unique identification. When finished, a message is sent to CONTRL and the startup task exits (3).



PHA COLLECTION INTER-TASK FLOW

Figure 3

When CONTRL receives the second message, it starts COLLEC, if it is not already active, and sends it a "send by reference" message pointing to the buffer (4). If the particular MBD does not have active collections, COLLEC will load the MBD with its programs. To get the required CAMAC branch addresses COLLEC returns a message to CONTRL (5). CONTRL searches EDI format configuration files on disk (6) and builds tables with all required commands in a small PLAS region. A "send by reference" is returned to COLLEC (7). CONTRL detaches itself from the tables region. COLLEC loads these tables into the MBD (8) and detaches and deletes the tables region. The timing and data handling programs in the MBD are started (still 8). These go to sleep but will restart themselves when there is something to do. To start the MBD collecting data, COLLEC runs the low priority MBD program PHAINI (9). This copies buffer and timing information to the MBD. It sets a flag so that both timing and data collection will commence (10). Everytime a LAM occurs for a detector that is in use, the location in the PDP 11 buffer corresponding to the detector and the appropriate energy range is incremented (11). For a detector that is not in use the LAM is simply cleared. When the timing program, utilizing a duration timer, decides collection has finished, the LAM handling program is told to stop data collection for that detector (12). This stoppage will also occur if a buffer location exceeds 32000 counts. Information about elapsed time, ADC dead time, and the reason for collection termination is stored in the PDP 11 buffer. COLLEC is informed of the finish via an AST caused by an MBD interrupt (13). COLLEC detaches itself from the PLAS buffer and sends a message to CONTRL informing it of the completion (14). If there are no remaining active collections in the MBD, all the PHA MBD programs are terminated and the MBD memory freed. CONTRL writes the buffer and header to disk (15). It detaches itself from the buffer region and deletes it. The terminal from which the startup task ran has a completion message written to it.

Utilities

Several utilities are available. The startup tasks are classed as utilities. There is a utility DETKIL which allows the researcher to request immediate termination of collection but save the data. DETSTA allows the researcher to find out if a particular detector is busy.

Several utilities are available to the system manager who knows the password. DETKIL will kill a collection from a different terminal than the one from which the collection was started. GLOSTA dumps internal tables on CONTRL. SHUT causes the collection system to disallow any new collections. All collections which are active at the time SHUT is run will go to completion.

PLAS USAGE

PLAS regions are allocated by COLLEC for two reasons. A 256 word region is created for the MBD tables each time an MBD is to be loaded. Once the send by reference has been made to COLLEC, CONTRL detaches itself. When COLLEC detaches itself after loading the MBD the region is deleted. The minimum region allocation for buffer space is 4K words. More than one buffer may reside in a region if there is room. A buffer that is longer than 4K has a region dedicated to it. Once the last active collection in a region finishes, the region is detached and deleted. A maximum of eight regions may be in use.

Using PLAS to build the tables might seem overly complicated. COLLEC could have fetched the CAMAC command tables by itself. By letting CONTRL do this work, COLLEC which must be memory resident during all active collections, need have no Fortran I/O routines included in task image. The PLAS routines must also be used to attach the buffers and so perform a dual function. Table building occurs only when a collection is started and no other collections are active on the target MBD.

CAMAC BRANCH RECONFIGURATION

Once the data collection system is quiescent, modules may be moved around at will on the CAMAC branch. New detectors or multiplexors may be added or equipment may be removed. Crates may be added or removed. CONTRL uses two files in EDI format to build all branch addresses when a collection starts. One file, CONFIG.CMC which is used by all ITRI data collection tasks, lists all module locations on the branch. The second, SYMMAP.LIB which is used exclusively by the nuclear data collection system, lists multiplexor characteristics, detector names, and deadtime clock information. A third small file, PHACHL.LIB, controls the priority of the MBD programs.

It is not possible to replace CAMAC modules with other vendor modules unless the command sequence is exactly the same. Malfunctioning modules can be swapped out as required with identical modules.

If a PDP 11 goes down for a significant period of time, the MBD can be changed to another PDP 11. By editing CONFIG.CMC, PHACHL.LIB and SYMMAP.LIB, the operational PDP 11 can be brought up with two CAMAC branches within minutes.

RELIABILITY

The software has performed reliably. After initial checkout systems failures have always been hardware failures. The MBD has been the most unreliable component. The system has the same weaknesses of any distributed CAMAC system. Powerfailures at crate locations are not necessarily detected. The software is somewhat error insensitive. Collections will continue as long as their required hardware appears to be operational. This includes the MBD's on multiple MBD systems. For failing collections the system assumes that no data is better than potentially corrupted data.

TERMINAL OUTPUT

All terminal output is done via a message handling task. All messages are logged both to the user terminal and to the console log. Collection tasks send a standard system message. It contains a message number relevant to the collection status. The message output task accesses an English language text file and outputs this message. If the text file cannot be accessed or the message number does not have a corresponding text, the message number is printed. Up to five different messages can be buffered in the message output task (that is ten active output requests to the system). Other messages will be awaiting their turn in the receive queue. If for some reason the message output routine is not available, CONTRL will print a message to the user terminal.

The slave task attribute is not used to communicate TI's. None of the utility tasks are installed. Hence their execution name is the name of the terminal. This task name is passed as a part of all messages. CONTRL saves the name so that it can be used as needed.

FUTURE EXPANSION

Only two enhancements are planned at the present. One is the capability to link detectors together. This will allow multiple detectors to have simultaneous collection starts and stops. This type of collection will allow one buffer for all or a separate buffer for each detector. The other enhancement will allow the researcher to have collection stop when a maximum specified count is obtained in any one energy range.

COMMENTS

The primary advantage of the system has been the time saved by direct entry of data into computer storage. With the analysis programs available it is possible to quickly accept or reject data. Data can be recollected immediately if required. The most difficult problem to overcome has been caused by the removal of the researcher from the operation of the equipment. Equipment calibration and correctness monitoring, which has traditionally been a function which the researcher would perform, tends to be ignored because the system is "computerized".

The first demonstration of the system gave birth to an interesting anecdote. We were showing all the advantages of the system, pointing out the efficiency of operation with respect to personnel time. We were using the OMNI startup program for the demonstration. No more did researchers have to twirl thousands of knobs and play with paper tape. One question we were asked was "Do we always have to answer all four questions?".

ACKNOWLEDGEMENTS

Work performed under U. S. Department of Energy Contract Number EY-76-C-04-1013.

The authors thank Drs. R. O. McClellan, M. B. Snipes, S. H. Weissman and J. A. Mewhinney and Mr. F. Barr and Mr. G. J. Newton for reviewing this manuscript.

We regret to announce the recent death of Mr. A. J. Hulbert.

AUTHOR/SPEAKER INDEX

PAPERS NOT SUBMITTED FOR PUBLICATION

PLANNING FOR A DECNET NETWORK
M. Weinstein

RSTS/E DISK INTERNALS
M. Mayfield

OPTIMAL BASIC-PLUS PROGRAMMING TECHNIQUES
M. Mayfield

USING SYSTEM TABLES
B. Alcock

ADDING KEYWORDS AND FUNCTION NAMES INTO THE DICTIONARY OF BASIC/RT-11
F.I. Magee

APPLICATIONS OF DBMS-11 TO MANUFACTURING SYSTEMS
F.R. Cope

A GRAPHICS PACKAGE FOR COMPUTER CALCULUS
K. Wooldridge

SYSTEM PERFORMANCE UNDER AN EDUCATIONAL WORKLOAD
R. Strickland

LSI-11 MICROPROGRAMMING AND ADVANCED TECHNIQUES
D. Gaubatz

MICROCOMPUTERS - A 5-YEAR LOOK
B. Demmers

THE HIDDEN POWERS OF THE DSM-11 DATA BASE AND THE ASSIST-11
A. Waisman

APPLICATIONS OF THE DALL-J AUXILIARY UNIBUS CONTROLLER
B. Weiske

SEWER SYSTEM EVALUATION SURVEY
S. Katz

MICROGRAPHICS RETRIEVAL WITH DATA BASE MANAGEMENT
W. Tabor

DESIGN OF A SECURE HIGH PERFORMANCE, RSTS BASED TRANSACTION PROCESSING
R. Briggs

PROJECT MANAGEMENT FOR COMPLEX SOFTWARE PROJECTS
F. Viggiano

A DATA DRIVE PLOT ROUTINE
C. Watson

GAM/GDL FOR THE VS60
K. Wheaton

ASCII GRAPHICS: A TECO RUNOFF DRAWING SYSTEM FOR SMALL SYSTEMS USE
D. Gaubatz

CURRENT STATUS AND PLANS FOR DECSYSTEM-8/78
D.S. Harmer

AN ALL CMOS MICRO-8 DEVELOPMENT SYSTEM
H.M. Smith

HOSTING NON-VMS SOFTWARE
R.A. Vossler

INFORMATION PROCESSING AND THE OFFICE OF THE FUTURE
M.B. Andelman

A DISTRIBUTED DATA ACQUISITION/CONTROL/ PROCESSING SYSTEM IN A PRODUCT ENGINEERING ENVIRONMENT
F.A. Spitler

DIGITAL SYSTEM PERFORMANCE ANALYSIS
R. Fadden

PERFORMANCE ANALYSIS - STATE-OF-THE-ART
R. Fadden

TRAX PRODUCT OVERVIEW
E. Hopey

ATTENDANCE

NAME	COMPUTER
Abbott, Doug, Standard Engineering Corp.	LSI,11
Abitboul, Jeanne, Scancom Corp.	LSI,11
Abma, John, Wittenberg Univ.	11
Abramson, Robert, DEC-Merrimack	VAX,11
Abuttahir, Ibrahim H., Minis.of Pet. & Min. Res.	VAX
Adams, Kenneth J., Canada Cement Lafarge	11
Adleman, Henry, DEC-Maynard	8, 11
Agrawal, Arun, Gould, Inc.	LSI,11
Ahnberg, Don, DEC	LSI,11
Aird, Tom, Int'l Math & Stat Library	11
Aker, Eric, California State Univ.	LSI,11
Albers, Robert, E.R. Squibb & Sons, Inc.	11
Albert, Ronald L., Manufacturers Hanover TRS	11
Alcock, Bruce, Riverdale Country School	
Alderman, D. W., University of Utah	11
Alderman, John, Digital Communications	8
Aldrich, Mike, Yuba Community College	LSI,11
Alexander, Jarvis, LIOCS Corp.	LSI
Alexander, William, Solid State Measurements	LSI, 8
Alexanderson, John, DEC-Merrimack	
Aley, Ray E., Jr., EG&G	11
Allard, Henry, DEC-Maynard	LSI,11
Allee, Jim L., L.D. McFarland Co.	11
Allen, Michael J., Lawrence Livermore Lab.	8, 11
Allen, Robert, Cisco-Pacific Inc.	
Allen, Rosemary, Lawrence Berkeley Labs	VAX,11
Allred, F. Mark, Philip Morris U.S.A.	LSI,11
Alpern, David, Saber Laboratories, Inc.	11
Alpern, Eugene, Saber Laboratories, Inc.	11
Alton, Michael, Lawrence Livermore Lab.	8,11
Alvarez, Patrick, KQED, Inc.	LSI,11
Ames, Donald R., DEC-Tewksbury	VAX
Amma, Joseph Jr., US Steel Research	11
Andelman, Michael B., LIOCS Corp.	LSI,11
Anderson, Charlotte, Ford Aerospace & Comm Cpr.	VAX
Anderson, Frank, Grand Canyon College	11
Anderson, Lea, DEC-Maynard	LSI,11
Anderson, Philip A., Information Systems, Inc.	11
Anderson, R.R., Chevron Geophysical Co.	VAX,11
Anderson, Roger, Lawrence Livermore Labs	8,11
Andert, John, Deferred Compensation	
Andreae, Sypko, Lawrence Berkeley Labs	LSI,11
Andrews, Harold, GENRAD	8,11
Andrews, Neal, Tri/Valley Growers	11
Anthony, Ralph, Ralfarithms Software Serv.	LSI,11
Applebee, Ralph, Idaho, The College of	11
Aquilera, Alfonso, Microprocesa Dores	VAX
Armstrong, Gary, Lawrence Livermore Labs	8,11
Armstrong, Mike, Badger Meter, Inc.	LSI,11
Armstrong, Nick, Lawrence Berkeley Labs	
Arnold, Jon, Delta College	11
Arris, David L., General Dynamics/WDSC	LSI,11
Arrowsmith, Donald, Naval Air Propulsion Ctr.	11
Arsenault, Raymond, DEC-Merrimack	CTS, 8
Arvish, Al, Alpha Omega Systems, Inc.	8,11
Askey, Robert C., ESL, Inc.	LSI,11
Aspegren, Robert, Solano Community College	
Atkinson, Lee, Avco Systems Division	LSI,11
Aubrey, Tom, Amcor Computer Corp.	11
Aurbach, Richard L., Monsanto Agr. Prod.	LSI,11
Austin, Don, Lawrence Berkeley Labs	VAX,11
Autran, Roland, Chase Manhattan Bank	VAX,11
Avery, Richard, Litton Systems, Inc.	LSI
Baatz, Eric, DEC-Tewksbury	11
Baccus, Don, Oregon Minicomputer	LSI,11
Backus, Robert, Gould Systems	LSI,11
Bagalay, Ronald R., US Army Corps of Eng.	8
Baker, Don, San Francisco Unified School	
Baker, John, Lawrence Livermore Labs	LSI,11
Baker, June, Computer Sciences Corp.	VAX,11
Baker, Lawrence M., US Geological Survey	8,11
Baker, Steven, J. Baker & Assoc., Inc.	VAX,11
Baker, William, Miami-Dade Community	LSI
Baldridge, Neil, Compu-Share Incorporated	CTS
Baldwin, Rich, North County Compu Serv	VAX,11
Bales, Robert, DEC-Salem	
Ball, Robert, British Columbia Bldg.Corp.	
Banovsky, Michael, DEC-Marlboro	
Barale, Paul, Lawrence Berkeley Labs	
Barale, Ronald J., Lockheed LMSC	LSI,11
Bargmeyer, Bruce, US Department of Labor	
Barker, Bruce, Valley News	11
Barker, Raymond E., Caterpillar Tractor Co.	VAX,11
Barnett, Bill, LASL	LSI,11
Barr, John, Montana, University of	LSI,11
Barr, Lee A., TRW Systems	11
Barrett, Martin, Security Industry	11
Barringer, Tim, NASA Ames Research Ctr.	VAX,11
Barry, David, General Electric	VAX
Bartelt, Mark, CA Institute of Tech.	LSI,11
Barton, Victor, Lawrence Livermore Lab.	8,11
Basch, Robert, Liberty Mutual Ins. Co.	
Batchelder, Maynard, Electrend, Inc.	11
Bate, Stephen D., DEC-Charlotte	
Bates, Kenneth, Real-Time Systems	VAX,11
Battat, Franklin M., Liberty Goldd Fruit Co.	8,11
Bauer, Jerry, Toscana Baking Co., Inc.	11
Baymler, B.E., NOAA/EDIS/CEAS	11
Beal, G.S., Nat Water Rsh Institute	LSI,11
Bean, Dwight, San Diego, University of	11
Beaner, Gary, Rainbow Computing Inc.	LSI,11
Bearden, Robert G., Amcor Computer Corp.	11
Beattie, Bruce C., Merritt College Compu Ctr.	11
Beattie, Judith C., Merritt College Compu Ctr.	11
Becker, J. Alex, Kentucky Machinery Inc.	
Becker, Teri, Informatics	VAX,11
Bedford, Ray G., Lawrence Livermore Labs	8,11
Beebe, James A., Environment Protection AG	11
Beeby, John, O'Brien Corporation	11
Beek, Clyde J., Boeing Company	11
Beer, Donna, Arnar-Stone Laboratories	11
Beidle, David, CA State Univ & Colleges	LSI,11
Bello, Keith, California State Univ.	LSI,11
Bendebba, Mohammed, Johns Hopkins School	11
Benedict, John S., Russ Systems	11
Bennett, Richard K., Lockheed Palo Alto RS Lab.	11
Benion, Scott T., Los Alamos Scientific Lab.	VAX
Benoit, Thomas, Information Systems, Inc.	11
Bensman, Kerry W., DEC-Waltham	
Benson, Bill, Lawrence Berkeley Labs	VAX,11
Benthusen, Donald E., Sandia Laboratories	LSI,11
Benton, Louis, Staff Computer Technology	LSI,11
Bentsen, P. Craig, Celanesc Chemical Co.	11
Berg, Gary, Chemineer, Inc.	11
Berg, Gary, American Sign & Indicator	8,11
Bergen, R. E., Calgary Power Ltd.	11
Bergstresser, Philip, TRW Systems	VAX,11
Bernick, Myrna, Magnavox Research Labs	VAX,11
Berry, Barry C., U.S. Army	11
Berry, Hershel, Gulf Interstate Eng Co.	
Bersig, James, Mercedes-Benz of N.America	CTS,11
Besel, Hilmer, Loma Linda University	11
Bettencourt, Gloria M., Kaman Science Corp.	11

NAME	COMPUTER
Bezanson, Bob, Computer Synergy, Inc.	LSI,11
Bezeredi, Paul, DEC-Tewksbury	VAX,11
Bickley, Lyle, Fidelity Bank	8,11
Bieler, John, Ecological Analysts, Inc.	11
Bieszczad, Kay A., F. N. Cuthbert Co.	11
Biller, James, MIT	LSI,11
Billig, Rich, DEC-Marlboro	LSI,11
Birkel, Peter J., Naval Avionics Center	11
Bitkowsk, Thomas, System Consultants, Inc.	LSI,11
Bitter, Mark, Transaction Technology	VAX
Black, Peter, E. R. Squibb IST. FOR	LSI,11
Blackett, Kent, DEC-Marlboro	VAX
Blair, Rodger C., DEC-Merrimack	VAX,11
Blake, Bennet, Boeing Comm Airplane Co.	VAX,11
Blake-Knox,M.W., Bell-Northern Research	11
Blazek, Daniel R., Sandia Laboratories	VAX,11
Blood, Glenn, Continental Group	11
Bobbitt, John S., FNAC	11
Bock, Frederick, Usariem	LSI,11
Boebinger, John, Boebinger Agency	11
Boerger, Stephen, Cincinnati Milacron, Inc.	11
Bogert, Peter, Howard Savings Bank	11
Boles, Evelyn, Tri/Valley Growers	11
Boll, Jim, H. S. Crocher	
Bolson, Edward, Washington, Univ. of	11
Bonham, Verlene, Sierra Digital Systems	8,11
Bonham, William, Sierra Digital Systems	8,11
Booker, John, Experimental Computer	VAX
Boone, David, DEC-Calgary, Alberta	VAX,11
Borges, Jerry, Lawrence Berkeley Labs	LSI,11
Bornmann, Robert, Rockefeller University	LSI,11
Bosin, Kenneth, GENRAD, Inc.	11
Bosworth, Bruce W., Pitney Bowes	VAX,11
Boule, Richard, General Electric Co.	LSI,11
Bowlby, James O., Lawrence Berkeley Labs	VAX,11
Bowman, James E. Jr., Computer Labs, Inc.	11
Boyd, Laurence, Gilbert/Commonwealth	11
Boykin, Wilber, NASA Johnson Space Ctr.	8,11
Boyle, Brian, Interactive Management	LSI,11
Boyle, P.J.R., Colorado, University of	LSI, 8
Bradley, Don, Digital Telephone Systems	LSI
Brady, Joyce E., Boeing Compu Services Co.	11
Braglia, Bob, Pontiac Motor Division	LSI,11
Bramhall, Mark, DEC-Merrimack	VAX
Brandt, B., DEC-Santa Clara	VAX,11
Brandt, J. Joseph, Lawrence Livermore Lab.	LSI,11
Brandt, Tim, McHugh, Freeman & Assoc.	VAX,11
Brannan, Gary, Georgia-Pacific Corp.	LSI,11
Branton, Robert A., Consulting & Contract	11
Branum, Alyce, DEC-Marlboro	
Brauch, C.J., Multi-List	VAX,11
Breitenstein, Richard, Naval Weapons Ctr.	11
Brenner, Alan, Creative Systems	8,11
Brew, Donald R., Fed Bureau of Investigation	11
Brewer, Robert, Amherst Associates, Inc.	11
Bridge, Thomas, Norco Mills of Norfolk	CTS,11
Brierley, Stuart, Gamma Associates Ltd.	LSI,11
Briggs, Richard S., DEC-Meriden	
Brind, F. Alan, DEC-Merrimack	
Brindley, Bill, Defense Communications	LSI,11
Britton, Barbara J., Lawrence Berkeley Labs	VAX,11
Brodine, Robert, Argo Systems	8
Bronskill, M.J., Ontario Cancer Institute	11
Bronson, Mark, California, University of	11
Brooks, Clarence M., DEC-Merrimack	CTS
Broussard, Sharon M., Naval Air Rework Facility	11
Brown, Bill, DEC-Tewksbury	
Brown, Bob, DEC-Nashua	LSI,11
Brown, Howard, McLaughlin Research Corp.	11
Brown, James R., McAuto	8,11
Brown, John, Stanford Linear	
Brown, Linda, Afis/Ind	11
Brown, Reid, DEC-Tewksbury	VAX,11
Brown, Stephen, Merritt College	11
Bryant, Wayne M., Composition Systems, Inc.	8,11
Bryski, Sholom, Automated Concepts, Inc.	11
Buckley, Maurice R., Ford Aevo	LSI
Buddenhagen, David, Western Electric Co.	8
Buffenbarger, David, CNA Insurance	
Buhman, Don, Blue Mt. Community College	
Bunker, J. K., Chevron Research Company	LSI,11
Burd, William C., Sandia Laboratories	VAX,11
Burger, Hope, Merritt College	
Burger, Raymond Jr., Southern CA Rsh Institute	8,11
Burke, Thomas J., RCA	11
Burkhart, Bruce, Lawrence Berkeley Labs	VAX,11
Burnes, N. J., Informatics, Inc.	VAX,11
Burnett, James, Wagner Data Systems	CTS,11
Burns, Elinor, DEC-Maynard	
Burris, Randall, Oak Ridge National Lab.	
Butler, Robert, E.I. Dupont Denemours	VAX,11
Byfield, T. E., Duval Corporation	VAX,11
Byram, Susan K., California Scientific Sys.	8,11
Byrne, Charles, Philip Morris, Inc.	11
Cadieux, Edward, Information Systems, Inc.	11
Caldwell, Robert, Milliken & Company	VAX
Calek, Art, DEC-Rolling Meadows	VAX,15
Calko, Samuel, Philip Morris, Inc.	11
Cameron, M. Dianne, Consolidated Comp. Inc.	VAX,11
Campbell, Jay C., Business Computer Systems	VAX,11
Canafax, Thomas, Data Route, Inc.	8,11
Canal, F. C., Shell Francaise	8,11
Cannaveno, John J., Hoffmann La Roche, Inc.	VAX,11
Cannon, Philip, Science Applications, Inc.	LSI,11
Capel, David, Western Pacific Railroad	
Capowski, Joseph, North Carolina, Univ. of	LSI,11
Carabetta, Michael, Interactive Management	LSI,11
Caraher, William, E.I. Dupont Company	11
Cardoza, Wayne, Bell Laboratories	VAX
Carlock, Linda E., Hughes Aircraft Company	11
Carr, Richard, El Paso Natural Gas Co.	LSI
Carrato, Anthony, USAF-Air Training Com.	VAX,11
Carroll, A. B., American Business Compus	11
Carter, Clairmont, Babson College	11
Carter, G.W., Bunker Ramo Corp.	VAX,11
Carter, Vicki, Data Route, Inc.	8,11
Carter, William L., Intermedics, Inc.	11
Caruso, Sally, NALC 203C3 U.S. Navy	VAX,11
Cary, Clifford N., Creare, Inc.	LSI,11
Casillo, John, Product Management Corp.	11
Casper, E. H., Diamond Shamrock Corp.	VAX
Cassaro, Edward S., California, Univ. of	LSI,11
Cassinelli, Shirley, Lawrence Berkeley Labs	VAX,11
Castain, Eric, Ecological Analysts	11
Catinella, Paul, Railcar Maintenance Co.	11
Caulk, Harry, Collins Pine Company	8,11
Cerchio, Gerard, Informatics/Programming	LSI,11
Chadwick, H. E., Western Electric Co., Inc.	LSI,11
Chaffeur, James, United States Computers	11
Chalmers, Bryan, Willipeg, University of	11
Chalmers, Leslie, Crocker National Bank	11
Chamberlain, Ryan K., Impack Services	CTS,11
Chan, Dennis, Bell Laboratories	11
Chan, Paul, Lawrence Berkeley Labs	VAX,11
Chan, Ping Chung, Hong Kong Polytechnic	11
Chao, James Lee, Lawrence Berkeley Labs	LSI,11
Chapin, David, Boys Town Research Center	11
Chapman, Dennis, Georgia-Pacific Corp.	8,11
Charlot, Courtland, DEC-Maynard	LSI,11
Chen, Jones C., Sunoco, Inc.	11

NAME	COMPUTER
Chernoff, Anton, DEC-Maynard	LSI,CTS
Cherry, Michael, Texas, Univ. of,@ Dallas	11
Chi, Donald, U.S.Bureau of Mines	11
Chiang, James, Royal Hong Kong Jockey Cl	11
Chin, Sau, NWS	11
Chodosh, Daniel F., Smithlilne & French Labs	LSI,11
Choy, Steven, Harry Diamon Labs	8,11
Choy, Michi, Crocker National Bank	11
Christensen, Peter, National Bank of Detroit	11
Christian, Chris, Lucky Stores, Inc.	8,11
Christy, Peter, DEC-Maynard	
Chu, Hilton, Naval Weapons Station	11
Church, Gregg, Gejac, Inc.	LSI,11
Church, Janice, Gejac, Inc.	LSI,11
Church, James D., Westinghouse Broadcasting	LSI,11
Cirilo, Lionel, H. S. Crocker	
Clark, Alan R., Lawrence Berkeley Labs	VAX,11
Clark, Jerry, American Sign & Indicator	LSI,11
Clark, Michael, McDonnell Douglas	VAX
Clark, Richard, Zonge Engineering	8
Clarke, Timothy O., Menlo Computer Associates	8,11
Clemens, John, Damon Corporation	11
Cleveland, David H., Lawrence Berkeley Labs	LSI,11
Cline, James W., California, University of	8,11
Clingerman, David, Fairchild	11
Clites, Roy, DEC-Maynard	VAX,11
Coffeen, Tricia, Lawrence Berkeley Labs	VAX,11
Cohen, David B., Philip Morris Inter	CTS
Colan, Gary, Lutheran General Hospital	
Cole, Gary, DEC-Maynard	CTS, 8
Cole, Mary, DEC-Maynard	LSI
Cole, Michael, EG&G Idaho, Inc.	8,11
Cole, Stephen, Georgia Inst. of Technology	8,11
Cole, Sue, American Sign & Indicator	LSI,11
Cole, Vern, Georgia-Pacific Corp.	LSI,11
Coleman, H. Legare, Milliken & Company	VAX,11
Collins, John, 3M Company	11
Collins, Roger, U.S. Leasing Corporation	VAX,11
Condon, Terry, DEC-Merrimack	CTS
Conklin, Peter, DEC-Tewksbury	VAX,11
Conley, Chuck, DECUS-Marlboro	
Connell, Clyde H., Env.Rsh.Inst.of Michigan	8,11
Conville, John, Cableshare	
Cook, R. L., Bunker Ramo Corp.	VAX,11
Cooke, C.L., Digital Telephone Systems	8,11
Copeland, Lee, LDS Church	LSI,11
Copp, Anne S., DEC-Maynard	
Copper, Jack, Compuguard Corp.	LSI,11
Coppola, Victor, DEC-Maynard	
Cornwall, Susan, DEC-Maynard	CTS
Corthell, N. David, Generix Drug Corporation	
Coryell, James, Coryell Graphics	LSI, 8
Coryell, Judith, Moorpadt College	8
Coscia, Donald, Suffolk Community College	8,11
Cossalter, John, B.C. Telephone Company	VAX,11
Cossette, Angela, DEC-Maynard	
Coston, Arthur W., Applied Information Sys.	LSI,11
Cottrell, James, TRW Defense & Space Sys.	VAX,11
Cottrell, R. Les, Stanford Linear Accel Ctr.	LSI,11
Couch, Carol S., Naval Air Rework Facility	11
Coulsell, Robert R., International Comm. Sci.	11
Couvreur, Thomas, Chemical Abstracts Service	VAX,11
Cox, Patricia R., Los Alamos Scientific Lab.	11
Cramer, Timothy, Puget Sound, University of	11
Crapuchettes, Jim, Menlo Computer Associates	
Crawforth, Jim, U.S. Geological Survey	
Creamer, Roger, CTB/McGraw-Hill	VAX,11
Cromartie, Edmund S., Magnavox Govt.& Ind.Co.	LSI,11
Crosby, Richard, DEC-Merrimack	CTS, 8
Cross, Allen R., Arc Associates	11
Cross, John P., Soil Testing Services, Inc.	11
Cross, Kenneth, Oak Ridge National Lab	LSI,11
Crossnoe, Marvin, Compu-Share, Inc.	11
Crow, Jerry, Georgia-Pacific Corp.	8,11
Crow, Vern, Battelle-Northwest	VAX
Crowley, Roger, Union-Tribune Publishing	11
Crum, Sterling, Nordata	VAX,11
Cruson, Pieter, Quodata Corporation	
Cuddy, Dave, Bell-Northern Research	LSI,11
Cumming, Gordon C.G., Computer Dynamics Inc.	11
Cuomo, Vince, Beckman Instruments, Inc.	
Cureton, Kenneth L., Dialogue Computer Sci.	LSI,11
Curley, Robert, American College	11
Curley, William, Federal Reserve Bank	VAX,11
Curtis, Daniel, Fermi Nat'l Accelerator	VAX,11
Curtis, Robert E., U.S. Government	VAX
Cushman, Robert, Jones & Lamson	LSI
Cutler, James, Michigan, University of	8,11
Cutter, Mark, Lawrence Berkeley Labs	VAX,11
Cytry, Allan, Bankers Trust Company	
Dagraca, John J., Hughes Aircraft	
Daley, Robert, DEC-Merrimack	VAX,11
Dalrymple, Brent, U.S. Geological Survey	8
Daly, Kathleen, Solano Community College	
Dammeier, John, Tacoma News Tribune	11
Damon, Peter, DEC-Maynard	VAX
Damour, Paul, DEC-Mill Valley	VAX
Danbury, Thomas, Survey Sampling, Inc.	11
Dancy, Marion, DEC-Tewksbury	VAX
Daniels, David W., KMS Fusion, Inc.	
Daniels, Robert, Monsanto Co.	LSI,11
Danner, Bruce, Rose-Hulman Inst.	11
Darley, Lucy, Evans Griffiths & Hart	VAX,11
Dash, Mike, John Fluke Mfg. Co., Inc.	VAX,11
Dashevsky, Marc, Evans Griffiths & Hart	VAX,11
Daugherty, Michael, DEC-Merrimack	CTS, 8
Davila, Roberto, DEC-New York	VAX,11
Davis, Burt, Los Alamos Scientific Lab.	VAX
Davis, Dorsey, Dow Chemical Company	11
Davis, Gregory, Lawrence Livermore Labs	LSI,11
Davis, Jerry M., Foto Air	8
Davis, Lawrence W., American Col.of Radiology	11
Davis, Mary S., California State Univ.	11
Davis, Roy, DEC-Tucson	
Davison, Marilyn, DEC-Maynard	VAX,11
Davy, Donn, Lawrence Berkeley Labs	VAX,11
Dawson, Randy, DEC-Maynard	
Dayringer, Henry, Monsanto	LSI,11
Dean, Robert, Kentucky State University	11
Deeter, Wayne, Able Computer Technology	LSI,11
Degraffenreid, Duncan, USDA, SEA, Communica	LSI,11
Degroff, Michael, WR Grace & Company	11
Degroot, Tony, Lawrence Livermore Labs	LSI,11
Deininger, Axel, William M. Mercer, Inc.	VAX,11
Delaney, Dave, Ramada Inns, Inc.	11
Delano, E. Leon, Kenyon College	11
Delia, Clark, DEC-Tewksbury	11
Deller, George, Computer Synergy	LSI,11
Deller, Steven R., Computer Sciences Corp.	VAX,11
Deloyht, Thomas E., Panhandle Eastern Pipe Line	11
Demasek, Frank, General Motors Corp.	11
Demers, Kenneth, United Tech Research Ctr.	LSI,11
Demmer, William, DEC-Tewksbury	LSI,11
Denniston, Dave, DEC-Maynard	
Denny, Charles, DEC-Marlboro	VAX
Desaussure, Raymond, Lawrence Livermore Labs	VAX
Desmarais, Joyce, DEC-Maynard	
Dey, Walter, TRIUMF/UBC	VAX,11
DiBartolomeo, May I., US Food & Drug Adm/Edro	11
Dickinson, Sandy, DEC-Maynard	
Dierich, Leonard A., Economics Laboratory, Inc.	11

NAME	COMPUTER
Dietrich, W.V., Kastle Systems	LSI,11
Dightam, John, Canadian Broadcasting	11
Dill, William Jr., Matrelon, Inc.	LSI
Dilworth, John, Lawrence Berkeley Labs	LSI,11
Dionne, Paul J., Battelle Pacific NW Labs	11
Dippner, Ralph, New York State Univ.	8,11
Dixon, John, EMR Telemetry	LSI,11
Dohan, D. A., California, University of	8,11
Dolan, Robert A., Comdesign, Inc.	VAX
Dole, Stuart, California, University of	11
Dollar, Glenn, Rainbow Computing, Inc.	LSI,11
Dollar, William R., Crystal Oil Co.	11
Dollard, John A., Navy Per Rsh & Dev Ctr.	11
Dolph, Edwin J., Schlumberger-Doll Rsh Ctr.	VAX,11
Doman, Jennifer, Hydroscience, Inc.	11
Donahue, John G., DEC-El Segundo	LSI,11
Doob, Michael, Manitoba, University of	8
Dosen, Robert, Fermi Nat Accelerator Lab.	VAX,11
Doupont, Paul, Lawrence Livermore Labs	8,11
Dowlin, Kenneth E., Pikes Peak Reg. Library	
Downie, Jerry L., Coor's Porcelain Company	11
Downward, James G., KMS Fusion, Inc.	VAX,11
Doyle, Terry, System Development Corp.	LSI,11
Dray, Robert, Digital Equipment Co. Ltd.	
Dress, Henry, Naval Air Propulsion Ctr.	11
Drummond, John, Ontario Hydro	LSI,11
Dubay, David J., DEC-Merrimack	8,11
Dudley, Charles M., PRC/Information Sci. Co.	VAX
Duffy, Darrell, DEC-Maynard	11
Duffy, James M., Lawrence Livermore Labs	VAX,11
Duft, Thomas G., 154 Riveredge Road	VAX,11
Duke, Dennis C., Interactive Info Systems	11
Duncan, Anne, DEC-Merrimack	VAX,11
Duncan, Russell, US Government Printing Office	11
Dunham, Sam, California State University	11
Durnford, Dick, Alpha Omega Systems, Inc.	8,11
Dutton, Ross, North West Computer Ser.	11
Dutton, Robert P., ESL/Inc.	LSI,11
Dye, Toby, California University	LSI,11
Easton, John T., Minnesota, University of	8,11
Eau Claire, Joseph L., Booz, Allen & Hamilton	VAX,11
Ebinger, Larry, Sandia Laboratories	LSI,11
Eckman, Ren, Computer Synergy	LSI,11
Eckroth, Gary, DEC-Merrimack	
Edelstein, Ronald, American Cyanamid Company	11
Edmundson, Edward H, Minnesota, University of	8,11
Edwards, Bruce, Altel Data	11
Edwards, Steven, Cal Poly University	8,11
Edwards, Tom, Tri/Valley Growers	11
Eggers, Maury C., Corning Glass Works	11
Ehrlich, Charles, Crocker National Bank	LSI,11
Einstein, Stewart A., RDA	11
Eisenberg, Lawrence H., Eisenberg Law Corp.	CTS, 8
Eley, Herbert W., U.S. Navy	VAX,11
Elkine, David, Arnar-Stone Laboratories	11
Ellington, Jerry, Crocker Bank	11
Elliott, Paul, Technology Inc.	VAX,11
Ellis, Charles R., Indiana University	LSI,11
Ellis, Russell, System Design	LSI,11
Emerson, Mark, Seattle Pacific University	LSI,11
Enders, Robert B., Think Inc.	8,11
Englberg, Norman, General Electric	VAX,11
Engleman, Roger, USDA-SEA-AR	11
English, Paul, Boeing Comm Airplane Co.	8
Enicks, Charles R., Commonwealth Clinical Sys.	11
Epstein, Arnold, J. Baker & Assoc., Inc.	8,11
Equals, R.V., TRW Energy Systems Group	CTS,11
Erickson, Robert, Stiffel Company	11
Esbensen, Dan, North County Compu Serv	VAX,11
Esfandiari, Mary Ann, NASA/Goddard Space Fl Ctr.	11
Estes, James W., Int. Crops Rsh. Institute	
Esteva, Carlos A., Microprocesadores	LSI,11
Ethier, Errol E., Wachusett Reg. Sch. Dist.	
Evans, Thomas, Evans Griffiths & Hart	VAX,11
Evanson, R. A., Paragon Data Systems, Inc.	CTS,11
Everhart, Glenn C., RCA	VAX
Ewen, Carl G., Tubular Steel Inc.	11
Facer, Eilene, TRW	11
Fafarman, Dave, EDS Nuclear Inc./ATD	VAX
Farnan, Ron, Institute for Law	11
Farrell, B., Informatics, Inc.	11
Farrell, H. James, St. Mary's College	11
Farrington, N.J., Computer Applications	CTS
Fauber, Marion M., Delco Products Div GMC	VAX,11
Faulconer, Robert, DECUS-Marlboro	
Faunt, Douglas, ESL, Inc.	LSI
Fein, Michael, DEC-Maynard	
Felling, Stephanie, Electrend, Inc.	11
Fenrick, Michael, Science Applications, Inc.	11
Ferber, B., Santa Clara County	
Ferguson, Francis, Alberta Government Tele.	VAX,11
Ferraro, Frank, Georgia-Pacific Corp.	LSI,11
Ferry, James Jr.	LSI, 8
Ferry, James III	LSI, 8
Fettes, George, Foothills Hospital	LSI,11
Figler, Alan, Telemed Corp.	11
Finch, Geoff, DEC-Nashua	
Finch, Joanne, E.R. Squibb and Sons	11
Fincke, William B., Scripps Inst. Oceanography	LSI,11
Fish, David, Lockheed Space & Missile	VAX,11
Fite, Ken	
Fitzgerald, Brian, DEC-Merrimack	
Fjelsted, Kevin, Lawrence Berkeley Labs	LSI,11
Fleischer, Richard, Ohrbach's Inc.	11
Flesher, Richard L., Medical Arts Laboratory	LSI,11
Floyd, Richard A., California, Univ. of	VAX,11
Flynn, Rick, TRW Information Services	11
Fogels, E. A., Toronto, University of	LSI,11
Foley, George B., Bell Labs	LSI,11
Foley, Kenneth, Planning Research Corp.	11
Foote, Brian, Illinois, University of	LSI,11
Ford, Brian, Numerical Algorithms Grp.	VAX,11
Ford, Donald, Public Broadcasting Serv.	VAX,11
Ford, James, Hoffman La Roche Inc.	VAX,11
Forecast, John, DEC-Maynard	
Foreman, Alling C., PMC, Inc.	8,11
Forster, Doug, Sylvania	VAX,11
Foster, Frank, Dow Chemical Co.	LSI,11
Frailey, Dennis, Texas Instruments Inc.	VAX,11
Frankel, Allan, Integrated Software Sys.	LSI,VAX
Fraser, Michael, Armed Forces Radiology	LSI,11
Frazer, Kent, Rubicon Corporation	
Frean, Charles, DEC-Maynard	
Fredericks, Frank J., Rockwell International	VAX,11
Fredricksen, Tom, Accounting Service Co.	LSI
Freeborn, Chris, Esso Chemical Canada	11
Freeman, Ronald B., Bell Laboratories	VAX,11
Freeman, Dean, Arkansas, University of	LSI,11
Freiburger, Dana, California State Univ.	11
Freier, Alan O., DEC-Carrollton	LSI,11
French, Donald Jr., General Dynamics/WDSC	
French, Raymond B., Boeing Comm Airplane Co.	VAX,11
French, Richard D., Cincom Systems of Canada	VAX
Fried, Robert, DEC-Merrimack	VAX,11
Friedrich, Kurt, DEC-Tewksbury	
Friesen, Richard D., Lawrence Livermore Labs	8,11
Froemming, G., Informatics, Inc.	VAX,11
Fultyn, Robert V., Los Alamos Scientific Lab.	VAX
Furchner, Donna, American Sign & Indicator	LSI,11
Fushimi, Fred C., Monsanto Research Corp.	8,11

NAME	COMPUTER
Fustes, Manuel, Gulf Interstate Eng. Co.	
Gaarder, Bruce L, Macalester College	11
Gabelnick, Stephen D., Argonne National Lab.	LSI,11
Gache, Donald, International Datasystems	8,11
Gagnon, Raymond Jr., Wyman-Gordon Company	11
Gale, David, Mini-Compu Business Appl.	
Galla, Ron, Georgia-Pacific Corp.	8,11
Gallant, Walter	
Gallup, Jeff, Lawrence Berkeley Labs	
Ganbarg, William, Chicago, University of	11
Gardner, Ted, Georgia-Pacific Corp.	8,11
Gardner, Brian, Comtech Group	
Garrett, Ronald, Valley News	11
Garth, William Jr., Chevrolet Engineering Ctr.	11
Garton, Bill A., Boeing Computer Services	VAX,11
Gau, Dennis, Western Electric	11
Gaubatz, Donald, DEC-Maynard	LSI
Gault, Susan, DEC-Tewksbury	VAX
Gauronskas, Charles T., Field Research	
Gausman, R. K., TRW Energy Systems Group	CTS,11
Gebbie, Ray, Guntert Sales Div. Inc.	
Gee, Wen-Sue, Lawrence Berkeley Labs	VAX,11
Geiger, Duane L., Colorado Systems Group	11
Geiger, Richard G., DEC-San Francisco	11
Geller, Dalia, Lawrence Berkeley Labs	
Gemmill, Bill, Orange County Hospital	LSI,11
Gendreau, Raymond, Kellogg Company	8,11
George, Pat, E & J Gallo	11
Gerber, Bob, LDS Church	LSI,11
Gernert, James M., Energy Ent. of Denver, Inc.	
Gertz, Kenneth A., Lawrence Livermore Labs	8,11
Gervasi, Samuel M., Rochester Telephone Corp.	
Gey, Fred, Lawrence Berkeley Labs	VAX,11
Ghose, Abigail J., Magnavox Govn't & Ind.	8,11
Giao, Anthony	8,11
Gaio, Mark	8,11
Gieraltowski, Gerald F., Argonne National Lab.	VAX
Giesa, Patrick L., Stein Distributing Co., Inc.	CTS
Giesler, Gregg, Los Alamos Scientific Lab.	LSI,11
Gifford, Robert, Steiger Tractor Inc.	11
Gildea, Thomas J., Nebraska, Univ. of Med. Ctr.	LSI,11
Gill, A. J., Interlake, Inc.	11
Gillette, Glenn L., DEC-Merrimack	VAX,11
Gilligan, Barry D., Mission Research Corp.	VAX
Gimbel, Rick, DEC-Marlboro	VAX,11
Gin, Helena Won, Lawrence Berkeley Labs	VAX,11
Girard, Paul, Alcan	11
Gisseler, James M., Illinois Tool Works, Inc.	11
Giudice, John, DEC-Marlboro	LSI
Glackemeyer, Richard, Boeing Computer Service	VAX,11
Glaser, Gerry, Skidmore, Owings & Merrill	11
Glazer, Eli, DEC-Maynard	
Glish, Michael K., Caterpillar Tractor Co.	VAX,11
Glisky, Dan, Pontiac Motor Division	LSI,11
Globe, Carla B., Chemical Abstracts Ser.	VAX,11
Glorioso, Robert M., DEC-Maynard	LSI,11
Glover, Dell, DEC-Marlboro	
Goding, David, Lewis and Clark College	VAX
Gogue, Michael, TRW Defense & Space Sys.	11
Goings, J. Steven, Mostek Corporation	VAX,11
Golde, Hellmut, Washington, University of	VAX
Golden, Donald, Litton Resources Systems	LSI,11
Goldman, Earl, Nitty Gritty Productions	11
Goldman, Karen, Nitty Gritty Productions	11
Goldman, L., DEC-Santa Clara	LSI,11
Goldstein, Jacob, Cusys Inc.	8,11
Goldwire, Henry C Jr., Los Alamos Sci. Lab.	LSI,11
Gonzales, Rafael, All Indian Pueblo Council	11
Goodhue, Alan, Digital Management Corp.	VAX,11
Goodman, Terri A., EG&G Inc.	11
Goodrich, Gerald, DEC-Maynard	LSI,11
Goodwin, Keith, Otero Jr. College	
Gordon, Jack, Lunday-Thagard Oil Co.	VAX,11
Gordon, Richard, Dofasco	LSI
Gordon, Richard H., Sonoma State University	11
Gorlen, Keith, Nat'l Inst. of Health	LSI
Gould, Judith	11
Graham, Kathy, ESL, Inc.	LSI,11
Graham, William, Systems Consultants, Inc.	LSI,11
Grand, Diana, Lawrence Berkeley Lab.	VAX,11
Graniero, Charles, Stanford Linear	
Grant, Chris, Willis, Conluffe, Tait	8
Gratzer, George, Manitoba, University of	8
Graves, Wayne R., D.C.A. Reliability Lab.	LSI,11
Gray, Raymond C., American Tel & Tel Co.	LSI,11
Gray, Leonard, Valley News	11
Gray, R., Skidmore, Owings & Merrill	11
Greco, Richard, Lewis and Clark College	VAX
Green, Jim, Washington, University of	11
Green, Kelly, Washington, University of	LSI,11
Green, Tom, California, University of	LSI,11
Greene, Jay, International Datasystems	8,11
Greenwood, James R., Lawrence Livermore Labs	LSI,11
Greer, CA, Naval Air Rework Facility	11
Gregory, Charles W., Colgate-Palmolive Co.	11
Greiman, Bill, Lawrence Berkeley Labs	VAX,11
Griffel, David, Admin Inc.	LSI,11
Griffith, Frank, Arizona Health Sci.Ctr.	11
Griffith, H. Russell, Digital Management Corp.	VAX,11
Griffith, Ronald D.L., Michigan, Univ. of	LSI,11
Griffith, W. D., Mobil Research & Develop.	11
Grigsby, Fred, Dofasco	LSI
Grimwood, Neil W., Valley National Bank	11
Grinstead, Harold B., Mason&Hanger-Silas Mason	8,11
Gross, Jerome, Union Electric Co.	11
Gruenemeier, Henry R., Airesearch Mfg. Company	11
Grundler, Mark W., Grinnell College	11
Grutze, Al, Tri/Valley Growers	11
Guarino, Ruth B., E.R. Squibb Inst. for Med.	LSI,11
Guerrero, Henry C., Atlantic Richfield Co.	11
Guldenschuh, Charles, DEC-Maynard	11
Gumm, Barbara, Lawrence Radiation Lab.	8,11
Gunderson, Robert L., James S. Kemper & Company	
Gustafson, Robert, Simultation Specialists	LSI,11
Guthrie, Charles W., Inco, Inc.	11
Guy, Richard, Loma Linda University	11
Guzman, Marc A., Cedars Sinai Medical Ctr.	11
Haas, Raymond J., Navy Personnel RSH & DEV	11
Hager, Mark J., Economics Laboratory, Inc.	11
Hagmeier, Jerry, Gardner-Denver Company	8,11
Haigh, David, DEC-Bedford	VAX
Hakimi, Ruben, Missouri, University of	11,15
Halbert, Daniel, California, University of	VAX
Hall, Judith J., DEC-Acton	
Hall, Sue, Southwestern at Memphis	11
Ham, Ronald J., DEC	VAX
Hamaker, David W.	LSI,11
Hamaker, Kathryn A.	LSI
Hamilton, Bruce, ITT Research Institute	
Hamma, George, Synergistic Technology	LSI,11
Hammersley, Richard, Gilbert/Commonwealth	11
Hancock, Jack, TRW Defense & Space Sys.	VAX
Hander, Edwin, U.S. Army Institute	LSI
Haney, Don, DEC-Maynard	LSI,11
Hankley, Donald, Naval Ship Eng. Ctr.	8,11
Hanks, J. G., U.S. Marine Corps	11
Hanks, Charles, DEC-Merrimack	
Hannan, Peter J., Minnesota, University of	11
Hanson, R.C., Engineering-Science	11
Hanus, Michael J., Wisconsin Med.College	11

NAME	COMPUTER
Hardesty, Dennis, Naval Weapons Support Ctr.	8
Hardin, Bill, Georgia-Pacific Corp.	11
Hardy, Donald E., DEC-Maynard	
Harlow, Frederick, Naval Research Lab.	8,11
Harney, Michael J., Construction Data Sys. Inc.	11
Harper, R.M., California, University of	LSI,11
Harrington, Richard J., Pacific Telephone Co.	8,11
Harris, D., Canada Cement Lafarer	11
Harris, J. Gregory, Georgia-Pacific Corp.	LSI,11
Harris, Kevin, DEC-Tewksbury	VAX,11
Harrison, James, Los Alamos Scientific Lab.	LSI,11
Harrison, Steve, Skidmore, Owings & Merrill	11
Hartman, Dan, Naval Air Rework Facility	11
Hartnett, Nicole, DEC-Maynard	
Hartnett, William, International Datasystems	8,11
Harvey, Bradford, County of Los Alamos	11
Hassinger, Robert, Liberty Mutual Rsh Ctr.	8,11
Hassler, Ardoth, Central State University	11
Haugen, Richard R., Intern'l Harvester Co.	VAX,11
Hauger, Carl Jr., E. I. Dupont	8,11
Havlin, Robert H., Hydro-Air Engineering, Inc.	LSI
Hawthorn, Paula, Lawrence Berkeley Labs	VAX,11
Hayden, J. Michael, Buck Knives, Inc.	11
Hayes, J. A., Computer Ctr CA St Univ.	LSI,11
Hayes, John G., South Central Bell Tel. Co.	LSI,11
Haymond, Ed, Georgia-Pacific Corp.	8,11
Heafer, John, Browning-Ferris Ind., Inc.	8,11
Heald, L., Douglas Inc.	11
Hebert, George, Quebec Deposit & Invt. Fund	
Heckler, Wayne G., TRW	VAX,11
Heckman, Brad, Lawrence Berkeley Labs	VAX,11
Heffner, Stephen, Independent Consultant	LSI,11
Heffron, Mathew, Beckman Instruments Inc.	
Hehmann, Robert A., Cedars Sinai Medical Ctr.	LSI,11
Heidebrecht, J.B., TRW DSSG	VAX,11
Heintz, David M., GENRAD Inc.	
Heintz, Philip, Radiation Oncology Ctr.	11
Helton, James W., EG&G Inc.	11
Hemphill, John P., Gilmore Envelope	11
Henderson, Bill, DEC-Merrimack	VAX,11
Henderson, Charles, Pomona Valley Aviation	11
Henderson, Lofton R., NCAR	11
Hensiek, F.W., EG&G	LSI, 8
Hermes, A. III, MIT	LSI,11
Herron, Peter, Suffolk County Comm. Col.	11
Higgins, Mike, Cetus Corporation	11
Higgins, Ronald D., DEC-Marlboro	
Hill, Dave, GSU Sacramento	11
Hill, Joe, Georgia-Pacific Corp.	8,11
Hill, Richard, Consolidated Film Ind.	LSI
Hilton, Roy, Chevron Oil Field Rsh.Co.	VAX,11
Hinkins, Ruth, Lawrence Berkeley Labs	VAX,11
Hirschfeld, David J., Business Controls Corp.	8,11
Hitson, Bruce L, Stanford Linear Accel. Ctr.	LSI,11
Ho, Chanh, New Britain Housing	11
Ho,Khanh, New Mexico Junior College	
Hobbs, Craig W., Iowa, University of	LSI,11
Hobde, Dennis, GTE Sylvania, Inc.	VAX,11
Hoffman, Robert C., Lawrence Berkeley Labs	11
Hoffman, Wilson, California, University of	11
Hogan, Cheryl, DECUS-Marlboro	
Hohmann, Edward C., Polytech State Univ.	11
Holbrook, E.D., Sandia Laboratories	11
Holdsworth, George, Staff Computer Tech Corp.	VAX,11
Holeman, C.W., Electron Inc.	11
Holloway, Ian, General Foods Ltd.	11
Holmes, Carl, Continental Computer	11
Holmes, Harvard, Lawrence Berkeley Labs	VAX,11
Holsworth, Frank, California, University of	11
Holt, Mary M., A.H. Robins Company	
Holter, R.F., Rockwell International	LSI,11
Hopkins, H. Kenneth, California, Univ. of	8
Hopp, R.J., Swift & Co., R&D Center	11
Horn, Bruce, Los Alamos Scientific Lab.	LSI,11
Hornik, Gerald J., DEC-Merrimack	VAX,11
Hourican, J.F., W.R. Grace & Company	11
House, Ronald A., Naval Underwater Systems	VAX,11
Hoverter, Earl, Naval Air Rework Facility	11
Hovland, Hal, Hal Systems Corp.	11
Howard, Dan, Bell Canada	
Howe, Bill, Collins Pine Company	8,11
Howell, Tim, U.S. Fleet Leasing Inc.	11
Hoy, Herbert H., NASA Ames Research Center	LSI,11
Hoyè, Vicki, Henry Wurst	11
Hsia, Albert, DEC-Tewksbury	
Hubbard, John W., Pacific Data Systems	8
Hubbard, William E., L.D. McFarland Company	11
Hube, Randall, Xerox Corporation	8,11
Huffington, James, Virginia Mason Hospital	8,11
Hughes, Olwen, Sranot	11
Hull, Leota, Pan American University	LSI,11
Hull, Marilyn J.,	11
Humlie, George P.A., Morgan Equipment Company	CTS,11
Hunt, Dan P., EG&G Inc.	LSI
Hunter, Barrie, DEC-Maynard	
Husami, Art, DEC-Santa Ana	
Hustvedt, Richard, DEC-Tewksbury	VAX
Hunter, Richard, Enterprise Companies	
Hyman, Neil, Microcomputer Systems Corp.	
Illencik, Stephen, Republic Buildings Corp.	11
Imblum, Ray W., DEC-Santa Ana	
Infante, Frank, DEC-Maynard	LSI,11
Ingle, James, W.R. Grace, Cryovac Div.	11
Inglis, Geoffrey B., Rochester, Univ. of	8
Ingram, Al, DEC-El Segundo	VAX,11
Ingram, Larry, Hennepin County Library	11
Inman, Dale M., Phelps Dodge Corporation	11
Iobst, John, ANPA/RI	LSI,11
Irons, Lloyd, Tested Time Sharing Ltd.	
Isaacs, Fred P., Applilon, Inc.	VAX
Isbelle, Ted, California Res. & Tech.	VAX
Isidro, Ed, Crocker National Bank	11
Iverson, Robert, Jet Propulsion Lab.	LSI,11
Ivey, Wm. Max, Arizona State University	11
Jackson, Barry C., Lawrence Livermore Labs	LSI,11
Jackson, Bill, E & J Gallo	LSI,11
Jackson, Michael Q., British Columbia Railway	11
Jacobsen, Ralph, Data Enterprises	11
Jacoby, Rick, Informatics, Inc.	11
Jaffee, M., DEC-Santa Clara	LSI,11
James, John, Cetus Corporation	LSI,11
Jameson, Gail, Tektronix Inc.	
Jansen, Ronald, DEC-Maynard	8
Jayanthi, Sarma, ITT Telecommunications	11
Jekowski, John P., EG&G Inc.	LSI,11
Jellinek, John, Enterprise Companies	
Jenkinson, John, Mostek	11
Jensen, Kathleen, DEC-Tewksbury	VAX
Jesse, Richard H., Lawrence Livermore Labs	LSI,11
Jimenez, Tomas C., NAT Resources Mgmt. Center	11
Johnson, Brian, Jenson Compu Sys Ltd.	VAX,11
Johnson, Darryl, Interactive Mang. Sys. Inc.	LSI,11
Johnson, David A., Western Pacific Railroad	
Johnson, David K., Union Pacific Railroad Co.	11
Johnson, David R., US Postal Service	11
Johnson, Ellen, Cummins Service & Sales	11
Johnson, Eric, Jackson Laboratory	LSI,11
Johnson, Glenn B., DEC-Maynard	8,11
Johnson, M.A., W.R. Grace Company	11
Johnson, Mark, Alberta Rsh. Council	VAX,11

NAME	COMPUTER
Johnson, Paul D., California, University of	LSI,11
Johnson, Rowland R., Lawrence Berkeley Labs	VAX,11
Johnson, Stephan, DEC-Maynard	
Johnson, Susan, Los Alamos Scientific Lab.	LSI,11
Johnson, Ted, DEC-Maynard	
Johnston, Dick, E & J Gallo	LSI,11
Jones, Brian, Versatec	LSI,11
Jones, Byron M., Bucyrus-Erie Company	LSI,11
Jones, Frank R., Esso Chemical Canada	11
Jones, Hilary O., Sandia Laboratories	LSI,11
Jones, John F., Crystal Oil Company	11
Jones, Michael R., Boeing Computer Services	11
Jones, Ronald, N. West Development Corp.	
Jones, Terry, West Texas Equipment Co.	11
Jory, Roger, Lawrence Berkeley Labs	VAX,11
Joseph, Allen, Valco, Inc.	
Josephs, William H., Magnavox	
Joza, David W., Wyman-Gordon Company	11
Juhl, Daniel A., MDB Systems, Inc.	LSI,11
Jukes, Robert E., Naval Reserve Support	11
Julian, Carol, Valley News	11
Jurgens, Judy, DEC-Tewksbury	VAX
Kaczoqka, Peter, Time Share Corp.	LSI,11
Kafka, D.K., Chevron Research Company	LSI,11
Kafka, Hal, Measurex Corporation	VAX,11
Kahle, Ronald H., Hamilton/Avnet	LSI
Kahn, Ted, Cetus Corporation	11
Kanter, David B., IMS Computer Services	CTS, 8
Kapka, Joseph, Hambrecht and Quist	VAX,11
Kaplan, Roy, Aeronautical Research	VAX,11
Kapps, Charles, Temple University	LSI,11
Kareiva, Alan L., Dennen Steel Corporation	
Kasfelein, John, Taylor University	VAX,11
Katz, Robert, DEC-Marlboro	VAX,11
Katz, Stuart, Commercial Computer Serv.	11
Katzung, B.G., California, University of	11
Kauder, T.C., Rockwell International	LSI,11
Kavetsky, Edward M., GM Manufacturing Dev.	LSI,11
Kawakami, Ron, Academic Computer Service	VAX,11
Kawaye, Gary, California State University	11
Kawell, Leonard, DEC-Tewksbury	VAX
Kearney, James J., Walsh Associates Inc.	11
Keenan, George, E.I. Dupont De Nemours & Co.	11
Keeton, Andrew, MCC Powers	LSI,11
Keith, Stephen B., American Col. of Radiology	VAX,11
Keller, Tim, Singer Link Division	LSI,11
Kelley, Carl, Robert Cliff & Assoc. Inc.	LSI,11
Kelley, Michael S., Microcomputer Systems Corp.	11
Kelley, R.W., Sandia Laboratories	11
Kellogg, Martin, Los Alamos Scientific Lab.	11
Kelly, Chris, Lovelace Biom. & Env. Res.	LSI,11
Kelly, David S., Teledyne Controls	LSI,11
Kelly, Helen, San Diego State Univ.	VAX,11
Kelly, Joseph V., Wyman-Gordon Company	LSI,11
Kelsay, Jim, General Dynamics	LSI,11
Kelsey, Joseph M., Washington, Univ. of	VAX
Kendall, Burton, Measurex Corporation	LSI,11
Kendrick, N.S. Jr., Georgia Institute of Tech	8,11
Kenton, James, Arnar-Stone Laboratories	
Kenzie, Tony, Borden Chemical	11
Kerr, Douglas, Informatics/Programming	LSI,11
Kerr, George, Harbor Branch Foundation	11
Kesling, Wayne E., Monsanto Research Corp.	8,11
Kessler, Allan R., Admins Inc.	
Khan, Hassen A., Pullman Kellogg	11
Kibrick, Robert, Lick Observatory	8
Kiesel, Joanne, Free Communion Church	CTS
Killefn, Jeffrey J., Minuteman Tech.	
Kilroy, Robert, Naval Air Rework Fac.	11
Kilty, Nancy, DEC-Bedford	VAX

NAME	COMPUTER
Kimble, Graham, SPSS Inc Northfield Div.	VAX,11
Kimura, Arato, Behlen Mfg. Company	11
Kimura, Irene, Ecological Analysts	11
Kindschuh, Kevin, Crawford Service Co., Inc.	11
King, D. Wayne, B.C. Telephone Company	11
King, F., TRW-DSSG	VAX,11
Kinnan, Fred A., U.S. Army	LSI,11
Kinnear, Kim, DEC-Tewksbury	11
Kinzey, Glen D., Professional Datasystems	8,11
Kipple, Howard, New Mexico Junior College	
Kirkeng, Johan, Cytrol, Inc.	11
Kittell, Richard, Los Alamos Scientific Lab.	LSI,11
Klapatch, Ken, Computer Sciences Corp.	LSI,11
Klein, Gary, Computer Science Corp.	LSI,11
Kleinhammer, Pat, Rainbow Computing Inc.	LSI,11
Klingler, Val, EG&G Idaho, Inc.	11,15
Knight, Vernon E., U.S. Navy	11
Knipp, Randolph S., E.I. Dupont	8,11
Knox, Margaret H., Computation Center	11
Knudson, Richard, Georgia-Pacific Corp.	LSI,11
Kobine, Tony, Finar Sys. Limited	11
Kobrin, R.J., Mobil R & D Corporation	8,11
Kocsis, Dave, Intersil, Inc.	LSI, 8
Kodimer, Dennis, Terak Corp.	LSI,11
Koehler, James D., The Poise Company	11
Koeninger, R. Kent, California State	11
Koetke, Walter, Putnam/Northern Boces	
Kohlbrand, Janice, Dow Chemical Company	11
Kokenge, Dan, E & J Gallo	LSI,11
Koley, Ray, New Britain Housing	11
Kolm, David, Student	LSI,11
Komarmy, Louis, Children's Hospital of SF	LSI,11
Konopacky, John E., CESA #9 NEPL	11
Koolish, Daniel, Hartley Data Systems, Ltd.	8,11
Koolish, Ruth, Hartley Data Systems, Ltd.	8,11
Koolkin, Larry, Univ. of TX Medical Branch	11
Kopp, Ladd L., Pomona Valley Aviation	11
Korn, Granino, Arizona, University of	LSI,11
Kortesoja, Alan A., Manufacturing Data System	LSI,11
Koster, Ken, Teltone Corporation	LSI,11
Kowal, Al, Canadian Broadcasting	11
Kowalyshyn, John, Labatt Breweries	VAX
Kozlowski, Thomas, Los Alamos Scientific Lab.	LSI,11
Kraus, Joe, Crocker National Bank	11
Krcmar, Stan, Dissly Research	8,11
Kremer, John A.,US Steel Corporation	LSI,11
Kridle, Robert, California, Univ. of	LSI,11
Kristol, David M., Massachusetts Compu Assoc.	11
Kroepfl, David J., Lawrence Livermore Labs	LSI,11
Krol, M., Dept of Environment	11
Kroll, Geoffrey T., Illinois Bell Telephone	8,11
Kron, Peter, Software Ag. of N. America	VAX,11
Krones, Robert, Informatics/Programming	LSI, 8
Kroph, John, Seattle Pacific Univ.	LSI,11
Krueger, Wade, DEC-Lanham	11
Kruijk, Rob, DEC-Maynard	CTS,11
Krupp, James, Middlebury College	
Krupp, Michael, Children's Hospital of SF	11
Krysler, David, Monterey Peninsula Herald	CTS, 8
Kudrna, Ken, Georgia-Pacific Corp.	8,11
Kuff, Hal, Oceanic Enterprises, Inc.	11
Kulaga, Joseph, Argonne National Labs	LSI,11
Kulemin, Jean, Tri/Valley Growers	11
Kuo, Ivy, Lawrence Berkeley Labs	VAX,11
Kuramon, Aki, Lawrence Livermore Labs	LSI, 8
Kurasaki, Kerry, Second Source	8,11
Kurg, Haldi U., Miami Police Department	11
Kurjan, Philip	11
Kursewicz, Bernard, Mobil Research & Dev. Crp.	8,11
Kwok, Linda, Lawrence Berkeley Labs	VAX,11
Kyle, Charles F., Vanderbilt University	LSI,11

NAME	COMPUTER
Labiak, William G., Lawrence Livermore Labs	LSI
Labiniec, John, New Britain Housing	11
Labosky, Bonnie, SPSS, Inc.	VAX,11
Lacey, Donald E., Bendix Corp.	LSI,11
Lacroute, Bernard, DEC-Tewksbury	VAX,11
Lacy, Sharon, Lawrence Livermore Labs	LSI,11
Lahey, Terri, Fermilab	VAX
Laight, Peter D., British Columbia Railway	11
Landrum, G.J., Niosh	LSI,11
Lane, Robert L., DEC-Merrimack	
Lane, Stewart E., Advanced Data Systems, Inc.	LSI,11
Lane, Ted C., Bay Area Rapid Transit	VAX,11
Lang, Bob, Crocker National Bank	11
Lang. W.W., MacMurray College	11
Lange, Robert, LCS Minicompute Systems	8,11
Langer, Rudolph S.,Lawrence Livermore Labs	LSI
Langhofer, Laurn, UCLA Ctr for Health Science	LSI,11
Langston, Peter, Davis Polk & Wardwell	LSI,11
Lantz, Mel, Morgan Equipment	CTS,11
Lanz, Kai, Stanford University	LSI,11
Lanza, M.D., Chevron Research Company	LSI,11
LaPierre, Richard, Lawrence Berkeley Lab.	LSI,11
Larer, Gerald F., Occupational Skills Ctr.	11
Larson, James B., Rockwell International	VAX,11
Lash, Arthur, Nichols College	11
Laster, LaMar, Intermedics Inc.	11
Lathrop, Robert T., US Postal Service	
Lavaller, Richard, Alcan	LSI,11
Lauth, Norbert Jr., Public Broadcasting Serv.	VAX,11
Lawless, Thomas, U.S. Environmental	8,11
Lawrence, Thomas, Management Sciences	VAX,11
Lawson, W.V., Applicon Inc.	11
Lawyer, Bryan, Lawrence Livermore Lab.	VAX,LSI
Leacy, Ian, Cableshare	LSI,11
Learson, Jack, DEC-Merrimack	8,11
Leary, Bridget, Flavorland Industries	11
Leboss, Bruce, McGraw-Hill Publications	
Lebowitz, Jerry A., Fleet Analysis Center	11
Lech, Bernard, DEC-Maynard	
Ledbetter, Wallace, Georgia-Pacific Corp.	8,11
Lee, David, California, Univ. of	VAX,11
Lee, George, Bridgeport-Textron,Contrs	LSI,11
Lee, Larry J., System Industries	11
Lee, Lawrence, Citidata	LSI,11
Lefebvre, Lowell, Sytek Inc.	8,11
Lehotsky, Alan, DEC	VAX
Leino, Arthur, Stanford Linear	
Lellman, James, G.D. Searle & Company	VAX,11
Lemmer, Thomas G., Dalgety, Inc.	11
Lenhart, Donald, Caterpillar Tractor Co.	11
Lennon, William J., Lawrence Livermore Labs	8,11
Lenski, Laurence, Kaman Sciences Corp.	11
Lenz, S.J., Sandia Laboratories	LSI,11
Leonard, John, Aeronautical Res. Assoc.	
Lerner, Joe, Tennessee, University of	VAX
Lesser, Victor R., Massachusetts, Univ. of	VAX
Lev, Howard, DEC-Tewksbury	LSI,11
Levine, Barbara, Lawrence Berkeley Labs	VAX,11
Levine, Michael N., US Naval Weapons Ctr.	LSI,11
Levinz, Pam, DEC	
Levy, Harold, Applicon, Inc.	VAX
Levy, Henry M., DEC-Tewksbury	VAX
Lew, John P., DEC-Santa Ana	VAX,11
Lewis, Dave, LDS Church	LSI,11
Lewis, J. Otis, Philip Morris USA	8,11
Lewis, Mark F., Federal Aviation Admin.	11
Lewis, Reiko, Dewberry, Nealon & Davis	11
Li, Stella K.H., Dow Chemical Pacific	8,11
Lidster, Kenneth C., DISC	8,11
Liebold, Klaus, Salk Institute	11
Lieske, Jerald, RPGIG	8

NAME	COMPUTER
Lin, Dan, DEC-Maynard	11
Lincicome, Jack, First Int'l Services Corp.	11
Lind, James, Frank J. Seiler	VAX,11
Linder, Jack, Coors Porcelain Company	11
Lindsey, Randy, Cibar, Inc.	VAX,11
Linse, Donna, DEC	11
Lipcon, Eli, DEC-Maynard	11
Lipman, Mayer S., Roberts & Schaefer Co.	11
Lisee, Remi, DEC-Merrimack	8,11
Liston, Don, Lawrence Livermore Labs	LSI,11
Litchfield, Barbara J., Value Data Processing Inc.	11
Little, John, Instrumentation Lab.	8,11
Little, Skip, Woods Hole Oceanographic	VAX,11
Little, Todd, Science Applications Inc.	LSI,11
Littrell, David F., Baxter-Travenol Labs	11
Livingston, Jim, Lawrence Berkeley Labs	VAX,11
Lloyd, Brian, UCSD	8,11
Lloyd, Phil, TRW	11
Lockhart, Robert, Lawrence Livermore Labs	
Lockrey, Brian, ITT North Electric Co.	11
Lockwood, Jonathan, Harris Semiconductor	8
Lockwood, Mary	8
Loftin, Lavon, Millsaps College	8,11
Logg, Connie, Stanford Linear Accel.Ctr.	LSI,11
Loitz, Gary S., Watkins-Johnson Company	LSI
Loken, Stewart, Lawrence Berkeley Labs	VAX,11
Lomicka, Roy, DEC-Maynard	
Long, Joel, United Industry, Inc.	
Long, A.J. II, South Central Bell Tel Co.	LSI,11
Long, Dick, Computer Synergy	LSI,11
Lopez, Hector, Cerveceria Cuauhtemoc SA	
Louis, Raymond, Lawrence Berkeley Lab.	LSI,11
Loveland, Richard A., DEC-Maynard	
Lowe, Mike, E & J Gallo	11
Lowenstein, Carl, Marine Physical Lab.	LSI,11
Lowry, Edward, DEC-Maynard	
Lubell, Bradford, California, Univ. of	8,11
Luca, John, TRT Telecommunications	VAX,11
Luhring, Hank, Solano Cty. Sup. of Schools	11
Lundquist, Arthur, US Air Force Systems Com.	LSI,11
Lyddon, Sandra L., Chevron Oil Field Rsh. Co.	VAX,11
Lyman, John, Lawrence Berkeley Lab.	LSI,11
Maas, Margaret, DEC-Maynard	
MacGibbon, George A., CBL Auckland Limited	11
Mack, Ronald M., Tektronix, Inc.	11
Mackichan, Alan, KBM, Inc.	LSI
Madrian, Jost, Salt Lake City School	11
MaGee, F.I., Sandia Laboratories	11
Magidson, William, PLM, Inc.	11
Mahon, George, Ford Aerospace & Comm.	VAX
Mallet, Charles, DEC-Maynard	
Malone, James, General Connecticut	11
Malone, Ray, Xerox Corporation	11
Manelski, Denis, Mints	8
Mangiarelli, Ronald R., U. S. Air Force	11
Mann, Barbara, TRW Defense & Space Sys.	11
Mann, Emory H. Jr., General Electric Co.	11
Manter, Walter, DEC-Marlboro	8,11
Manton, John, Mostek	VAX,LSI
Marcek, Steven, DEC-Oakland	VAX
March, Pam, Liocs Corporation	LSI,11
Marcil, Mike, DEC	11
Marek, John M., Lebus Data Centers Inc.	CTS, 8
Marino, Richard A., Data Processing Design	VAX,11
Markley, Richard L., Corning Glass Works	11
Marks, J.P. Telesensory Systems, Inc.	LSI,11
Marler, Kevin M., Union Pacific Railroad Co.	11
Marschke, Vern, NCA Corporation	11
Marsh, Richard, Tulasi, Ltd.	CTS,11
Marshall, Ted, Data Processing Design	LSI,11

NAME	COMPUTER
Martin, Arvid L., General Motors Rsh.Labs	VAX
Martin, Carol A., TRW Systems	11
Martin, Jerome A., First Computer Corporation	LSI,11
Martindale, Jerry L., Joy Machinery Company	11
Martinelli, Jack, Cetus Corporation	11
Martinez, Sabiniano, Equitec Financial Group	11
Marvel, Mary, General Motors Res. Lab.	11
Marx, Joel, Varian Graphics	LSI,11
Mason, G.R., Bonker Ramo	VAX
Mastrandrea, Joseph, Mt Sinai Ins. Comp. Sci.	LSI,11
Matejcek, Paul, Varian Mat	11
Matlack, Elizabeth, DEC-El Segundo	VAX,11
Matsumoto, Scott, Lawrence University	11
Mattheiss, Paul, Suntech, Inc.	11
Matthews, Thomas, Ricks College	
Mattison, Bonnie, California, Univ. of	11
Maurer, Hank, DEC-Marlboro	VAX
Maurice, Michael, Reel Trophy, Inc.	LSI,11
Mayer, Tim, Cibar, Inc.	11
Mayfield, Mike, Data Processing Design	VAX,11
Mayras, Shell Francaise	11
Mazzari, Darrel A., Color Corp. of America	VAX, 8
McCarthy, Mike, DEC-Colorado Springs	
McCarty, Jack, Inslaw	11
McCleary, Ron, Harding College	11
McClurg, William, United Technologies	LSI,11
McCormick, John, Alberta Government Tele.	LSI,11
McCormick, R.B., Atlantic Richfield Co.	11
McCoy, Daniel, Informatics Inc.	11
McCray, Wm. Arthur, DEC-Tewksbury	
McCue, Kevin, TRW DSSG	VAX,11
McDaniel, Larry, California, Univ. of	VAX,11
McDaniel, R.S., Alberta Research Council	LSI,11
McDonald, Joyce, World Book-Childcraft	LSI,11
McDonald, Lenard, Rockwell International	11
McDonough, John, New York Telephone Co.	11
McDonough, Martin F., Value Data Processing	11
McFerrin, Paul, Bell Telephone Lab.	LSI,11
McFerrin, Steven, Bendix Field Eng.	LSI,11
McGhie, Dennis, SRI International	8,11
McGinnis, Gerald, Argonne National Labs	VAX,11
McGinnis, Kenneth, Pennsylvania, Univ. of	11
McGlinchey, James, Fischer & Porter Systems	LSI,11
McGraw, Joseph, Bendix Epid	8,11
McGuinness, James, Schering Corp.	
McIlvaine, Jim, Bridgeport-Textron Control	LSI,11
McIntosh, Stuart, Admins, Inc.	
McIntyre, Tom, DEC-Maynard	8,11
McKenna, Michael, Time Share Corporation	LSI,11
McLawhon, George B., Hallibueton Services	LSI,11
McLean, Donald, MacNeal & Schwendler	VAX
McLeod, Jim, American Sign & Indicator	LSI,11
McMacken, Dennis, U.S. Geological Survey	11
McMichael, Patrick, Wabco Union Switch & Sign	11
McMillan, Guy G., California, Univ. of	
McMillin, Leslie S., Energy Ent of Denver Inc.	
McNaughton, Bruce, DEC-Nashua	
McNeal, Sharon, EG&G Inc.	LSI,11
McNeish, Andy, Canadian Broadcasting Corp.	8,11
McParland, Charles, Lawrence Berkeley Labs	LSI,11
McPhaden, Christine, California, Univ. of	11
McPharlin, Tom, Systems Control Inc.	VAX,11
McRitchie, Bruce, Bridge Brand Food Service	
Meador, M.D., Northwest Outdoor Stores	CTS,11
Mears, Donald, Minnesota, Univ. of	LSI,11
Medlin, Terry P., NIH INIDR	LSI,11
Meese, Robert, DEC-Marlboro	VAX,11
Mehren, D.J., Duval Corporation	VAX,11
Mehta, Gautam, Westinghouse Electric	11
Melancon, Jason, Standard Structures, Inc.	11
Mellor, N.P., Smithkline & French Labs	11
Melnick, Jerry, DEC-Marlboro	
Meloche, Gilles, National Research Council	LSI,11
Mena, Harry, Algonquin College	
Menk, Peter, A.C.U.	
Meny, Susan, Connecticut General Life	11
Merchant, Michael J., EG&G Inc.	11
Merrell, Greg, Allen Bradley Company	VAX,11
Merrill, Roy, Cetus Corporation	LSI
Merriman, Michael, Applied Information Dev.	LSI
Merritt, Philip E., Hughes Research Labs	LSI,11
Mess, George, Kalium Chemicals	
Michael, Greg, DEC-Colorado Springs	VAX,11
Michelson, Randy, Lawrence Berkeley Labs	
Miles, Ken, Nordata	VAX,11
Mileski, Jack, DEC-Tewksbury	
Miller, Christopher, Herman Miller, Inc.	11
Miller, Dan G., Los Alamos Scientific Lab.	VAX
Miller, David, GTE Sylvania	11
Miller, Jay L., GTE Sylvania, ESG-WS	11
Miller, Jerry, Progressive Management	11
Miller, John A., Online Data Processing	VAX
Miller, Randolph R., System Development Corp.	VAX,11
Miller, Robert A., Lockheed Missiles & Space	
Miller, Ross W., Online Data Processing	VAX
Mills, Vincent T., San Francisco State Univ.	VAX,11
Mitchell, Jeffrey W., DEC-Marlboro	
Mitchell, Terrell K., DEC-Merrimack	VAX,11
Mitchell, William, AFRRI	LSI,11
Mitsch, Stephen, Systems Consultants, Inc.	11
Mock, Melissa, Crocker National Bank	11
Moe, Christine, Stanford University	11
Moersdorf, Gerard B., Software Results Corp.	LSI,11
Moffa, Roy J., DEC-Maynard	LSI
Mohan, Sat, TIW Industries Ltd.	11
Monaghan, Terry, Borden Chemical	11
Mond, Lawrence A., Anstat, Inc.	VAX,11
Monia, Charles, DEC-Tewksbury	VAX
Montague, Bruce, School of Aerospace Med.	8,11
Monterubio, Fred, Bridgeton Hotels, Inc.	LSI,11
Moody, K.B., W.R. Grace & Company	11
Moore, James, Stanford University	LSI,11
Moore, Raymond A., Hyde Park Chemical Corp.	11
Moore, Robert D., Scripps Inst Oceanography	8
Moore, Shirley, Crocker National Bank	11
Moore, Terrence, Stanford Univ. Med Center	11
Moothart, Eric, Data Processing Design	VAX,11
Morris, Robert, E.R. Squibb & Sons, Inc.	11
Morrison, James, DEC-Marlboro	VAX
Morton, Jerry R., Chemineer, Inc.	
Morton, John, B-N Software Research	11
Mounteer, William D., Tested Time Sharing Ltd.	
Mountjoy, David R., Bendix Corporation	
Moy, Lidia, California, Univ. of	CTS
Moyer, Richard, Stanford Linear	
Moyle, Gordon, New South Wales, Univ. of	8,11
Mucci, John, DEC-Marlboro	VAX,11
Mueller, Edward, Logicon, Inc.	
Mueller, Martin R., Informatics Incorporated	LSI,11
Mueller, Michael, Rockhurst College	8,11
Mundy, Linda, Cetus Corporation	11
Murtland, Lori, Creative Systems	11
Mussetter, Robert, Mussetter Reality, Inc.	8
Mustain, Charles W., Berea City Schools	11
Mustain, Richard D., Philip Morris USA	8,11
Myers, Jack, Stanford Linear Accel. Ctr.	LSI,11
Nace, Roger, DEC-Englewood	VAX,11
Nadel, Lesta, Lawrence Berkeley Labs	VAX,11
Naglee, John, American Tel & Tel Co.	11
Napkori, Joseph, Safeguard Business System	11
Natowitz, Jerry, Cincinnati, University of	11

NAME	COMPUTER
Naylor, Mark, General Foods Ltd.	11
Naylor, William, Gannett Co., Inc.	11
Neeland, James, Hughes Research Labs	VAX,11
Neill, Jim, Bell Canada	11
Neilson, David, Lawrence Livermore Lab.	LSI,11
Nelson, Brian, Toledo, University of	11
Nelson, Dick, Domain Industries, Inc.	
Nelson, Gary, American Management	VAX,11
Nelson, Gregory, Ford Aerospace & Comm Corp.	VAX
Nelson, R.L.	8,11
Neuberger, Kent, Key Air Conditioning Co.	11
Newcombe, Pete, York Graphic Services	LSI,11
Newell, Timothy, Boebinger Agency, Inc.	VAX,11
Nicholas, Pete, Lawrence Livermore Lab.	LSI
Nichols, Herb, DEC-Tewksbury	11
Nichols, Lee H. III, E. I. Dupont	8
Nichols, Tom, DEC-Marlboro	VAX,11
Nicholson, Paul R., Battelle, Pacific NW Labs	11
Nickles, John C., California, Univ. of	LSI,11
Nicol, David, Pan American Technical	11
Niday, James, Lawrence Livermore Lab.	8,11
Nieh, Luther, General Electric Co.	
Nighbor, Bill, Hal Systems Corp.	11
Nirenberg, Isabel, Space Astronomy Lab.	LSI,11
Noble, Douglas R., Drug Enforcement Admin	11
Noble, Donna, Kitchell Contractors, Inc.	11
Nordby, David H., G.D. Searle & Company	VAX
Norman, Jerry, United States Computers	11
Normington, Dave, NCA Corporation	11
Norris, Kathryn S., DEC-Tewksbury	VAX
North, Ken, Hughes Helicopters	11
Noyce, William, DEC-Merrimack	
Nunnally, John, Harding College	11
Nusbaum, Robert, DEC-Maynard	LSI,11
O'Connell, Janet, Business Inf Systems Inc.	
O'Connell, Joseph P., Business Inf Systems Inc.	
O'Reilly, J. Michael, Mini Data Systems Inc.	8,11
Oakes, William Jr., Los Alamos Scientific Lab.	VAX
Oakland, Frederic, Cutler Hammer Inc.	LSI,15
Oconnell , John, Datatrol, Inc.	LSI,11
Odom, Warren E., Good News Book Store	LSI,11
Okada, So, ASR Corporation	LSI,11
Olien, David M., Iowa State University	VAX,11
Olmstead, Kent, Los Alamos Scientific Lab.	VAX
Olsen, Larry, G. D. Searle	VAX
Onuma, Lambert, DEC-Oakland	
Oothoudt, Michael, Los Alamos Scientific Lab.	VAX,11
Oppenheimer, Jim, Lawrence Livermore Lab.	LSI,11
Oppermann, Curt, Missouri Pacific Railroad	11
Osborne, Stan, DEC-San Francisco	8,11
Oskirko, Maryann, DECUS-Marlboro	
Ostlund, James J., California, Univ. of	LSI,11
Otte, Mike, American Telecom	11
Ovalle, Alejandro, Comision Fed de Electric	VAX,11
Overby, D. Russel, Oak Ridge National Lab.	8,11
Overton, Mack, U.S. Food & Drug Admin.	LSI,11
Owen, Jan, DEC-Santa Clara	
Padwa, Stephen, Brookhaven National Lab.	VAX,11
Page, Calvin, Georgia-Pacific Corp.	8,11
Page, William, DEC-Tewksbury	VAX
Pajerski, Rose, NASA Goddard Space Flight	VAX
Palladino, Robert A., Schering Corporation	11
Palmer, Dean L., GTE Sylvania	VAX,11
Palmer, Thomas, Illinois Bell Tel Co.	8,11
Palmerston, David W., DEC-Oakland	11
Papajcik, Ronald, Horsburgh & Scott Co.	
Park, Denise, Mischer Enterprises, Inc.	11
Parker, David, Duke University	LSI,11
Parker, Ron, Canadian Hydrographic Ser.	11

NAME	COMPUTER
Parrish, Donna B., Fluor Corporation	LSI,11
Parson, Bob, NP Time Sharing Service	VAX,11
Parsons, Gilbert, Parsons Brake Service	
Parsons, Jim, Naval Air Rework Facility	11
Passafiume, Joseph, DEC-Maynard	LSI,11
Patel, J.N., Budd Company	11
Patterson, Gary, Maritime Administration	LSI,11
Patterson, Robert, Litton UHS	LSI,11
Paul, Roger, American Sign & Indicator	LSI
Paulk, J.W., Daniel International Corp.	11
Paulson, Joe, Domain Industries, Inc.	
Paulson, Boyd, Standard University	LSI,11
Payne, Mary, DEC-Maynard	8,11
Payne, Richard, Woods Hole Oceanographic	VAX
Payne, William L., EG&G Inc.	11
Pearson, James, NWC Federal Credit Union	LSI,11
Pearson, Larry, DEC-Maynard	LSI,11
Pearson, Robert L., R.A. Hanson Company	
Pearson, Stanton, DEC-Maynard	
Pech, Bernard, Lawrence Berkeley Lab.	VAX,11
Peck, Robert C., California, Univ. of	LSI,11
Peckoff, Barry, Philip Morris Intl.	8,11
Peebles, James E., Arizona, University of	11
Pekin, Dian, DEC-Bedford	K/LSI
Pellegrino, Harry, Naval Avionics Center	11
Pellerin, Roy, Teltone Corp.	11
Pelletier, Arthur, Dept. of National Defense	11
Pensak, David A., E.I. Dupont De Nemours	VAX
Pensak, Lois B., Widener College	11
Pepin, Paul S., Delco Products Div. GMC	VAX,11
Perk, Harry, Dow Chemical Co.	11
Perry, Dennis, Los Alamos Scientific Lab.	LSI,11
Perry, Thorne, Baxter Travenol	11
Pessin, J.S., Bunker Ramo Corp.	VAX,11
Peters, Carol, DEC-Tewksbury	VAX
Petersen, Jay H., ESL, Inc.	LSI,11
Peterson, Curtis, Multi-List	VAX,11
Peterson, James C., Environmental Protection AG	11
Peterson, Roger L., Lawrence Livermore Labs	LSI
Pettet, Mark, Tektronix, Inc.	LSI,11
Pfeifer, Larry L., Signal Technology Inc.	VAX,11
Phalen, Thomas D., U.S. Food & Drug Adm/EDRO	11
Phelps, William, Francis and Nygren	11
Phillips, L.N., Transport Canada	11
Pickering, Howard, Tested Time Sharing Ltd.	11
Pickford, Terry, ABC-TV	
Picott, William, DECUS-Marlboro	
Piel, Gary P., United States Air Force	11
Piela, Ron, Illinois Tool Works	11
Pierce, D.M., Sandia Laboratories	8,11
Pigman, Richard, DEC-Maynard	
Pine, B.J., Nat'l Comp Performance Co.	LSI,11
Pinfield, Edward, Colorado, Univ. of	LSI
Pitliuk, Jaime, Security Industry	11
Pitluck, Samuel, Lawrence Berkeley Lab.	11
Pollack, J. Eric., DEC-Tewksbury	
Ponting, Bob, Versatec	VAX,11
Ponzio, Gloria A., Mini Computer Supplier	CTS
Poolman, Robert, Santa Barbara Cty. Mental Hlth.	11
Poon, Bobbi, Ford Aerospace & Comm Corp.	VAX
Poppendieck, Mary, 3M Company	LSI
Porz, Donna, Arizona State University	11
Post, Craig, Naval Weapons Center	VAX,11
Post, William, Computer Science Corp.	LSI,11
Poust, Roy, Georgia-Pacific Corp.	LSI,11
Poutissou, Renee, Montreal, Univ. of	11
Powell, Bruce, California State Univ.	LSI,11
Powell, James, Stanford University	LSI,11
Powell, John, Nat'l Inst. of Health	LSI,11
Powell, Ricky H., HQDA Taggen (DAAG-PLS)	11
Powers, David, Cadillac Motor Division	LSI,11
Prather, John, Sun Shipbuilding Co.	11

NAME	COMPUTER
Pratt, Orville, Lovelace Biom.& Env. Res.	VAX,11
Preston, David, Hutchinson Community	11
Pribadi, K.S., Compuguard Corporation	LSI
Priborsky, Tony, Arizona State University	VAX,11
Prinzivalli, Pete R., Measurex	VAX
Provost, Thomas, MIT	LSI,11
Puls, James, Chicago, University of	11
Pulsipher, D.C., Sandia Laboratories	11
Purdue, Clint, Sandia Laboratories	11
Puttress, John, Applied Dynamics Int.	VAX,11
Quigley, Donald, Naval Underwater System	VAX
Qureshi, Muzaffar A., Population Council	11
Racer, C.W., Chevron Geophysical Co.	VAX,11
Racklyeft, David, General Motors	11
Radcliffe, Clark, California, University of	8
Radford, Kenneth, DCA Reliability	11
Ragan, Logan, Nu. West Development Corp.	
Ralston, Carl, DEC-Marlboro	VAX
Ramsey, Newell R., EG&G, Inc.	LSI,11
Randall, Ron, DEC-Marlboro	VAX,11
Rappley, David, Arkansas Gazette	8,11
Rasmussen, Gary, Gary S. Rasmussen & Assoc.	11
Rasmussen, Warren, San Francisco State Univ.	11
Rasted, John, JTR Associates	11
Rasted, Susan, Software Dynamics Inc.	11
Ravo, William G., Naval Underwater Systems	VAX
Rawlinson, S., TRW-DSSG	VAX
Reardon, James M., Fluor Corporation	LSI,11
Reder, Thomas, Dow Chemical Co.	
Redmon, Robert, United Catalyst	11
Reece, Randy, Multi-List Inc.	
Reese, C.H., Chevron Research Company	LSI,11
Rehbein, Carl, DEC-Maynard	VAX,11
Reibling, Lyle, Lear Siegler, Inc.	LSI,11
Reicheld, Harold, Dofasco	LSI
Reilly, John, CDI Oakland	
Reinap, Ginny, DEC-Maynard	
Reiners, Edward, Philip Morris, Inc.	8,11
Reinkemeyer, J., GSD/IBM	11
Reisert, Christopher, Manufacturers Hanover TRS	11
Reite, Martin, Colorado, University of	LSI,11
Remedios, A.J., Chevron Research Company	LSI,11
Renneke, David R., Augustana College	11
Reno, Douglas, Abbott Laboratories	11
Renta, Charles, Pertec	LSI,11
Restagno, Dominick, Manufacturers Hanover	VAX,11
Reynolds, John D., System Development Corp.	VAX
Reynolds, Walter E., Applied	8,11
Rhew, Ki-Won, El-Jay, Inc.	VAX,11
Rhodes, Owen F., Florida Computer Inc.	VAX,11
Rich, Emil, American Broadcasting Co.	LSI,11
Rich, Steve, Dewberry, Nealon & Davis	11
Richardson, Wendell, Bingham Mechanical	11
Richmond, Rick, Pikes Peak Regional Library	11
Richter, C.E., Western Electric Company	LSI,11
Rickson, Bryce, Vartron Corp.	CTS,11
Riebs, Andrew, DEC-Merrimack	11
Riel, David A., Fed. Bureau of Investigation	11
Rieman, Gerald A., Dayton Data Processing	LSI,CTS
Riley, Patrick, United States Computers	11
Riley, Robert, Western Electric Co., Inc.	11
Riquet, Danielle, SII	11
Risch, Douglas, Lawrence Livermore Labs	LSI
Risso, William L., National Inst. of Health	11
Ritchie, David, Fermi Nat'l Accelerator	
Ritchie, R.W., Washington, Univ. of	VAX
Ritenour, Robert P., EG&G Inc.	11
Rix, Alfred, Tacoma News Tribune	11
Roach, Hal, Cerritos College	

NAME	COMPUTER
Roads, Curtis, Compter Music Journal	VAX,11
Robert, Ron, US Fleet Leasing	11
Roberts, Carol, Nat'l Transportation	11
Roberts, James L., DCA Reliability Labs	11
Robertson, Larry, R.S. Association	VAX,LSI
Robinson, Gordon, Charles S. Lewis & Co. Inc.	11
Robinson, Mike, United Computing Systems	11
Robinson, Richard V., US Department of Labor	VAX
Robison, Frank H., Daytronic Corp.	LSI,11
Rodean, F. Glynn, Plymouth State College	8,11
Rodgers, David, DEC-Tewksbury	VAX
Rodgers, Tom, Crocker National Bank	11
Rogers, George H., Canada College	11
Roland, David, Informatics, Inc.	11
Romanoff, Robert, National Inst of Health	LSI,11
Roode, R. David, Informatics PMI	LSI,11
Root, Stephen, DEC-Maynard	8
Ropchan, Wally, Stanford University	LSI,11
Ropp, Patricia, Hood College	11
Rose, J.D., Academic Computing Service	LSI,11
Rose, Stanley, Bankers Trust Co.	11
Rosen, Daniel, Transcomm Data Systems	11
Rosenbluh, Kathy, Chicago, University of	11
Ross, Carlos, Grupo Cydas	11
Ross, Ken, Ross Systems, Inc.	
Rothe, Edward, NASA-GSFC, Infom. Analysis	LSI,11
Rothermel, John G., TRW DSSG	VAX,11
Roush, Ellard T., GIPD	11
Roux, Gerald, Boeing Company	11
Roveda, John, Charlie Systems Inc.	CTS, 8
Rowan, Willian H., California, Univ. of	VAX,11
Ruch, Peter, United States Air Force	LSI,11
Rudkin, Ralph, CTEC, Inc.	11
Rudy, Jeffrey, DEC-Merrimack	VAX,11
Ruff, Curtiss, Motorola	
Runyon, John, Philip Morris International	LSI,11
Runyon, Marilyn	LSI
Russ, Allen E., Tulane University	LSI,11
Ryan, Joe, NALC 203C U.S. Navy	VAX,11
Ryan, Tim, University Data Systems	LSI
Rynes, Philip, Argonne National Lab.	LSI,11
Ryshpan, Jonathan, Microform	LSI,11
Saabes, M., TRW-DSSG	LSI,VAX
Sabetta, Donna, EDS Nuclear Inc.	CTS,11
Sadler, Dan, Compuserv	8,11
Sadler, Lisa, Compuserv	8,11
Sadofsky, Mike, DEC-Maynard	
Salas, George R., IVIC	
Salmon, Donald, Vernay Laboratories, Inc.	LSI,11
Saloky, Al, DEC-Merrimack	11
Saltzberg, Steven A., Louisville, Univ. of	
Samson, Paul, Teltone Corp.	11
Samuel, Dan J., Digital Pathways, Inc.	LSI,11
Sanborn, Robert, National Bank of Detroit	11
Sanders, Debbie, S&H Computer Leasing Inc.	
Sanders, Harry, S&H Computer Leasing Inc.	LSI,11
Santon, Lee, Ontario Cancer Institute	LSI,11
Sapp, Ed, DEC-Maynard	LSI,11
Sarbin, Theodore R., DEC-Santa Ana	8,11
Sasseen, Doug, National Food Processors	LSI,11
Sather, Louis, Wisconsin,University of	VAX,11
Satterlee, Duane, International Harvester Co.	LSI,11
Saylor, Jean, System Development Corp.	11
Scandora, Anthony, Science Applications Inc.	LSI,VAX
Scanlon, Harold, EG&G Washington Analytic	11
Scardino, Shirley, DEC-Santa Clara	LSI,11
Scarloss, Walter, Corning Glass Works	LSI,11
Schachter, Ted M., Los Alamos Scientific Lab.	LSI,11
Schaefer, Ronald, DEC-Tewksbury	VAX,11
Schainbaum, Julius, Smith Kline & French Labs	8,11

NAME	COMPUTER
Scheckter, Michael, Hughes Aircraft Co.	VAX,11
Scheremeta, James T., Boeing Computer Serv.	VAX,LSI
Scherrer, Deborah, Lawrence Berkeley Lab.	VAX,11
Schick, William, Fairleigh Dickinson Univ.	8,11
Schioeder, Esther, Lawrence Livermore Labs	VAX
Schloerb, Paul, Kansas, University of	11
Schlumpf, Michael A., Bally Mfg. Corp.	LSI,11
Schmauder, Philip, Dept. of Air Force	8,11
Schmeichel, Robert, DEC-Maynard	11
Schmidt, David, Management Science Assoc.	VAX,11
Schneider, Richard, DEC-Maynard	
Schneider, W.S., Bunker Ramo Corp.	VAX,11
Schoenberger, Annette, MacMurray College	11
Schoenberger, R.J., MacMurray College	11
Schoeppe, Henry, Sandia Laboratories	LSI,11
Schoeppner, Mike, Bendix Kansas City	11
Schofield, L.N., General Dymanics/WDSC	
Scholz, Warren, Tennessee Gas Pipeline	11
Schomberg, Paul, NCA Corporation	11
Schopp, Ken, EDS Nuclear Inc.	VAX,11
Schrader, David, 3M Company	11
Schramm, Michael, H.S. Crocker Co., Inc.	LSI,11
Schrick, Dale P., Meda Sonics	11
Schriesheim, Jeffrey, DEC-Maynard	
Schroeder, James, Battelle Northwest	VAX
Schueler, Rich, Crocker National Bank	11
Schurr, David, Kaiser Hospital	8,11
Schurter, James L., MacMurray College	11
Sconce, William J., Industrial Specialties	11
Scott, Wayne, Vernay Laboratories Inc.	11
Scotten, Arthur, Comm.and Computer Service	11
Seefeldt, B., TRW-DSSG	VAX
Seifert, William, Los Alamos Scientific Lab.	LSI,11
Selim, Lydia, Crown Zellerbach	11
Seminario, Carlos, Georgia Tech	8
Senechal, Paul, Dofasco	LSI
Sergejew, Alex A., Auckland Sch.of Medicine	8,11
Settle, Dominic, Fotomat Corporation	11
Seufert, Steven, DEC-Maynard	LSI,11
Severyn, John, Lawrence Livermore Labs	VAX,11
Seward, Robert C., Pertec	8,11
Sexton, Jim, Amcor Computer Corp.	11
Seymour, Richard, Washington, Univ. of	LSI,11
Shacter, Stewart, First National Bank of St.Paul	11
Shah, S., DEC-San Francisco	VAX,11
Shannon, Enoch J., California State Univ.	11
Shanzer, Herbert, DEC-Maynard	
Sharp, Robert, Inco, Inc.	LSI,11
Sharpe, Stan, Cincinnati Milacron Inc.	11
Shartle, Harold F., US Navy Ships Parts Cont.	11
Shaulis, Roger, Philip Morris U.S.A.	8,11
Shear, Robert D., Shear Development Corp.	LSI,11
Shelton, Rick, Commercial Computer Sys.	8,11
Sherborn, Dennis, Zimmerman Metals, Inc.	11
Sheridan, Kim A., Interactive Info. Systems	11
Sherman, Thomas, DEC-Tewksbury	11
Sherrill, Nathaniel, US Geological Survey	CTS, 8
Sherrod, Phil, Vanderbilt University	VAX,11
Shindell, Dav, DEC-Santa Ana	LSI,11
Shirley, Fred, DEC- Merrimack	
Shlaer, Sally, Lawrence Berkeley Labs	11
Shpiz, Leo, DEC-Merrimack	
Shropshire, K. David, USAF-Air Training Com.	VAX,11
Shurman, Gary, Professional Datasystems	8,11
Shurtleff, Robert D., DEC-Merrmiack	11
Shuster, Joseph, World Book Childcraft	LSI,11
Sidenblad, Paul, System Development Corp.	VAX
Siegel, Albert L., Battelle Columbus Labs	VAX
Siegenthal, Martin, San Francisco State Univ.	VAX,11
Siemens, Phillip, Menol Computer Associates	8,11
Siftar, Gary, Cisco	LSI,VAX
Sikes, Dale, Lockheed Missiles & Space	LSI,11
Sills, Andrew, UCLA Ctr. for Health Sci.	LSI,11
Simich, Ellen, DEC-Merrimack	
Simmons, Frank, Accurate Data Systems, Inc.	CTS,11
Simmons, Louis, Citibank	8,11
Simmons, Stephen, US Leasing Corporation	VAX,11
Simms, A., DEC-Santa Clara	LSI,11
Simon, Randolph F., US Navy Astronautics Grp.	8,11
Simpson, Lawrence, Michael Reese Hospital	VAX,11
Sisson, Norwood, Arizona State University	8,11
Sivertson, John N., Johnson & Johnson	11
Skalabrin, Val, Second Source	8,11
Skelton, Claudia, DEC-San Francisco	VAX,11
Skelton, Steve, DEC-Maynard	VAX
Skoog, Clifford, Sandia Laboratories	LSI,11
Skorodinsky, Robert G., System Development Corp.	11
Skret, Dan, Vindicator Corp.	CTS, 8
Slaski, Kenneth, Schering-Plough Corp.	
Slezak, Ken, EMR Telemetry	LSI,11
Small, Steven L., Western Electric Co.	8,11
Small, W.H., Mobil Research & Dev. Corp.	11
Smith, Alan B., Dow Chemical	11
Smith, Albert, USDE Bonneville Power	11
Smith, Barry, Oregon Minicomputer	8,11
Smith, Christopher, EG&G Inc.	11
Smith, David A., DEC-Blue Bell	LSI,11
Smith, D.R., Toronto, University of	LSI,11
Smith, Gary, Area Two Educational	11
Smith, Jerry, Education & Res.Comp.Ctr.	VAX,11
Smith, Joseph, DEC-Maynard	
Smith, Mickey, US Air Force	8,11
Smith, Paul, Naval Air Rework Facility	11
Smith, Raymond V., New England Nuclear	VAX
Smith, Sandy, Area Two Educ. Computer Ctr.	11
Snapper, Arthur, Western Michigan Univ.	8
Snawder, Paul T., Santa Clara County	11
Sneddon, T.W., Sandia Laboratories	
Snipes, H., TRW-DSSG	VAX
Snively, C.P., IBM/GSD	11
Snow, David, DEC-Tewksbury	11
Snow, Scott, Mini-Compu Business Applications	
Snyder, Frank D., Westinghouse Elec Corp.	LSI,11
Sobek, Walter, Metropolitan Sanitary	VAX,11
Soehomonian, Vatche, DEC-Oakland	
Soivenski, Mitchell S., Liberty Mutual Ins. Co.	11
Solo, Beverly, Merritt College	11
Solomon, David, Riverdale Country School	VAX
Somes, Austin H., NASA Ames Research Ctr.	VAX,11
Sopka, Elisabeth, Sitear Development Corp.	
Sosa, Arturo T., Comision Fed. de Electricity	11
South, Charles, Hughes Research Labs	VAX,11
Spann, James, Lawrence Livermore Lab.	LSI
Spavin, Barbara, DG&G Inc.	11
Speake, Tom, DEC-New York	
Spears, Bill, Baylor College of Medicine	11
Spear, David, Systems Consultants, Inc.	VAX,11
Spelfogel, Carol, Shawmut Bank of Boston	11
Spence, Martha L., DEC-Maynard	VAX,11
Spence, Robert, DEC-Maynard	LSI,11
Spinek, Ronald, DEC-Maynard	VAX,11
Spitz, Charles, DEC-Tewksbury	11
Sporn, Patricia, FDA Bureau Medical Device	11
Sprague, William, General Motors Corp.	LSI,11
Springer, Allen, Amos Press	11
Sprouse, Gene, Rainbow Computing	
St. Armour, Jerry, Union-Tribune Publishing	11
Stafford, Damon, Lovelace Biom.& Env.Res.	LSI,11
Stafsudd, Jacquie, Hughes Research Labs	VAX,11
Stagg, David, Yale Univ. Sch. of Medicine	LSI,11
Stamerjohn, Ralph, Monsanto	LSI,11
Stanfill, R. James, Swedish Hospital	11

NAME	COMPUTER
Stange, Whitey, Computer Applications Corp.	CTS
Stanzione, Dennis, Export Credit Corporation	
Stark, Duane, Doubletree, Inc.	11
Starkey, Jim, DEC-Merrimack	VAX,11
Steele, Craig S., Harvard University	VAX
Steele, Robert, Inco, Inc.	LSI,11
Steenbergen, Robert B., EG&G Inc.	11
Steinberg, Daniel, SRI International	11
Steingart, Alec, Republic Nat'l Bank of NY	11
Stepanek, Steven, CSU, Northbridge	LSI,11
Stern, David M., Research Systems Inc.	8,11
Sternick, Wendy S., Atlantic Richfield Co.	11
Stevens, Edwin, EMDA, Inc.	LSI,11
Stevens, Kenneth, USAFSAM/BRP, Brooks Air	8,11
Stevens, T.C., Toronto, University of	LSI,11
Stewart, E.H., Rectec, Inc.	LSI,11
Stewart, Jim, British Columbia Bldg. Cor.	
Stewart, John, Van Ert Electric Co., Inc.	CTS,11
Stewart, Russell, Dow Chemical Co.	8,11
Sthultz, Michael R., Sunrise Hospital	LSI,11
Stinson, Murray, Scientific Placement Inc.	11
Stockdill, James W., US Navy Ships Parts Cont.	11
Stockley, C.H., Sandia Laboratory	11
Stodden, Joan, Varian Graphics	VAX,11
Stodghill, Walter, Motorola	
Stoklosa, Henry, Du Pont Company	LSI,11
Stosick, Jim, Stanford University	LSI,11
Strackbein, Ray, Chalfont Communications	11
Strange, Fred, Lawrence Livermore Labs	8,11
Strauss, Dick, DEC-Maynard	
Strecker, William D., DEC-Tewksbury	VAX
Strelko, Ronald, Milprint	11
Strezleck, S., Standard Brands, Inc.	
Strickland, James, Caterpilar Tractor Co.	11
Stroebe, D.G., Chevron Research Company	LSI,11
Stroh, William, Oceanic Enterprises, Inc.	CTS,11
Strutt, Chris, DEC-Reading	VAX,11
Stuart, Antoinette, USDA Forest Service	VAX,LSI
Stupak, Matt, CMI Corporation	11
Sturtevant, Jim, DEC-San Francisco	LSI,11
Stylos, Paul, DEC-Marlboro	
Sukiennik, Anthony, DEC-Merrimack	
Sullivan, Stephen J., New Mexico,Univ.of	11
Sullivan, Tim, International Harvester	LSI,11
Sully, Chris, Transcomm Data System	
Sumner, Thos, California, University of	LSI,11
Surma, Gary A., V.H. Monette & Company	11
Surovov, Peter A., General Electric Co.	11
Suski, Greg, Lawrence Livermore Labs	VAX,LSI
Sventek, Joe, Lawrence Berkeley Labs	VAX,11
Sventek, Virginia, Lawrence Berkeley Labs	VAX,11
Swanborg, Rick, Helix Technology Corp.	VAX,11
Swenson, Steven, Farinon Electric	11
Swierczewski, Joseph, Garden Way Mfg. Co., Inc.	
Swihart, Stanley J., PDS Associates	LSI,11
Swindell, C. Eric, Mason Clinic	
Sykes, David, CTEC, Inc.	11
Sykes, Patricia, Tailored Software	LSI,11
Sykes, Robert, Tailored Software	LSI,11
Sykora, Ronald G., EG&G Inc.	11
Szeto, Simon, DEC-Merrimack	11
Szurek, John L., US Army	LSI,11
Tabata, Les, Lawrence Berkeley Labs	VAX,11
Tabor, William, Florida Computer, Inc.	11
Tallerico, Dennis, Diamond Shamrock Corp.	11,15
Tamer, Katherine, Technology Incorporated	8,11
Tani, Kenneth, Tani Company	8,11
Tani, Jon R., Electronics Research Lab.	LSI, 8
Tanner, Bruce, Cerritos College	
Tannery, Verna, Dow Chemical USA	11
Tardif, Gill, DEC-Merrimack	LSI,11
Taylor, Ed, Finar Systems Ltd.	11
Taylor, James L., Brigham Young University	LSI,11
Taylor, Jonathan, DEC	VAX,LSI
Taylor, Kenneth, Bingham Mechanical	11
Taylor, Michael, San Francisco Fire Dept.	11
Taylor, Pamela M., Iowa State University	VAX,11
Teague, Charles T., VA Medical Center	8
Tellez, Anne, Cooperative Services, Inc.	11
Tenison, Ronald, Gatlin Gabel School	LSI,11
Tenney, Sam, Alpha Computing Inc.	LSI,11
Terrell, Mark, Informatics - PMI	
Terry, George, Tennessee Gas Pipeline Co.	11
Tetelman, Bruce, Columbia University	11
Thagard, Gregory, Lunday-Thagard Oil Co.	VAX,11
Thatcher, Raymond V., Union Carbide Corp.	
Thiel, David, GENRAD	8,11
Thissell, George, DEC-Marlboro	11
Thomas, Darrell G., Lovelace Biom. & Env. Res.	LSI,11
Thomas, Richard, Lawrence Berkeley Lab.	LSI,11
Thompson, Greg, DEC-Moffett Field	LSI,11
Thompson, John, Intermetrics, Inc.	VAX,11
Thompson, Patricia, NASA-Johnson Space Ctr.	LSI,11
Threatt, Douglas, USAFSAM/BRP	8,11
Throneburg, Monte, Grain Systems, Inc.	11
Tieman, Leonard J., Bell Helicopter Textron	11
Tikson, Michael, Battelle - Columbus	
Timm, Philip E., Bell Telephone Labs	VAX,11
Tippie, J.W., Argonne National Lab.	LSI,11
Tirmari, George H., Navy Astronautics Group	8,11
Tnnkel, Mitchell, DEC-Merrimack	
Todd, William, DEC-Merrimack	VAX,11
Toms, Shackelfor, Technical Advisors Inc.	LSI,11
Toscano, John, Cabot Corp.	11
Toth, Anne, DEC-Maynard	8
Townsend, Timothy, Stanford University	11
Tracy, Thomas E., Environmental Protection	11
Trauger, Gary, Battelle-Northwest	
Travis, Dale, Argonne Nat'l Labs	VAX,11
Tritch, Michael, Alley Crafts Company	11
Trujillo, Stephen C., Arizona, Univ. of	8
Tubbs, John, V.A. Hospital	8,11
Tucciarone, Alex, New Britain Housing	11
Tucker, Paul, Battelle-Northwest	VAX,11
Tumblin, Henry, Jackson Laboratory	8,11
Tunison, Michael S., Perfection Spring & Stamp	
Turcich, George W., Interlake, Inc.	11
Turner, David B., Data Processing Design, Inc.	11
Turner, John, Waikato, University of	11
Turner, Martha W., American Col.of Radiology	11
Turner, Ron, American Sign & Indicator	8,11
Tuso, Rosanne, Mints/NAMSB	11
Tyler, George, Lawrence Livermore Lab.	LSI,11
Tymoczko, Kathy L., SPSS, Inc.	VAX,11
Tysdal, Orville, Jones & Jones	11
Uhrich, Mark L., DEC-Maynard	8,11
Ulloa, Guillermo, Crocker Nuclear Lab.	LSI,11
Uptmor, Terry B., Pacific Gas & Elec.Co.	
Uter, Thomas G., University of California	8,11
Uzi, Gelleg, Grandt Israel	11
Valcarcel, Reynaldo, Gillette	11
Valdez, Gerardo F., Grupo Cydsa	11
Valentine, M.A., Pacific Telephone & Tele.	11
Van Camp, Warren, Informatics, Inc.	LSI,11
Van Doren, E.D., Kaman Sciences Corp.	LSI,11
Van Lehn, Allan, Lawrence Livermore Lab.	8,11
Van Sweringen, R.A., Exxon Research & Eng Co.	11
Van Volkenburg, Donald, DEC-Tewksbury	11
Vanarsdall, Paul J., Lawrence Livermore Labs	LSI,11

NAME	COMPUTER
Vanderpool, Tom, 3M Company	LSI,11
Vanevery, James J., Boeing Computer Services	VAX,11
Vanhorn, Earl C., DEC-Maynard	VAX,11
Vann, Dave, Oregon Minicomputer	LSI,11
Vardas, Leo S., Lawrence Berkeley Labs	LSI,11
Vaughn, Donald D., Tri-State Generation	VAX,11
Vavroch, Duane, Gazette Company	CTS,11
Venning,E. Peter, CBL Canterbury Limited	11
Verroest, Philippe, Shell Francaise	11
Viana, Thomas A., Naval Underwater Systems	VAX,11
Victor, Graeme A., Stanford University	11
Viehmann, Norman, Viehmann Corporation	11
Viggiano, Frank, Spiridells & Associates	VAX,11
Vilandre, John E., Minnesota, University of	11
Viscarola, Peter, DEC-Bedford	
Vivian, Stan, DEC-Winnipec, Manitoba	8,11
Voet, Janet, Telesensory Systems, Inc.	CTS,11
Vossler, Roger, TRW DSSG	VAX,11
Vuoso, Jerome, Brooklyn Friends School	8
Wagner, Kermit, GSA/FPA/MCL/ISD	
Walker, Justin C., Nat.Bureau of Standards	LSI,11
Walker, Larry, Lawrence University	11
Waller, George W., Arete Associates	VAX,11
Walraven, Robert, California, Univ. of	11
Walty, Kevin, Booz, Allen & Hamilton	11
Walus, Bryan, McDonnell Douglas	VAX
Ward, Charles, Argonne National Lab.	VAX,11
Ward, Sandra, Argonne National Lab.	11
Warner, Kurt E., Commonwealth Clinical Sys.	11
Wasley, David L., California, Univ. of	VAX
Watson, Bill, Nevada, University of	
Watson, Charles, Battelle-Northwest	
Watson, Larry, Grand Canyon College	11
Watts, Jeffrey C., California, Univ. of	8
Watts, Michael, Custom Computer Services	VAX,11
Watts, P.M., W.R. Grace & Company	11
Weatherhead, Alan, San Francisco State Univ.	11
Webber, Theodore, DEC-Merrimack	
Webster, Ronald, Toledo, University of	
Wecker, Stuart, DEC-Maynard	
Wedig, Gary, Merrell Research Center	VAX,11
Weg, Aaron, ADL Data Systems, Inc.	8,11
Wehry, Arthur H., Actron-McDonnell Douglas	VAX,11
Weihs, Mark, Toledo Edison Company	LSI
Weiser, Neil, Continental Group	11
Weiske, William, DEC-Nashua	11
Weitzman, Mertin, Microsound Company	LSI
Welch, Daniel F., Flight Systems Inc.	11
Welch, Ron, Cities of Pleasant Hill	
Welch, William D., System Development Corp.	VAX
Wells, Mary H., EG&G Inc.	11
Wempe, James, Dalgety, Inc.	11
Wentz, Carroll, York Graphic Services	LSI,11
Werner, Nancy, Bell Laboratories	VAX,11
Werner, R.E., Lawrence Livermore Labs	LSI
Westhaver, Al, Schnitzer Steel	
Weston, Daniel J., Roth Corporation	11
Weston, Mike, Los Altos High School	VAX,11
Weyand, Thomas, Technical Advisors, Inc.	
Wheaton, Kenneth L., Interactive Graphic Sys.	LSI,11
Wheeler, Dana, Naval Rework Facility	11
Whicker, Douglas J., Barron & Associates	8
White, L.C., Tri-Valley Growers	11
White, Rod V., MacDonald Services	11
Whitehurst, Jeanne K., Boeing Comm.Air. Co.	VAX,11
Whitfill, Jim, Los Alamos Scientific Lab.	LSI,11
Whitney, Rusty, Oregon Minicomputer	VAX,11
Whyde, Raymond, Battelle Columbus Labs	VAX
Wick, Darrell, Camusun College	
Wiehe, John, Deluxe Equipment Co.	11
Wiener, Sandra, Stanford University	LSI
Wierzba, Steven, Kaiser Permanente Medical	11
Wiggins, Harvey, Texas, Univ.of @ Dallas	11
Wikkerink, Robert, Lawrence Livermore Lab.	LSI
Wild, Donald, Aetna Insurance Co.	11
Wilder, A.L., Kastle Systems	LSI,11
Wiley, Ken, Lawrence Berkeley Labs	VAX,11
Wilfert, Thomas, Motorola	11
Willer, Richard, Illinois Bell Telephone	8,11
Williams, David, Prototype Develop Assoc.	VAX
Williams, Donald, Jet Propulsion Lab	LSI
Williams, Garry, Miliken & Company	11
Williams, R.J., Pertec Computer Corp.	LSI,11
Williamson, Marc, Georgia-Pacific Corp.	8,11
Williksen, Wayne, Ford Aerospace & Comm Corp.	VAX
Willis, James, DEC-Maynard	8
Wilner, Vicki, Massachusetts Inst.of Tech.	VAX
Wilson, Alan, Varian Graphics	VAX,11
Wilson, Frank, Hughes Aircraft	8
Wilson, Richard, St. Joseph's Hospital	LSI
Windsor, Brian, DEC-Calgary	
Winstrom, Lee, DEC-Bellevue	VAX,11
Winters, James, Volunteer State	11
Wirtz, Paul, Cytrol, Inc.	11
Witcher, Mark F., Diamond Shamrock	LSI,11
Witek, Richard, DEC-Merrimack	11
Wittenberg, Mark, Informatics/Programming	LSI,11
Wogsberg, Eric, Computer Technology	8,11
Wolchak, John, Saskatcheisan, Univ.of	11
Wolf, Richard, Pittsburgh, Univ. of	VAX
Wolsky, Gilbert, DEC-Merrimack	
Womack, Michael M., The Poise Company	11
Wong, Edmound, California, Univ. of	CTS
Wong, Jennie, Naval Air Rework Facility	11
Wong, John, Tektronix	11
Wong, Sam, National Semiconductor	8,11
Wong, Sheldon, Lawrence Berkeley Labs	VAX,11
Wong, Will, Crocker National Bank	11
Wood, John, Lawrence Berkeley Labs	LSI,11
Wood, Peter, Lawrence Berkeley Labs	LSI,11
Wood, Randall, MS Band of Choctaw Indian	11
Wood, Robert, Bell Laboratories	111
Woods, Wesley, The Boeing Computer	
Woodworth, Mike, Measurex Corporation	VAX
Wool, Thomas, E.I. Dupont De Nemours	LSI,11
Wooldridge, Kent, California State College	11
Woolford, Donald C., Cancer & Leukemia Group	11
Worstell, Glen, Parsec Systems	LSI,11
Wright, Bruce, Duke University	LSI,11
Wright, Edward, DEC-Maynard	
Wright, Fred, DEC-Oakland	VAX,11
Wright, John, Florida Computer	VAX,11
Wright, Larry, American Sign & Indicator	LSI,11
Wright, Robert, California State College	LSI,11
Wu, Shirley, Manufacturers Hanover	VAX,11
Wyman, Elizabeth, DEC-Waltham	VAX,11
Wyncott, George, Purdue Univ. Comp. Center	VAX,11
Wyrick, T.B., Texas Gas Transmission	VAX
Yang, Teresa, DEC-Merrimack	VAX,11
Yen, Albert, Lawrence Berkeley Labs	VAX
Yeraska, Robert, DEC-Tewksbury	VAX,11
Young, Bill, California, Univ. of	VAX,11
Young, Bob, Helix Technology Corp.	11
Young, George, Georgia-Pacific Corp.	11
Young, Jim, E & J Gallo	LSI,11
Young, John M., Crown-Zellerbach	11
Young, Steven, Missouri Pacific Railroad	11
Yu-Kuang, A., Borden Chemical	11
Yves, Riquet,Systems Informatiques	11

NAME	COMPUTER
Zaborowsky, Benjamin, Eli Lilly and Company	LSI,11
Zalkind, Herb, DEC-El Segundo	VAX,11
Zane, Ronald, California, Univ. of	8,11
Zappala, Chuck, Teltone Corp.	11
Zeise, Fred, Data Systems Design, Inc.	8,11
Zeisler, James, Enterprise Companies	
Zeitlin, Jim, Cetus Corporation	LSI,11
Zepeda, Raul, Comision Fed de Electric	VAX,11
Zerr, Alan, Altel Data/A.G.T.	11
Zima, Paul J., Evans & Sutherland Compu	VAX,11
Zimmer, William, DEC-Maynard	
Zingheim, Joseph, Stanford Linear	LSI,11
Zirkle, L.D., Sandia Corporation	11
Zlaket, Jerome, Rockwell International	VAX,11
Zoller, John, Xerox Corporation	LSI,11
Zongker, Layle, Los Alamos Scientific Lab.	
Zornes, Aaron, Cincom Systems	VAX,11
Swonitzer, Rodney E., Monolithic Systems Corp.	8,11
Zywiol, Gary, GM Mfg. Development	LSI

Printed in U.S.A.