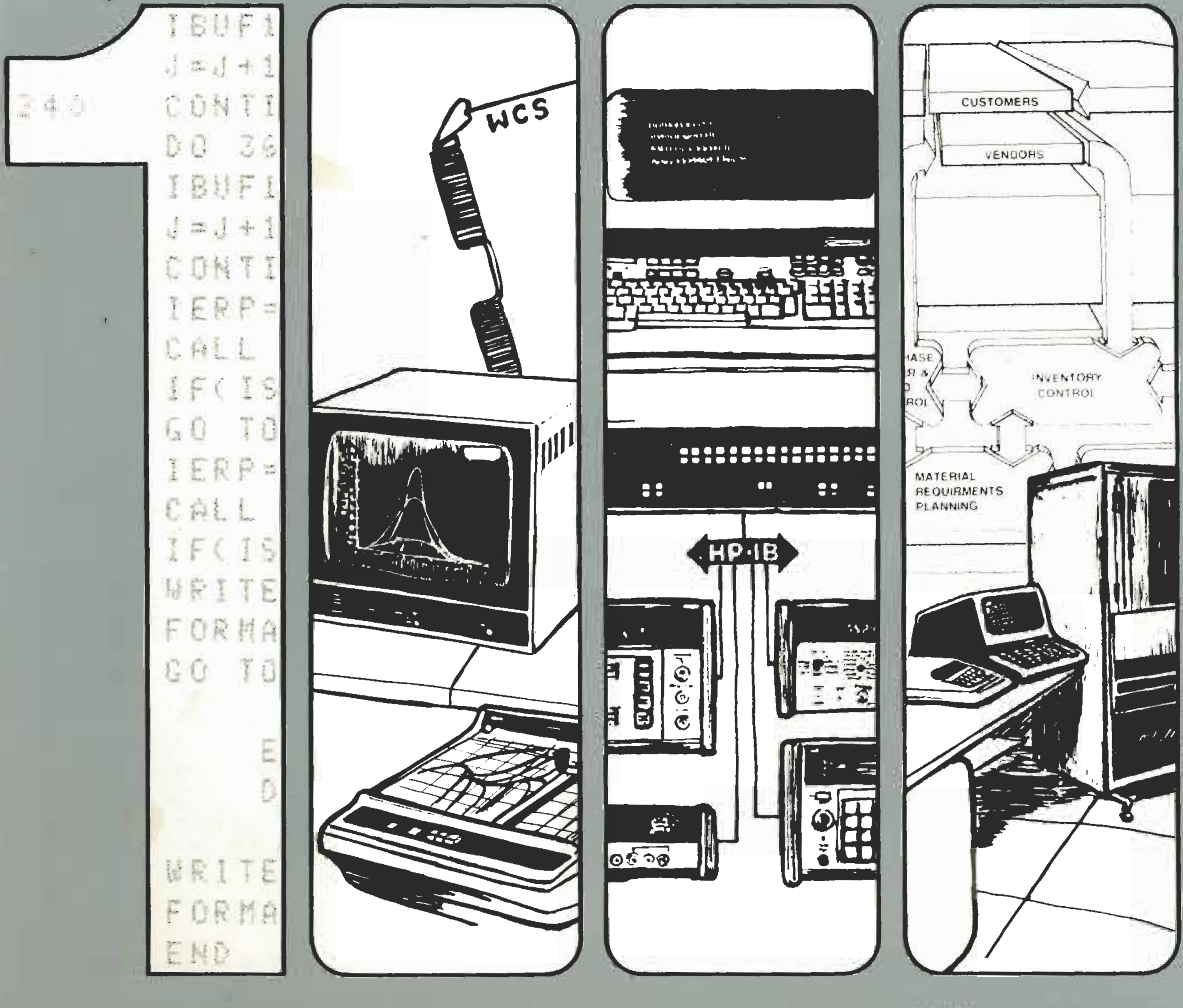
HEWLETT PACKARD

Hewlett-Packard Computer Systems COMMUNICATOR



1981

Volume V Issue 4

HEWLETT-PACKARD
COMPUTER SYSTEMS

Volume V
Issue 4

COMMUNICATOR/1000

Feature Articles

Departments

ABOUT THIS ISSUE

This issue of the Communicator/1000 includes one feature article contributed by an HP Customer, and one article reprinted from a previous Communicator. Our supply of feature articles is dwindling, so now is a great time to submit a feature article for publication consideration. Who knows you may be the next winner of an HP 32E calculator. See the section entitled "Becoming a Published Author in the Communicator/1000", for the mechanics of submitting an article.

In the Operating Systems category we have an article by Matt Betts at Fischer Body in Warren, Michigan. Matt describes a scheme using SSGA to coordinate the passing of data between two asynchronous programs, and is entitled "Program to Program Data Passing Using FIFO Queues in SSGA".

The instrumentation category features a reprint from a previous Communicator/1000. The article appeared in Vol. III Issue 4 and was a calculator award winner. So if you missed "The Fundamentals of HP-IB Addressing" by Neil Kuhn, now is your chance to read it.

With the introduction of the new RTE-6 Operating System, the Communicator editors are looking forward to publishing many interesting articles. However, we need your support. Take the opportunity to get familiar with RTE-6 by writing a Feature Article for the Communicator/1000. We look forward to hearing from you.

Unfortunately we didn't have any field or factory competitors for the 32E calculator. However, in the customer category, the decision was diffuclt. The judges selected:

"Program to Program Data Passing Using FIFO Queues in SSGA"
By Matt Betts, Fisher Body Division, GM in Warren, Michigan

BECOME A PUBLISHED AUTHOR IN THE COMMUNICATOR/1000 . . .

The COMMUNICATOR is a technical publication designed for HP 1000 computer users. Through technical articles, the direct answering of customers' technical questions, cataloging of contributed user programs, and publication of new product announcements and product training schedules, the COMMUNICATOR strives to help each reader utililize their HP 1000's more effectively.

The Feature Articles are clearly the most important part of the COMMUNICATOR. Feature Articles are intended to promote a significant cross-fertilization of ideas, to provide in-depth technical descriptions of application programs that could be useful to a wide range of users, and to increase user understanding of the most sophisticated capabilities designed into HP software. You might think of the COMMUNICATOR as a publication which can extend your awareness of HP 1000's to include that of thousands of users worldwide as well as that of many HP engineers in Data Systems factories at Cupertino, California and Grenoble, France.

To accomplish these goals, editors of the COMMUNICATOR actively seek technical articles from HP 1000 customers, HP Systems Engineers in the Field, and Marketing and R&D Engineers in the factories. Technical articles from customers are most highly valued because it is customers who are closest to real-world applications.

WIN AN HP-32E CALCULATOR!

Authoring a published article provides a uniquely satisfying and visible feeling of accomplishment. To provide a more tangible benefit, however, HP gives away three free HP-32E hand-held calculators to Feature Article authors in each COMMUNICATOR/1000 issue! Authors are divided into three categories. A calculator is awarded to the author of the best Feature Article in each of the author categories. The three author categories are:

1. HP 1000 Customers;

2. HP field employees;

3. HP division employees.

Each author category is judged separately. A calculator prize will be awarded even if there is only one entry in an author category.

Feature Articles are judged on the following bases: (1) quality of technical content; (2) level of interest to a wide spectrum of COMMUNICATOR/1000 readers; (3) thoroughness with which subject is covered; and, (4) clarity of presentation.

What is a Feature Article? A Feature Article meets the following criteria:

1. Its topic is of general technical interest to COMMUNICATOR/1000 readers;

2. The topic falls into one of the following categories —

   OPERATING SYSTEMS
   DATA COMMUNICATIONS
   INSTRUMENTATION
   COMPUTATION
   OPERATIONS MANAGEMENT
   LANGUAGES

3. The article covers at least two pages of the COMMUNICATOR/1000, exclusive of listings and illustrations (i.e., at least 1650 words).

There is a little fine print with regard to eligibility for receiving a calculator; it follows. No individual author will be awarded more than one calculator in a calendar year. In the case of multiple authors, the calculator will be awarded to the first listed author of the winning article. An article which is part of a series will compete on its own merits with other articles in the issue. The total of all articles in the series will not compete against the total of all articles in another series.

All winners of calculators will be announced in the issue of the COMMUNICATOR/1000 in which their articles appear. Again, all Feature Articles are judged by an impartial panel of three DSD Technical Marketing Engineers.

A SPECIAL DEAL IN THE OEM CORNER

When an HP 1000 OEM writes a Feature Article that is not only technically detailed and insightful but also application-oriented as opposed to theoretical, then that OEM may ask that the article be included in THE OEM CORNER. A Feature Article included in THE OEM CORNER may contain up to 150 words of pure product description as well as a picture or illustration of the OEM'S product or its unique contribution. HP's objective is twofold: (1) to promote awareness of the capabilities HP 1000 OEMs' products among all HP 1000 users; and, (2) to publish an article of technical interest and depth.

IF YOU'RE PRESSED FOR TIME . . .

If you are short of time, but still have that urge to express yourself technically, don't forget the COMMUNICATOR/1000 BIT BUCKET. It's the perfect place for a short description of a routine you've written or an insight you've had.

THE MECHANICS OF SUBMITTING AN ARTICLE

If at all possible please submit an RTE File containing the text of your article recorded on a Minicartridge (preferrably) or on a paper tape along with the line printer or typed copy of your article. This will help all of us to be more efficient. The Minicartridge will be returned to you promptly. Please include your address and phone number along with your article.

All articles are subject to editorship and minor revisions. The author will be contacted if there is any question of changing the information content. Articles requiring a major revision will be returned to the author with an explanatory note and suggestions for change. We hope not to return any articles at all; if we do, we would like to work closely with the author to improve the article. HP does, however, reserve the right to reject articles that are not technical or that are not of general interest to COMMUNICATOR/1000 readers.

Please submit your COMMUNICATOR/1000 article to the following address:

Editor, COMMUNICATOR/1000
Data Systems Division
Hewlett-Packard Company
11000 Wolfe Road
Cupertino, California 95014
USA

The Editor looks forward to an exciting year of articles in the COMMUNICATOR/1000.

With best regards,

The Editor

LETTERS TO THE EDITOR

Dear Ms. Editor,

I was pleasantly surprised to see my letter in the Volume IV issue 6. The fact that you have replied has encouraged me to write in these requests. From my as-far-out-as-one-can-get end user's point of view the HP 1000 RTE stands up very well, but there are a few things I would like to see:

- A faster version of the Help program.
- Though the HP 1000 has a nice internal byte addressing system, FORTRAN doesn't know it. I would like to see some character string capability, I'm tired of the 2HXX stuff.
- Selling videotapes which could teach me about the HP 1000 system.

Regards,

C. S. Hoppman
South Pacific Commission
New Caledonia

Dear Mr. Hoppeman,

Usually the Communicator/1000 doesn't respond to enhancement requests, but the introduction of the new RTE-6 Operating System and the new FORTRAN 77 Compiler has made many an enhancement request reality. The first one you mention, a faster HELP program, has been included as part of the RTE-6 Operating System. Now the HELP program not only finds your error messages much faster, but there is also an on-line quick reference guide for terminal commands. Your second request for string capability, is available in the new FORTRAN 77 Compiler. No longer do you have to struggle with the 2HXX, and the FORTRAN 77 Compiler also has string manipulation functions like concatenation. Please note that program development in FORTRAN 77 is only available under the RTE-6 operating system. Your third request for videotape lectures, is only available for FORTRAN 4X. Please consult Volume V Issue 1 for information on how to order the Fortran videotape. Thanks for the interest in our products.

Sincerely

Ms. Editor

Dear Ms. Editor,

The article presented by Mr. Jeff Deakin (Volume V issue 2) is very useful for a system with limited disc space. However, a further gain in disc space organization can be obtained by grouping all the relevant files into a single job file and applying a batch mode processing technique.

This can be done by merging source files, procedure files, data files and documentation files into a job file named "&TEST" as follows:

```
:JO,&TEST
:RU,FTN4,5,6,%TEST
FTN4,L
     PROGRAM TEST
     .
     .
     .
     END
     END$
:OF,TEST
:RU,LOADR,,%TEST
:RU,10G
DATA .....
:PU,&TEST
:OF,10G
:EOJ
DOCUMENTATION
  THIS IS A TEST PROGRAM FOR ...
END OF DOCUMENTATION
```

To RUN this program, simply type

```
RU,JOB,&TEST   [EXECUTE]
```

As the relocatable files are purged at the end of Job, No disc space are required to store the relocatables which release more disc space for program storage. The cartridge will be very neat as all the procedure files, source code, documentation as well as data can be merged into a single file.

YOU NEED NOT LOOK AROUND FOR BITS AND PIECES OF A SET OF FILES ANY MORE !

The only disadvantage is that all commands and errors will be displayed at the system console.

Sincerely,

W. S. Wong
VARTA PE LTD.
Singapore

Dear Mr. Wong

Thank you for forwarding this information to the Communicator/1000, I am sure some of our readers will appreciate this information.

Sincerely,

Ms. Editor

Dear Ms. Editor,

Are you aware of any utilities which will read a 9895 8" floppy from the HP 85, on the HP 1000 with a 9895 drive? If so, please could you forward either the software listing, or information leading to where I could obtain this utility.

Sincerely,

Dereck Roach
Solar Energy Research Inst.
Western Australia

Dear Mr. Roach,

In order for you to transfer data files between an HP 85 to an HP 1000 via an 8" floppy from an HP 9895 disc drive, the files must be in LIF format. Currently there is a utility program available from th HP 85 User's Group which has the capability to transfer a file to the 9895 in LIF Type 1 file format. On the HP 1000 side, there is a utility program supported under the new RTE-6 Operating System which also supports the LIF Type 1 file format and is called '%LIF'. These two utilities will give you the capability to transfer files in the LIF Type 1 format. For more information on the new RTE-6 Operating System see the Bulletin section of this issue.

Sincerely,

Ms. Editor

FILE MANAGEMENT USING SYMBOLS AND RESERVED WORDS

Arthur P. Briscoe/Revlon Health Care Group, Scarsdale NY

The six character limitation on HP File Manager file names is not a serious limitation but it is sometimes inconvenient. It would be nice if at least ten characters were available which would allow for the creation of a file which had a more meaningful name. Combining symbols with text and using reserved words is one way I have found to expand the meaning of a file name as well as help control the files created by all members of the programming staff.

File naming conventions may be a low priority subject in view of all the other problems associated with a project; however, when a file is desperately needed (usually at 10:00 p.m. after the programmers left and just before a next day deadline) the name suddenly becomes the most important topic of discussion.

The usual solution is to try to guess the file name and compare this name to a directory listing where there may be one to two hundred names. If this is not successful, the next step is to try to reach the file owner by phone. This may or may not be successful. If you try to reach an individual who just left the staff (possibly on bad terms) a phone call may bring other words none of which happens to include a file name.

A programming staff can generate numerous kinds of files that have a variety of uses. The file that is usually generated most and is probably considered to be of greatest importance is the program source file. There are however, numerous other files created which are equally important. Consider the following:

1. File Manager transfer files used to load programs, set system dates and times, initialize terminals, RP programs, etc.
2. Files for use as input to the Relocating Loader.
3. Data base schemas.
4. Report files for input to Query.
5. Data files for input the Image utilities such as files for DBBLD, DBULD and DBLOD.
6. A WELCOM file for initializing a system after bootup.
7. A HELLO file for use by the users copy of File Manager which the user logs on to in an RTE system with Session.
8. An initialization file for input to the DS utility DINIT.
9. An answer file for the Accounts Manager utility program ACCTS.
10. Binary library files.

This list could easily be continued since there are many different files that are created during the time that an HP system is utilized at an installation. Usually the number of files increases when a new product is added where files may be used to initialize and/or maintain the new product.

Although a six character file name can be arranged with more combinations and permutations than probably could be used, it also follows that the name should be meainingful. A file named "XEGJSV" does not offer the slighest suggestion as to its use to anyone but the owner of the file. Sometimes the owner of the file will not recognize the name. This can be easily demonstrated by passing around a large directory listing and asking the owners to identify their files. Many file names will be identified but many will not.

BIT BUCKET

In order to help determine the general function of a file, it is helpful to prefix the file name with a symbol, a combination of symbols, or use a reserved word. The central idea is to break down files into functional categories such as transfer files, loader files, source files, or schemas and to assign selected symbols to each functional category. When a file has a very specialized function, it should be named with an exclusive or reserved word. File classifications should be made as quickly as possible since the symbols or names can be revised at a later time if the origional classification did not turn out to be suitable.

HP suggests using a standard set of file conventions in the RTE-IVB Systems Managers Manual in the section called "File System Conventions." The chapter identifies several symbols which turn out to be a good start. After installation of our two HP 1000 systems with FORTRAN, BASIC, Graphics, DS, Image, RJE, a 7906 and a 7925 (120M byte disc) the importance of this chapter and good file organization became obvious.

The following table was derived by enhancing the list recommended by HP. New symbols were added and the idea of reserved words was also included. This list will be under constant revision as new symbols and words are added. For no particular reason I will list the reserved words first followed by the symbols.

NAME/SYMBOL PREFIX	**EXPLANATION**
LEGEND	This file contains the listing of all the symbols and reserved words as shown here. This is one of the few source files allowed on LU 2 or LU 3 so that it can be accessed by everyone whenever necessary.
INFO	A file containing text for use as the Session Sign-On file. This file is maintained by the System Librarian and contains any information which may be important to all users such as scheduled maintenance times, new subroutines or software, and general announcements.
WELCOM	The infamous WELCOM file used after bootup.
FTEMP	A "template" used as a documentation header for all production FORTRAN programs. No program will be cataloged without it.
&	The current version of a FORTRAN source program as known by all FORTRAN users.
<	The previous revision of a FORTRAN source program. (Since this version is usually not compiled the "&" is not used). Example: &RGEN = Version 1.5 <RGEN = Version 1.4
>	Second previous revision of a FORTRAN source program. Example: &RGEN = Version 1.5 <RGEN = Version 1.4 >RGEN = Version 1.3
&&	Future revision of a FORTRAN source program or subroutine. Example: &RGEN = Version 1.5 &&RGEN = Version 1.6
&	Test program for a subroutine or another program. The name is one less character than the name of the subroutine or program being tested. Example: &RGE = Test program for routine &RGEN.
%	The standard binary output file from the FORTRAN compiler.

*	Used to identify general File Manager transfer files. This type of file can be useful in performing various tasks such as purging files, running programs, etc.
	Example: *IMULT = Initialize Multipoint line
*/	File Manager transfer file used specifically to "RP" programs and/or segments.
*\	File Manager transfer file used specifically to "RP,," programs and/or segments.
*L	File Manager transfer file used specifically to load programs.
	Example: *LDBLD = Load program DBBLD
"	File Manager transfer file used specifically to list a program and associated subroutines. It usually contains a series of "LI" commands that lists a main program and its associated subroutines. The first "LI" command should list the transfer file to provide an index into the list- ing that will be produced.
]	Identifies all Session Monitor Hello files.
[	Files to be used with the Relocating Loader.
	Example: RU,LOADR,[RGEN = Load all &RGEN programs
'	The standard list file for output from the FORTRAN compiler.
=	A command file containing file names to be used by the MERGE utility.
	Example: RU,MERGE,=RGEN = Merge all &RGEN binaries.
$	A binary library or destination file created by the MERGE utility. (This is the new symbol used by HP for libraries.)
	Example: RU,MERGE,=RGEN,$RGEN = Merge all binaries associated with &RGEN.
?	Files for input to QUERY which produce reports as output.
	Example: RU,QUERY,?RPORT
@	Data base schema.
	Example: RU,DBDS,@CDMS1
↑	Data file containing formatted data suitable for input to the Image utility DBBLD.
	Example: RU,DBBLD,↑ERMSG
.	A file that can be purged. These files are the first to go when additional room is required on a disc LU.
!	Binary absolute object code.
)	Accounts Manager command file.
	Example: RU,ACCTS,)ACCT

Locating a file is easily performed using the "DL" command. If you are looking for a file that loads a program the sequence may be:

```
:DL,*L----::-11
CR=00011
 ILAB=CR11    NXTR= 00205 NXSEC=082 #SEC/TR=128 LAST TR=00288 #DR TR=02

NAME    TYPE    SIZE/LU     OPEN TO

*LF4X   00004 00003 BLKS      (Load FORTRAN 4X)
*LDBSP  00004 00002 BLKS      (Load DBSPA)
*LDBLD  00004 00007 BLKS      (Load DBBLD)
*LQURY  00003 00007 BLKS      (Load QUERY)
*LDBUL  00004 00002 BLKS      (Load DBULD)
*LDBLO  00004 00003 BLKS      (Load DBLOD)
*LBASC  00004 00003 BLKS      (Load BASIC)

:
```

With only four characters remaining to define a file after using "*L" it may be necessary to probe into the file using the Editor to further clarify the file contents, but usually four characters can be chosen which will give a good indication as to its use.

As can be seen, with little effort, an efficient method can be used to assist in locating files of interest with out imposing a great hardship on any staff member. Gradually implementing this type of idea will save time in the long run. It will also cause you to give more attention to other kinds of files, which if managed properly, can eliminate the time lost in running repetitive tasks. This method has proved to be helpful in the area of operations and configuration management.

SCHEDULING BASIC ON MULTIPLE TERMINALS

Olaf Meyer/HP Copenhagen

The following program was triggered by one of our customers, who wanted to run multiple copies of BASIC on an RTE-IVB system. The system had Session Monitor and Spooling, 64k words of memory, and one 7900 disc drive. He wanted to run BASIC from 12 terminals.

This customer obviously had problems both with lack of memory and of LG-tracks on the disc. If you are scheduling BASIC from 12 terminals, you will have 12 copies of the FMGR swapped out, occupying around 36 LG-tracks. Also you will have 12 ID-segments filled out for the FMGR copies. Since the terminal users are not using the FMGR copies, it would be nice to free the above resources.

So rather than scheduling BASIC directly, the terminal users schedule the following program, that will do the following:

1. pick up true lu number xx and form FMGXX and BASXX
2. blank the id-segment for basic copy 'BASXX'
3. do an 'OF,FMGXX,8'
4. schedule BASXX immediate, no wait

When the terminal user is through using BASIC ('bye'), he will not receive the usual ':' from the FMGXX copy, since this has been removed from the system. Thus, to do a normal log-off he must generate a breakmode prompt by hitting any key and then issue an 'RS' command to reschedule the FMGXX copy. Then he will be able to do a normal log-off.

```
FTN4,L
      PROGRAM BASIC(),OFF FMGXX, SCHED. BASXX, OM-810830-C
C**********************************************************************
C
C          THIS PROGRAM WILL 'OFF,8' FMGXX PRESENTLY RUNNING UNDER THE
C     SESSION AND SCHEDULE THE CORRESPONDING BASIC COPY BASXX, THUS
C     AVOIDING WASTE OF LG-TRACKS USED TO HOLD THE SWAPPED FMGR COPIES
C     UNDER NORMAL SCHEDULING OF BASIC.
C          IT IS ASSUMED THAT THE ID SEGMENT FOR BASIC IS FILLED OUT,
C     I.E. AN 'RP,BASIC' HAS BEEN PERFORMED BEFORE SCHEDULING THE
C     PROGRAM. THIS COULD BE ACCOMPLISHED IN THE 'WELCOM' FILL.
C          NOTE, THAT WHEN BASIC IS TERMINATED WITH THE 'BYE' COMMAND,
C     YOU MUST HIT A KEY TO GET A SYSTEM PROMPT AND ISSUE AN "RS"
C     COMMAND TO RESCHEDULE FMGXX AND THEN DO A NORMAL LOG-OFF.
C
C                        OLAF MEYER, HP A/S, DENMARK
C
C**********************************************************************
```

```
C
      INTEGER IBUF1(14),IBUF2(3),NAME(3),1,1A,INUM,ITEST,LUT,IERR
C
      DATA IBUF1/2HOF,2H,F,2HMG,2H  ,2H,8/,IBUF2/2HBA,2HS1,2HC /,
     +            NAME/2HBA,2HSI,2HC /
C
C---GET TRUE LU #
C
      ITEST=1
      CALL LUTRU(ITEST,LUT)
C
C---CONVERT LUT TO ASCII
C
      I=KCVT(LUT)
C
C---IF LUT IS LESS THAN 10, MAKE 1 = 0X (CHANGE SPACE TO 0)
C
      IF(LUT .LT. 10) I=I+10000B
C
C---PLACE I IN STRINGS IBUF1 & IBUF2 (DEC. STRING ARITHM.)
C
      CALL SMOVE(I,1,2,IBUF1,7)
      CALL SMOVE(I,1,2,IBUF2,4)
C
C---BLANK ID-SEGMENT OF 'BASXX'
C
      CALL IDRPD(IBUF2,IERR)
      IF(IERR .NE. 0) GO TO 9000
C
C---'OF,8' FMGXX
C
      INUM=10
      IA=MESSS(IBUF1,INUM)
C
C---RESTORE 'BASIC' TO 'BASXX'
C
      CALL IDDUP(NAME,IBUF2,IERR)
      IF(IERR .NE. 0- GO TO 9001
C
C---SCHEDULE BASXX IMM. NO WAIT
C
      CALL EXEC(10,IBUF2)
      GO TO 9999
C
9000  WRITE(1,10) IERR
10    FORMAT(/"BASIC, ERROR IN CALL IDRPD:", I5)
      GO TO 9999
C
9001  WRITE(1,20) IERR
20    FORMAT(/"BASIC, ERROR IN CAL IDDUP:",I5)
      GO TO 9999
9999  END
      END$
```

ONE MORE TIME

Dave Markwald/HP Bellevue

INTRODUCTION

When booting up an RTE operating system there is the need to set the system time. Frequently, this task is ignored or forgotten, although RTE sends a 'SET TIME' message to the system console. The problems from having an incorrect system time range from receiving editor, compiler or loader listings which are incorrectly dated, to obtaining measurment data from automatic tests, data acqusisiton, or process control operations that do not correspond to real world time, thereby invalidating costly or unrepeatable events. Another problem is encountered when a remote DS/1000 node is booted and there is not an operator or system console available to interactively set the system time.

Currently, there are several time setting programs in existence. For instance the RTE-IVB System Manager's Manual (92068-90006) contains the *STIME example while the DS/1000-IV Network Manager's Manual Vol. I (91750-90010) has a remote bootup example. Unfortunately, these examples are narrow in their application. The RTE initialization program, RTEIN, was developed to be a broadly based routine capable of executing on all current RTE operating systems (II/III/IVA/IVB/6/L/XL). RTEIN is able to set the system time either through interactive exchanges with an operator or by accessing a source node over a DS/1000 network. Additionally, RTEIN has the capability to set FMGR global 1P thus contributing a powerful flexiblity to the boot process.

HOW DOES IT WORK?

Interactive RTEIN execution begins by telling the operator to set the system time, then prompts for the MONTH/DAY/YEAR. The operator supplied date is parsed into integer values including corrections for leap year and then checked for legality. Incidentaly, the YEAR value is entered as the last two digits of the year and RTEIN will generate the correct four digit year from 1974 to 2075. Illegal dates cause an error message and the operator is reprompted for a legal date. Successful entry of a legal date allows RTEIN to prompt for the time in HOURS:MINS:SECS (24-hour clock). Illegal times cause a corresponding error message to be displayed and the operator is reprompted for a legal time. If no date or time is entered, RTEIN will skip setting the system time.

For DS/1000 remote operations RTEIN makes a DEXEC call to obtain the time at the source node specified in the run string. An unsuccessful DEXEC call will revert RTEIN to the interactive mode of operation.

Once a useable date and time are available the operating system type is determined through a call to the FORTRAN callable routine OPSYS, which interfaces to the MATH/FORMATTER Library assembly subroutine .OPSY. Based on the system type the date and time values are constructed into time messages compatible with the executing operating system. For RTE-II/III/IV/6 operating systems the time is set by calling the system subroutine MESSS. For RTE-L/XL operating systems the subroutine STIML is used to set the system time. STIML is necessary because the RTE-L/XL MESSS subroutine is capable of processing only the BASE SET operator commands, however the 'TM' command is a member of the COMND SET (92070-1X076) operator commands and is executed by the system subroutine TM.. (92070-1X105). The technique used by STIML is to emulate the COMND module 'TM' process by developing the required parameters and calling the TM.. subroutine directly.

Once the system time is set, RTEIN will echo it and prompt for operator approval. A negative operator response results in repeating the time setting process.

Upon successfully setting the system time the operator is prompted for the need to set FMGR global 1P. However, if RTEIN is in its remote mode of operation, FMGR Global 1P is set according to the value of the third run string parameter. Next, the DS/1000 program UPLIN is scheduled to run immediately to place it into the time list corresponding to the new system time (this must be done by the WELCOM file for RTE-L/XL systems).

RTEIN then passes the chosen Global 1P value to FMGR and terminates.

INSTALLING IT ON YOUR SYSTEM

RTEIN was written to be compatible with both FTN4 and FTN4X. Compiler option 'D' will cause RTEIN to be compiled for use with DS/1000; alternately, it can be compiled to avoid using DS/1000. Most CRT messages use inverse video and have been tested with both 264X and 262X terminals. When loading be sure to specify access to the SSGA area if you are using the DS/1000 version. Note that non-L/XL systems do not need the STIML subroutine and can ignore the undefined external. Some subroutines from the Decimal String Arithmetic Routines Library(24306-60001) are used by RTEIN. Typically, RTEIN is loaded temporarily and then saved as a type 6 file.

IN CONCLUSION

I like to thank Dick Deonigi/HP Bellevue SEO for the initial idea and feature suggestions. Also Kent Ferson/DSD Technical Marketing for the method of setting the RTE-L/XL time. Happy time setting!

```
FTN4,L,D
D     PROGRAM RTEIN(19,89),RTE INITIALIZATION (W/DS) <811102.1513>
C
C     PROGRAM RTEIN(3,89),RTE INITIALIZATION (NO DS) <811102.1513>
C
C     D. MARKWALD  HP/NSR-BELLEVUE SEO
C
C     *****************************************************
C     *                                                   *
C     * EDIT & COMPILE RTEIN ACCORDING TO YOUR NEED FOR DS  *
C     * THE D COMPILER OPTION WILL INCLUDE THE DS CAPABILITY *
C     * RTEIN CAN BE COMPILED EITHER WITH FTN4 OR FTN4X     *
C     *                                                   *
C     *****************************************************
C
C     RTEIN IS USED TO SET SYSTEM TIME & FMGR GLOBAL 1P.
C
C     RTEIN SHOULD BE EXECUTED FROM THE WELCOM FILE AS FOLLOWS:
C
C     :SV,4,,IH *** SAMPLE WELCOM FILE
C     :OF,UPLIN,FL  <--------------------+
C     :RU,RTEIN [,LUOP [,NODE [,OPTN]]]  I
C     :IF,1P,NE,1,3                      I
C     :DP,PACKING LU 2 & LU 3            +-- Required for RTE L/XL and
C     :PK,-2                             I   DS/1000 operations to remove
C     :PK,-3                             I   UPLIN from the time list,
C     :ON,UPLIN  <-----------------------+   set the system time, and
C     :SV,0,,IH                              reschedule UPLIN.
C     :EX
C
C     WHERE: LUOP - LOGICAL UNIT NUMBER OF SCHEDULING TERMINAL.
C            NODE - NODE NUMBER OF A DS/1000-IV NODE TO USE
C                   AS THE TIME SOURCE.
C            OPTN - YES = SET FMGR GLOBAL 1P TO 1.
```

```
C
C      TM FORMATS:
C
C      II/III/IV/6 SYNTAX - TM,YEAR,DAY,HOUR,MIN,SEC
C                              YEAR - FOUR DIGITS
C                              DAY  - THREE DIGIT "JULIAN" DATE
C
C      L/XL SYNTAX       - TM,HOUR,MIN,SEC,MONTH,DAY,YEAR
C                              MONTH - 1 TO 12
C                              DAY   - 1 TO 31
C                              YEAR  - 1976 TO 2144
C                           NOTE:  L/XL WILL NOT SET THE TIME IF
C                           THERE ARE ANY PROGRAMS IN THE TIME LIST.
C
C      LOADING INFORMATION:
C
C      IVB/6 SYSTEMS -
C
C      OP,SS (for DS/1000 access)
C      REL,%RTEIN
C      SEA,%OPSYS
C      SEA,%DECAR
C      FO
C      END
C
C      L/XL SYSTEMS -
C
C      ECHO
C      SNAP,SNAP
C      CPAGE
C      SCOM (or LCOM for RTE-XL and DS/1000 access)
C      OUTPUT,RTEIN
C      REL,%RTEIN
C      SEA,%OPSYS
C      SEA,%STIML
C      SEA,$CMDLB
C      SEA,%DECAR
C      END
C
C
       INTEGER YEAR,JDAY,MNTH,IDAY,HOUR,MINS,SECS,MNDA(12),
     + TIMM(15),ONUPLN(6),FTIM(15),LEAP,STMT,NODE,OPTN,SYSTEM,
     + PRSBF(33),SLASH,COLON,LBCMA,CMASP,IA,IB,ICHR,IERR,I
       LOGICAL FOUND,RMOT
C
       DATA MNDA / 0, 31, 59 ,90, 120, 151, 181, 212,
     +           243, 273, 304, 334 /
       DATA TIMM/2HTM,2H, ,13*2H  /
D      DATA ONUPLN/2HON, 2H,U, 2HPL, 2HIN, 2H,N, 2HOW/
       DATA SLASH/027400B/
       DATA COLON/035000B/
       DATA LBCMA/000054B/
       DATA CMASP/026040B/
C
C      GET SCHEDULING PARAMETERS
```

```
C
       CALL RMPAR(FTIM)
       LUOP=FTIM(1)
       NODE=FTIM(2)
       OPTN=FTIM(3)
D      RMOT=.FALSE.
       IF(LUOP.LT.1.OR.LUOP.GT.255) LUOP=1
D      IF(NODE) 50,80,60
D   50 NODE=-1
D   60 RMOT=.TRUE.
C
C      WHICH TIME SOURCE?
C
    80 CONTINUE
D      IF(RMOT) GO TO 190
C
C      TELL USER TO SET TIME
C ␍
␊  100 CALL REIO(2,LUOP,22H␛H␛J␛&dC SET TIME ␛&d@,-22) ␍
␊C␛Z
   102 DO 105 I=1,15
   105 FTIM(I)=2H
C
C      GET DATE FROM USER
C ␍
␊       CALL REIO(2,LUOP, ␍
␊      +62H␛&dB by entering date as- month/day/year  (ie,12/25/80) ␛&d@ _, ␍
␊      +-62) ␍
␊C␛Z
       CALL REIO(1,LUOP+400B,FTIM,-8)
       CALL ABREG(IA,IB)
       IF(IB.LT.1) GO TO 500
       CALL SPUT(FTIM,IB+1,LBCMA)
       FOUND=.FALSE.
       ICHR=0
   108 IERR=0
       ICHR=ICHR+1
       IF(ICHR.GT.IB) 110,112
   110 IF(FOUND) 118,500
   112 IF(JSCOM(FTIM,ICHR,ICHR,SLASH,1,IERR)) 108,114,108
   114 CALL SPUT(FTIM,ICHR,LBCMA)
       FOUND=.TRUE.
       IF(ICHR.LE.IB) GO TO 108
   118 CALL PARSE(FTIM,IB,PRSBF)
       MNTH=PRSBF(2)
       IDAY=PRSBF(6)
       YEAR=PRSBF(10)
C
C      LEGAL VALUES?
C
       ASSIGN 125 TO STMT
       IF(MNTH.LT.1) ASSIGN 120 TO STMT
       IF(MNTH.GT.12) ASSIGN 120 TO STMT
       IF(IDAY.LT.1.OR.IDAY.GT.31) ASSIGN 120 TO STMT
       IF(YEAR.LT.0.OR.YEAR.GT.99) ASSIGN 120 TO STMT
       GO TO STMT
   120 CALL REIO(2,LUOP,15HILLEGAL DATE!,-15)
       GO TO 102
```

```
C
  125 IF(YEAR.GT.74) YEAR=YEAR+1900
      IF(YEAR.LT.75) YEAR=YEAR+2000
C
C     TAKE CARE OF LEAP YEAR
      JDAY=MNDA(MNTH)+IDAY
      LEAP=YEAR-4*(YEAR/4)
      IF (LEAP.EQ.0.AND.MNTH.EQ.2.AND.IDAY.GT.29) GO TO 120
      IF (LEAP.EQ.0.AND.MNTH.GT.2) JDAY=JDAY+1
C
C     GET TIME FROM USER
C ␍
␊  130 CALL REIO(2,LUOP, ␍
␊     +62H␛&dB by entering time as- hours:mins:secs (24-hr clock) ␛&d@ _, ␍
␊     +-62) ␍
␊      DO 140 I=1,15 ␍
␊  140 FTIM(I)=2H ␍
␊C␛Z
      CALL REIO(1,LUOP+400B,FTIM,-8)
      CALL ABREG(IA,IB)
      IF(IB.LT.1) GO TO 500
      CALL SPUT(FTIM,IB+1,LBCMA)
      FOUND=.FALSE.
      ICHR=0
  142 IERR=0
      ICHR=ICHR+1
      IF(ICHR.GT.IB) 144,146
  144 IF(FOUND) 150,500
  146 IF(JSCOM(FTIM,ICHR,ICHR,COLON,1,IERR)) 142,148,142
  148 CALL SPUT(FTIM,ICHR,LBCMA)
      FOUND=.TRUE.
      IF(ICHR.LE.IB) GO TO 142
  150 CALL PARSE(FTIM,IB,PRSBF)
      HOUR=PRSBF(2)
      MINS=PRSBF(6)
      SECS=PRSBF(10)
C
C     LEGAL VALUES?
C
      ASSIGN 200 TO STMT
      IF(HOUR.LT.0.OR.HOUR.GT.24) ASSIGN 170 TO STMT
      IF(MINS.LT.0.OR.MINS.GT.59) ASSIGN 170 TO STMT
      IF(SECS.LT.0.OR.SECS.GT.59) ASSIGN 170 TO STMT
      GO TO STMT
  170 CALL REIO(2,LUOP,15HILLEGAL TIME!,-15)
      GO TO 130
C
C     GET DATE AND TIME FROM DS/1000-IV NODE
C
D 190 CALL DEXEC(NODE,11+100000B,FTIM,YEAR)
D     GO TO 120
D 195 JDAY=FTIM(5)
D     HOUR=FTIM(4)
D     MINS=FTIM(3)
D     SECS=FTIM(2)
D     WRITE(LUOP,990) (FTIM(I), I=2,5),YEAR
D 990 FORMAT("DEXEC",4(X,I2),X,I4)
C
C     GET OPERATING SYSTEM TYPE
```

```
C
  200 CALL OPSYS(SYSTEM)
C
C     L OR XL?
      IF(SYSTEM.LT.-25) 210,250
C
C     X/XL TM MESSAGE
C      1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
C     TM, HR, MN, SC, MO, DY, YEAR
C
  210 TIMM(3)=KCVT(HOUR)
      TIMM(4)=CMASP
      TIMM(5)=KCVT(MINS)
      TIMM(6)=CMASP
      TIMM(7)=KCVT(SECS)
      TIMM(8)=CMASP
C
C     FIND MONTH & DAY OF MONTH
C     TAKE CARE OF LEAP YEAR
      LEAP=YEAR-4*(YEAR/4)
      DO 220 MNTH=12,1,-1
      IDAY=JDAY-MNDA(MNTH)
      IF (LEAP.EQ.0.AND.MNTH.GT.1) IDAY=IDAY-1
      IF(IDAY) 220,220,230
  220 CONTINUE
  230 TIMM(9)=KCVT(MNTH)
      TIMM(10)=CMASP
      TIMM(11)=KCVT(IDAY)
      CALL CNUMD(YEAR,TIMM(12))
      TIMM(12)=CMASP
      CALL PARSE(TIMM,28,PRSBF)
C     DECREMENT COUNTER BECAUSE TM.. DOESN'T NEED TM COMMAND
      PRSBF(33)=PRSBF(33)-1
      CALL STIML(PRSBF(33),PRSBF(5),IERR)
      IF(IERR.EQ.34) CALL REIO(2,LUOP,20HTIME-LIST NOT EMPTY!,-20)
      IF(IERR) 120,300,120
C
C     II/III/IV/6 TM MESSAGE
C      1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
C     TM,   YEAR,    DAY, HR, MN, SC
C
  250 CALL CNUMD(YEAR,TIMM(3))
      TIMM(6)=CMASP
      CALL CNUMD(JDAY,TIMM(7))
      TIMM(10)=CMASP
      TIMM(11)=KCVT(HOUR)
      TIMM(12)=CMASP
      TIMM(13)=KCVT(MINS)
      TIMM(14)=CMASP
      TIMM(15)=KCVT(SECS)
      I = MESSS(TIMM,30)
C
C     TELL USER THE CURRENT TIME
```

```
C
  300 CONTINUE
D     IF(RMOT) GO TO 510
      IF (OPTN(510,310,510))
  310 CALL FTIME(FTIM)

      CALL REIO(2,LUOP,
     +40H&dB System time and date is now: &d@ _,
     +-40)
      CALL REIO(2,LUOP,FTIM,15)
C
      CALL REIO(2,LUOP,0,0)
      CALL REIO(2,LUOP,28H&dB Correct? (YE,NO) &d@ _,-28)
CZ
      CALL REIO(1,LUOP+400B,I,-2)
      IF(I.EQ.2HYE.OR.I.EQ.2H  ) 500,100
C
  500 CALL REIO(2,LUOP,0,0)
      CALL REIO(2,LUOP,
     +36HDo FMGR optional commands? (YE,NO) _,-36)
C
C     GET FMGR GLOBAL 1P VALUE
C
      CALL REIO(1,LUOP+400B,OPTN,-2)
  510 TIMM(1)=0
      IF (OPTN.EQ.2HYE) TIMM(1)=1
C
C     CLEAR DISPLAY
C ␍
␊      CALL REIO(2,LUOP,4H␛H␛J,-4) ␍
␊C␛Z
C     RESET DS UPLIN EXECUTION TIME
C
D     I= MESSS(ONUPLN,13,LUOP)
C
C     RETURN FMGR GLOBAL 1P
C
      CALL PRTN(TIMM(1))
      END
      END$
```

```
ASMB,R,L,C
      NAM STIML,7 SET L/XL TIME  810624.0958
*
*     D. MARKWALD HP/NSR-BELLEVUE SEO
*
      EXT TM..,.ENTR,LOGLU
*
      ENT STIML,CAM.O
*
PCNT  BSS 1         PARAMETER COUNT
PBUF  BSS 1         START OF BUFFER
PERR  BSS 1         ERROR RETURN CODE
*
STIML NOP           RETRIEVE INPUT PARMS
      JSB .ENTR
      DEF PCNT
      LDA PCNT
      STA CNT       SAVE LOCALLY
      LDA PBUF
      STA BUF       SAVE LOCALLY
      LDA PERR
      STA ERR       SAVE LOCALLY
*
      JSB LOGLU     GET CONSOLE LU
      DEF *+2
      DEF CAM.O     DUMMY PARM
      STA CAM.O     SET LU
*
      JSB TM..      SET L/XL TIME!
      DEF EXIT
CNT   NOP
BUF   NOP
ERR   NOP
EXIT  JMP STIML,I   ALL DONE
CAM.O OCT 0         DEFAULT BIT BUCKET
      END


ASMB
      NAM OPSYS,7 GET .OPSY FOR FTN  810505.1742
*
*     D. MARKWALD  HP/NSR-BELLEVUE SEO
*
      EXT .ENTR,.OPSY
*
      ENT OPSYS
*
TYPE  NOP
OPSYS NOP           ENTRY POINT
      JSB .ENTR     GET RETURN
      DEF TYPE       PARAM ADDRESS
*
      JSB .OPSY     GET
      STA TYPE,I     SYSTEM TYPE
      JMP OPSYS,I   EXIT
      END
```

AN EDITOR FOR TYPE 1 FILES

Paul Henderson/Ft. Worth Texas

INTRODUCTION

Some RTE users need the capability to change an element within a type 1 file. The Hewlett-Packard interactive programs EDITR or EDIT/1000 supplied as a part of the Hewlett-Packard RTE system cannot be used on type 1 files. An interactive program named EDYT1 has been written to fulfill this need. A listing of this EDYT1 program is included with the article.

BACKGROUND

My application programs require a large amount of library and control reference information. This information is supplied to the application programs in two stages. The information is originally created by using the Hewlett-Packard program EDITR or EDIT/1000 and stored as type 4 files. One set of intermediate programs will read several of these type 4 files and combine them into a single type 1 file. Some of the other application programs will then read and use these type 1 files. This then permits faster operation of these application programs and also makes the task of programming for them much shorter and simpler in order to get all the needed information loaded from the disc.

Frequently it is necessary to alter a piece of information in one of the type 1 files on-line between runs of application programs. It is desirable to be able to do this without resorting to changing the type 4 file and then running the intermediate program to convert the type 4 files to a type 1 file. This is especially true when the change will be a temporary one for only a single pass of the application program.

The arrays of information contained in my type 1 files have been structured around the 128 word block size and record length used in transmitting and storing this type of information. This makes it very simple to keep up with the block number and the location in which a given piece of information is stored.

OPERATION

When the EDYT1 program is run it first requests the name of the file to be edited along with the format type for the data, the security code and cartridge reference number, and the block number. With this input the file is opened and the appropriate block is read from the file and the file is closed. A listing is then made on the terminal with the address for each word within the block along with the current contents of that word. The EDYT1 program then requests an element address within the block which is to be changed. The new value to be entered at that address within the block of data is then requested. Additional element addresses and values are then requested until an invalid element address (less than 1 or greater than 128 for integer types or 64 for real types) is entered which causes the updated block to be listed on the terminal and the file to be reopened and the updated block of data to be written back onto the disc and the file to be closed. Normally a zero is entered to terminate the edit session.

COMMENTS

Although written primarily for type 1 files, this program may be used with any type of disc file. It is not necessary to change the file in any way and the program is useful to examine the contents of any block of any disc file. One disadvantage is that the block number within the file must be known but with careful planning when dimensioning the arrays in the files to be stored this is not difficult.

The program is written in FORTRAN and should run with any Hewlett-Packard RTE operating system. The program could be modified to permit output to a printer or other medium. The program handles four types of data formats. They are Integer, Octal, ASCII, and Real. These may be specified by entering a single character in response to the program request. The default type is Integer which is specified by the return key or a space and return. Integer may also be specified by the character I. Type ASCII may be specified by an A or H. Real type may be specified by an E, F, or G. An Octal type is specified by an @, B, K, or O. Any other response will not be accepted by the program and the request will be repeated until a satisfactory type is entered. An example of the operation of the program is also shown.

CAUTION

The 128 word data block size should be retained in any modification to this program. This is due to the action of the disc driver in overwriting the remainder of a block when a partial block is written to the disc. For example, if a 129 word data block size were to be used the first word of the next block following the specified block would be read from the disc, but following the edit session when the 129 words of data were written back onto the disc every word of the block following the specified block would be changed. Thus any valid data that was on that portion of the disc would be destroyed.

A multiple of 128 words could be used safely but the entire data block would not fit conveniently on the display of the terminal.

```
FTN4,L
      PROGRAM EDYT1
C
C*********************************************************************
C*                                                                   *
C*    Program to edit type 1 files                                   *
C*            Written in September 1981 by Paul W. Henderson         *
C*            Post Office Box 5215, Fort Worth, Texas 761080215      *
C*            (817) 732-4811 Extension 2050                          *
C*                                                                   *
C*********************************************************************
C
C      Input data:
C
C      IFILE(3)      File name
C      ISC           File security code
C      ICR           File cartridge reference number
C      IFMAT         File format type(ASCII,Octal,Integer,Real)
C      IBLOK         Block number in the file
C      IELEM         Element number in the block
C
      DIMENSION IDCB(144),IPRAM(5),ICLK(15),IFILE(3),IDATA(128),
      1FDATA(64)
C
      EQUIVALENCE (IDATA(1),FDATA(1))
C
C The program requests the name of the file, the format type for the
C data, the security code, the cartridge reference number, and the
C block number within the file to be edited. The program then opens
C the file and reads the data from the block specified and closes
C the file. The data is then listed on the terminal in the format
C requested. An element number within the block is requested and
C then the new value to be put into that element is requested. The
C new value is entered into the data and another element and new
C value are requested. This is repeated until an invalid element
C number is entered (less than 1 or greater than 128 for integers
C or 64 for reals). An invalid element number causes the updated
C block of data to be listed on the terminal, the file to be opened,
C the updated block of data to be written back into the file, and the
C file to be closed. The program then terminates.
C
C The valid format specifications for the data types are:
C
C       Integer         ASCII          Octal          Real
C          I              A              @              E
C        Space            H              B              F
C                                        K              G
C                                        O
```

```
C
C**********************************************************************
C*                                                                   *
C*     Get the LU for the terminal                                   *
C*                                                                   *
C**********************************************************************
C
      CALL RMPAR(IPRAM)
      LUC  = IPRAM(1)
      IF(LUC .EQ. 0) LUC = 1
C
    5 FORMAT("Error ="I8)
C
C**********************************************************************
C*                                                                   *
C*     Parameter entry from the terminal keyboard                    *
C*                                                                   *
C**********************************************************************
C
   10 WRITE(LUC,20)
   20 FORMAT("File name ? _")
      READ(LUC,30)(IFILE(I),I=1,3)
   30 FORMAT(3A2)
      IFMAT = 1HI
   35 WRITE(LUC,40)
   40 FORMAT("Format type?_")
      READ(LUC,50)IFMAT
   50 FORMAT(1A1)
      IFMT = 1
      IF(IFMAT .EQ. 1HI .OR. IFMAT .EQ. 1H ) GO TO 70
      IFMT = 2
      IF(IFMAT .EQ. 1HA .OR. IFMAT .EQ. 1HH) GO TO 70
      IFMT = 3
      IF(IFMAT .EQ. 1H@ .OR. IFMAT .EQ. 1HB .OR.
     1   IFMAT .EQ. 1HK .OR. IFMAT .EQ. 1HO) GO TO 70
      IFMT = 4
      IF(IFMAT .EQ. 1HE .OR. IFMAT .EQ. 1HF .OR.
     1   IFMAT .EQ. 1HG) GO TO 70
      WRITE(LUC,60)
   60 FORMAT(" Format type error")
      GO TO 35
   70 WRITE(LUC,80)
   80 FORMAT("Security code, cartridge reference? _")
      READ(LUC,*)ISC,ICR
      WRITE(LUC,90)
   90 FORMAT("Block number? _")
      READ(LUC,*)IBLOK
```

```
C
C*********************************************************************
C*                                                                   *
C       Get the block of data from the file                          *
C*                                                                   *
C*********************************************************************
C
      CALL OPEN (IDCB,IERR,IFILE,2,ISC,ICR)
      IF(IERR .LT. 0) WRITE(LUC,5)IERR
      CALL READF(IDCB,IERR,IDATA,128,LEN,IBLOK)
      IF(IERR .LT. 0) WRITE(LUC,5)IERR
      CALL CLOSE(IDCB,IERR)
C
C
C*********************************************************************
C*                                                                   *
C*      Write a heading for the data on the terminal                 *
C*                                                                   *
C*                                                                   *
C*********************************************************************
C
      CALL FTIME(ICLK)
      WRITE(LUC,95)(IFILE(I),I=1,3),IBLOK,(ICLK(J),J=1,15)
   95 FORMAT(" File ",4A2," Block",I6," at ",15A2/," ")
C
C
C*********************************************************************
C*      Go to the proper format section to list and edit             *
C*********************************************************************
```

```
C
C*******************************************  Integer format
  100 WRITE(LUC,110)(I,IDATA(I),I=1,128)
  110 FORMAT(7(I4":"I6))
  120 WRITE(LUC,130)
  130 FORMAT("Element number ? _")
      READ(LUC,*)IELEM
      IF(IELEM .LT. 1 .OR. IELEM .GT. 128) GO TO 150
      WRITE(LUC,140)
  140 FORMAT("New Value ? _")
      READ(LUC,*)IDATA(IELEM)
      GO TO 120
  150 CALL FTIME(ICLK)
      WRITE(LUC,95)(IFILE(I),I=1,3),IBLOK,(ICLK(J),J=1,15)
      WRITE(LUC,110)(I,IDATA(I),I=1,128)
      GO TO 800
C*******************************************  ASCII format
  200 WRITE(LUC,210)(I,IDATA(I),I=1,128)
  210 FORMAT(10(I4,1X,1A2))
  220 WRITE(LUC,130)
      READ(LUC,*)IELEM
      IF(IELEM .LT. 1 .OR. IELEM .GT.  128) GO TO 250
      WRITE(LUC,140)
      READ(LUC,230)IDATA(IELEM)
  230 FORMAT(1A2)
      GO TO 220
  250 CALL FTIME(ICLK)
      WRITE(LUC,95)(IFILE(I),I=1,3),IBLOK,(ICLK(J),J=1,15)
      WRITE(LUC,210)(I,IDATA(I),I=1,128)
      GO TO 800
C*******************************************  Octal format
  300 WRITE(LUC,310)(I,IDATA(I),I=1,128)
  310 FORMAT(7(I4":"@6))
  320 WRITE(LUC,130)
      READ(LUC,*)IELEM
      IF(IELEM .LT. 1 .OR. IELEM .GT. 128) GO TO 350
      WRITE(LUC,140)
      READ(LUC,*)IDATA(IELEM)
      GO TO 320
  350 CALL FTIME(ICLK)
      WRITE(LUC,95)(IFILE(I),I=1,3),IBLOK,(ICLK(J),J=1,15)
      WRITE(LUC,310)(I,IDATA(I),I=1,128)
      GO TO 800
C*******************************************  Real number format
  400 WRITE(LUC,410)(I,FDATA(I),I=1,64)
  410 FORMAT(4(I4":"E14.6))
  420 WRITE(LUC,130)
      READ(LUC,*)IELEM
      IF(IELEM .LT. 1 .OR. IELEM .GT. 64) GO TO 450
      WRITE(LUC,140)
      READ(LUC,*)FDATA(IELEM)
      GO TO 420
  450 CALL FTIME(ICLK)
      WRITE(LUC,95)(IFILE(I),I=1,3),IBLOK,(ICLK(J),J=1,15)
      WRITE(LUC,410)(I,FDATA(I),I=1,64)
      GO TO 800
C*********************************************************************
C*                                                                   *
C*      Store the updated block back into the file                   *
C*                                                                   *
C*********************************************************************
```

```
C
  800 CALL OPEN (IDCB,IERR,IFILE,2,ISC,ICR)
      IF(IERR .LT. 0) WRITE(LUC,5)IERR
      CALL WRITF(IDCB,IERR,IDATA,128,IBLOK)
      IF(IERR .LT. 0) WRITE(LUC,5)IERR
      CALL CLOSE(IDCB,IERR)
C
C
      END

:RU,EDYT1
File name ?  FILE
Format type? I
Security code, cartridge reference? 0
Block number?  1
 File FILE   Block  12592 at :06 AM  SAT., 12  SEPT, 1981
  1:     1   2:     2   3:     3   4:     4   5:     5   6:     6   7:     7
  8:     8   9:     9  10:    10  11:    11  12:    12  13:    13  14:    14
 15:    15  16:     0  17:     0  18:     0  19:     0  20:     0  21:     0
 22:     0  23:     0  24:     0  25:     0  26:     0  27:     0  28:     0
 29:     0  30:     0  31:     0  32:     0  33:     0  34:     0  35:     0
 36:     0  37:     0  38:     0  39:     0  40:     0  41:     0  42:     0
 43:     0  44:     0  45:     0  46:     0  47:     0  48:     0  49:     0
 50:     0  51:     0  52:     0  53:     0  54:     0  55:     0  56:     0
 57:     0  58:     0  59:     0  60:     0  61:     0  62:     0  63:     0
 64:     0  65:     0  66:     0  67:     0  68:     0  69:     0  70:     0
 71:     0  72:     0  73:     0  74:     0  75:     0  76:     0  77:     0
 78:     0  79:     0  80:     0  81:     0  82:     0  83:     0  84:     0
 85:     0  86:     0  87:     0  88:     0  89:     0  90:     0  91:     0
 92:     0  93:     0  94:     0  95:     0  96:     0  97:     0  98:     0
 99:     0 100:     0 101:     0 102:     0 103:     0 104:     0 105:     0
106:     0 107:     0 108:     0 109:     0 110:     0 111:     0 112:     0
113:     0 114:     0 115:     0 116:     0 117:     0 118:     0 119:     0
120:     0 121:     0 122:     0 123:     0 124:     0 125:     0 126:     0
127:     0 128:     0
Element number ? 16
New Value ? 16
Element number ? 120
New Value ? 120
Element number ? 0
 File FILE   Block 12592 at :07 AM  SAT., 12  SEPT,  1981
  1:     1   2:     2   3:     3   4:     4   5:     5   6:     6   7:     7
  8:     8   9:     9  10:    10  11:    11  12:    12  13:    13  14:    14
 15:    15  16:    16  17:     0  18:     0  19:     0  20:     0  21:     0
 22:     0  23:     0  24:     0  25:     0  26:     0  27:     0  28:     0
 29:     0  30:     0  31:     0  32:     0  33:     0  34:     0  35:     0
 36:     0  37:     0  38:     0  39:     0  40.     0  41:     0  42:     0
 43:     0  44:     0  45:     0  46:     0  47:     0  48:     0  49:     0
 50:     0  51:     0  52:     0  53:     0  54:     0  55:     0  56:     0
 57:     0  58:     0  59:     0  60:     0  61:     0  62:     0  63:     0
 64:     0  65:     0  66:     0  67:     0  68:     0  69:     0  70:     0
 71:     0  72:     0  73:     0  74:     0  75:     0  76:     0  77:     0
 78:     0  79:     0  80:     0  81:     0  82:     0  83:     0  84:     0
 85:     0  86:     0  87:     0  88:     0  89:     0  90:     0  91:     0
 92:     0  93:     0  94:     0  95:     0  96:     0  97:     0  98:     0
 99:     0 100:     0 101:     0 102:     0 103:     0 104:     0 105:     0
106:     0 107:     0 108:     0 109:     0 110:     0 111:     0 112:     0
113:     0 114:     0 115:     0 116:     0 117:     0 118:     0 119:     0
120:   120 121:     0 122:     0 123:     0 124:     0 125:     0 126:     0
127:     0 128:     0
```

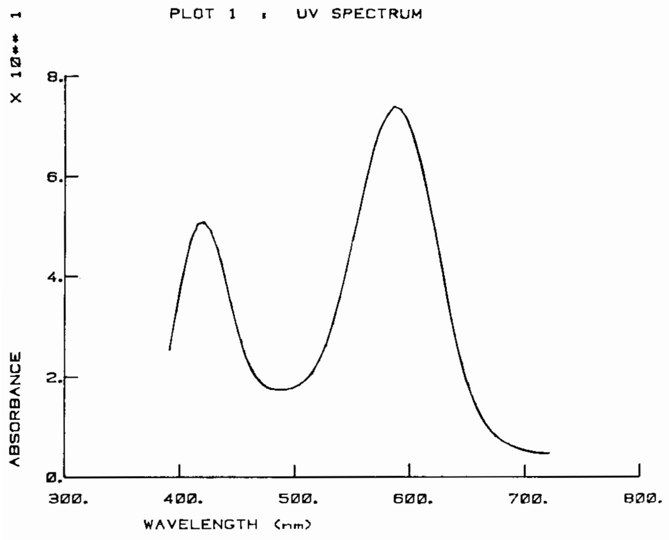
AUTOMATIC SCALING AND LOGARITHMIC PLOTTING FOR GRAPHICS/1000-II

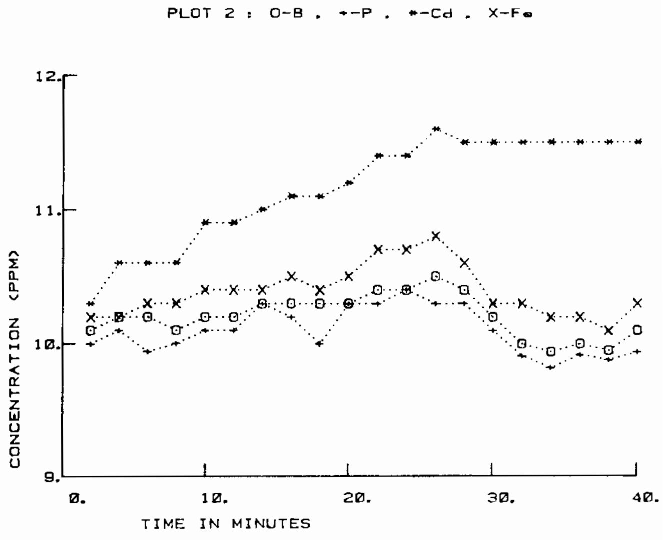
Terry O'Neal/Naval Research Lab, Washington D.C.
Elaine Mosakowski/HP

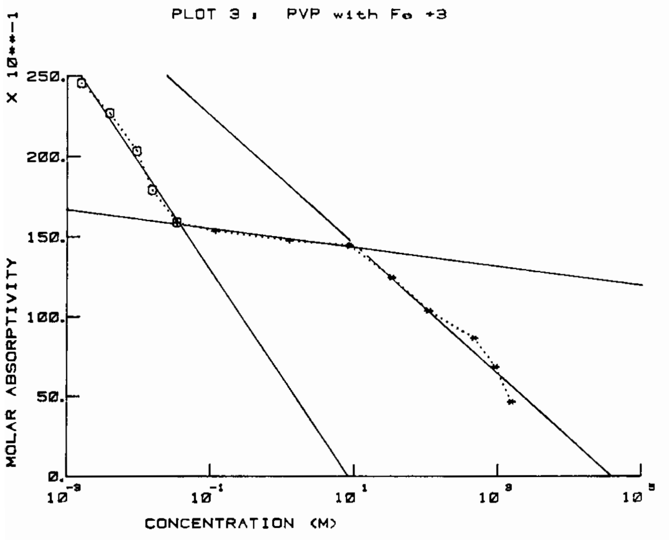
INTRODUCTION

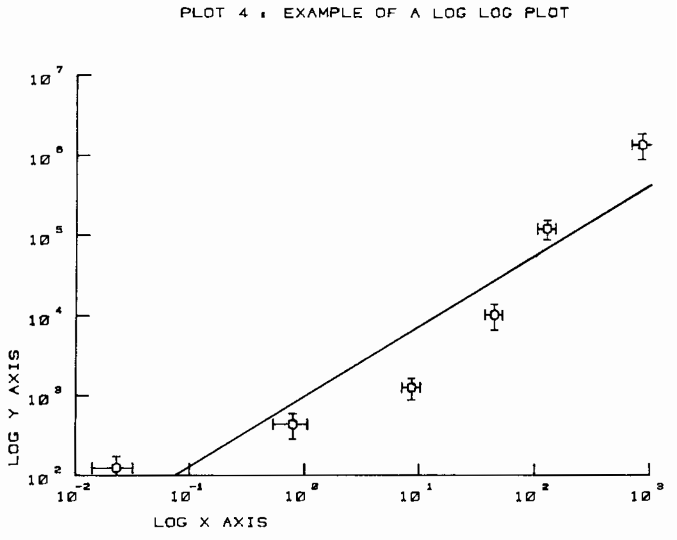
The need to represent data in a form more meaningful than tabular has prompted the development of new software for general scientific use. Hewlett-Packard's Graphics/1000-II is far superior to the old Graphics/1000 package but two major areas in plotting were overlooked. The new graphics package does not do scaling or handle logarithmic plotting.

The four subroutines in this article are FORTRAN callable by a user supplied plotting routine. Subroutine SCALE will linearly scale data. Subroutine LSCAL will logarithmically scale data. Subroutine LINAX will draw a linear axis. Subroutine LOGAX will draw a logarithmic axis. A semi-log plot can be obtained by combining linear scaling and axis drawing in one dimension with logarithmic scaling and axis drawing in the other dimension. Both axis drawing routines will draw and label tick marks.


PLOT 1 : UV SPECTRUM
X 10** 1
ABSORBANCE
WAVELENGTH (nm)
0.
2.
4.
6.
8.
300.
400.
500.
600.
700.
800.


PLOT 2 : O-B , +-P , *-Cd , X-Fe
CONCENTRATION (PPM)
TIME IN MINUTES
9.
10.
11.
12.
0.
10.
20.
30.
40.


PLOT 3 : PVP with Fe +3
X 10**-1
MOLAR ABSORPTIVITY
CONCENTRATION (M)
0.
50.
100.
150.
200.
250.


PLOT 4 : EXAMPLE OF A LOG LOG PLOT
LOG Y AXIS
LOG X AXIS

```
FTN4X,L
      SUBROUTINE SCALE (A,NP,AMIN,AMAX,TIC,IZERO,IEXP,LU,ISCAL)
C
C     THIS ROUTINE COMPUTES MAX & MIN VALUES,SCALES DATA TO
C     E FORMAT AND DETERMINES THE NUMBER OF TICK MARKS PER AXIS
C     A      - ARRAY TO BE SCALED
C     NP     - NUMBER OF POINTS IN ARRAY A
C     AMIN   - MINIMUN VALUE OF A
C     AMAX   - MAXIMUM VALUE OF A
C     TIC    - NUMBER OF TICK MARKS ON AXIS
C     IZERO  - SET TO 0 TO FORCE ORIGIN TO ZERO,NORMALLY = 1
C     IEXP   - EXPONENT OF BASE 10 TO WHICH A IS RAISED
C     LU     - LOGICAL UNIT NUMBER OF TERMINAL
C     ISCAL  - SCALING, 0 FOR AUTOMATIC, 1 FOR MANUAL
C
      DIMENSION A(256)
      ISML = 0
C
C     FIND MAX AND MIN VALUES OF A
C
      IF (ISCAL.EQ.1) GO TO 25
      AMAX = A(1)
      AMIN = A(1)
C
      DO 10 I = 1,NP
      IF (A(I).GT.AMAX) AMAX = A(I)
      IF (A(I).LT.AMIN) AMIN = A(I)
 10   CONTINUE
C
      IF (AMAX.NE.AMIN) GO TO 15
      AMAX = AMAX*1.1
      AMIN = AMIN*0.9
 15   IF (IZERO.NE.0. OR .AMIN.GE.0) GO TO 25
      WRITE (LU,20)
 20   FORMAT (" ** DATA HAS NEGATIVE VALUES ORIGIN CAN'T BE ZERO **"/)
      IZERO = 1
 25   IF (IZERO.EQ.0) AMIN = 0.
      IEXP = 0
      DIF = ABS(AMAX-AMIN)
C
C IF DIF IS LESS THAN 1/2 MAGNIFY SCALE
C
      IF (DIF.GT.0.5) GO TO 40
      ISML = 1
C
 30   DO 35 K = 1,NP
 35   A(K) = A(K)*10.
C
      AMIN = AMIN*10.
      AMAX = AMAX*10.
      IEXP = IEXP-1
      DIF = DIF*10.
      IF (DIF.LT.1.) GO TO 30
      GO TO 60
C
C     SCALE DATA TO E FORMAT AND SAVE EXPONENT (IEXP)
C
 40   IF (ABS(AMAX).GE.1. OR .ABS(AMIN).GE.1) GO TO 50
```

```
C
      DO 45 K = 1,NP
 45   A(K) = A(K)*10.
C
      AMIN = AMIN*10.
      AMAX = AMAX*10.
      IEXP = IEXP-1
      GO TO 40
C
 50   IF (ABS(AMAX).LE.1000. AND .ABS(AMIN).LE.1000) GO TO 60
C
      DO 55 K = 1,NP
 55   A(K) = A(K)/10.
C
      AMIN = AMIN/10.
      AMAX = AMAX/10.
      IEXP = IEXP+1
      GO TO 50
C
C     DETERMINE INTERVAL FACTOR
C
 60   DIF = ABS(AMAX-AMIN)
      IF (DIF.GT.5.) GO TO 65
      FACTR = 1.0
      GO TO 100
C
 65   IF (DIF.GT.10.) GO TO 70
      FACTR = 2.0
      GO TO 100
C
 70   IF (DIF.GT.25.) GO TO 75
      FACTR = 5.0
      GO TO 100
C
 75   IF (DIF.GT.50) GO TO 80
      FACTR = 10.0
      GO TO 100
C
 80   IF (DIF.GT.125) GO TO 85
      FACTR = 25.
      GO TO 100
C
 85   IF (DIF.GT.250) GO TO 90
      FACTR = 50.
      GO TO 100
C
 90   IF (DIF.GT.500) GO TO 95
      FACTR = 100.
      GO TO 100
C
 95   FACTR = 200.
C
C HANDLE ROUND-OFF ERROR
C
 100  ONE = 0.99999
      IF (AMIN.GE.0.) GO TO 105
C
C  ****  FOR NEGATIVE NUMBERS  ****
```

```
C
      TRUNK = (AMIN/FACTR)-ONE
      AMIN = AINT(TRUNK)*FACTR
      IF (AMAX.GE.0) GO TO 110
      TRUNK = AMAX/FACTR
      AMAX = AINT(TRUNK)*FACTR
      GO TO 115
C
C  ****  FOR POSITIVE NUMBERS  ****
C
 105  TRUNK = AMIN/FACTR
      AMIN = AINT(TRUNK)*FACTR
 110  TRUNK = (AMAX/FACTR)+ONE
      AMAX = AINT(TRUNK)*FACTR
C
C  ******************************
C
 115  DIF = ABS(AMAX-AMIN)
C
C     DETERMINE NUMBER OF TICK MARKS PER AXIS
C
      IF (DIF.NE.1.) GO TO 120
      TIC = 1.
      GO TO 150
C
 120  IF (DIF.GT.10) GO TO 125
      DIF = DIF*10.
      GO TO 120
C
 125  IF (DIF.LE.100) GO TO 130
      DIF = DIF/10.
      GO TO 125
C
 130  TIC = 3.
      IF (DIF.EQ.100.) GO TO 145
      IF (DIF.EQ.80.) GO TO 135
      IF (DIF.GE.60.) GO TO 150
      IF (DIF.EQ.50.) GO TO 145
      IF (DIF.EQ.40.) GO TO 135
      IF (DIF.EQ.30.) GO TO 150
      IF (DIF.EQ.25.) GO TO 145
      IF (DIF.EQ.20.) GO TO 140
      IF (DIF.EQ.15.) GO TO 150
      IF (DIF.EQ.12.5) GO TO 145
C
 135  TIC = 4.
      GO TO 150
 140  TIC = 2.
      GO TO 150
 145  TIC = 5.
 150  CONTINUE
      RETURN
      END
```

```
FTN4X,L
      SUBROUTINE LSCAL (A,NP,PMIN,PMAX,TIC,LU,ISCAL,LERR)
C
C     LOGARITHMIC SCALING ROUTINE TO COMPUTE MAX & MIN TO LOG BASE 10
C     AND DETERMINE THE NUMBER OF TICK MARKS PER AXIS
C     A      - ARRAY TO BE SCALED
C     NP     - NUMBER OF POINTS IN ARRAY X
C     PMIN   - POWER TO BASE 10 OF MINIMUM VALUE OF ARRAY A
C     PMAX   - POWER TO BASE 10 OF MAXIMUM VALUE OF ARRAY A
C     TIC    - NUMBER OF TICK MARKS ON AXIS
C     LU     - LOGICAL UNIT NUMBER FOR ERROR OUTPUT
C     ISCAL  - SCALING, 0 FOR AUTOMATIC, 1 FOR MANUAL
C     LERR   - LOG ERROR, CAN'T TAKE LOG OF A NON-POSITIVE NUMBER
C
      DIMENSION A(256)
C
C TEST ARRAY FOR NON-POSITIVE VALUES
C
      LERR = 0
C
      DO 10 I = 1,NP
      IF (A(I).GT.0) GO TO 10
      LERR = 1
 10   CONTINUE
C
      IF (LERR.EQ.1) WRITE (LU,15)
      IF (LERR.EQ.1) GO TO 90
      IF (ISCAL.EQ.1) GO TO 25
 15   FORMAT(" STOP ! YOU CAN'T USE LOGARITHMIC SCALING ROUTINE (LSCAL)"
     -/"         FOR NON-POSITIVE VALUES, USE A LINEAR SCALING ROUTINE")
C
C DETERMINE VALUES OF MAX AND MIN
C
      AMIN = A(1)
      AMAX = AMIN
C
      DO 20 I = 2,NP
      IF (AMIN.GT.A(I)) AMIN = A(I)
      IF (AMAX.LT.A(I)) AMAX = A(I)
 20   CONTINUE
C
      IF (AMIN.NE.AMAX) GO TO 30
      AMIN = 0.9*AMIN
      AMAX = 1.1*AMAX
      GO TO 30
 25   AMIN = PMIN
      AMAX = PMAX
C
C COMPUTE LOG OF MIN AND MAX VALUES
C
 30   POWL = ALOGT(AMIN)
      MINP = POWL
      POWH = ALOGT(AMAX)
      MAXP = POWH
```

```
C
      IF (ABS(FLOAT(MAXP)-POWH).LT.1.0E-05) GO TO 35
      IF (POWH.GE.0.0) MAXP = MAXP + 1
 35   IF (ABS(FLOAT(MINP)-POWL).LE.1.0E-05) GO TO 40
      IF (POWL.GE.0) GO TO 40
      MINP = MINP - 1
 40   PMIN = MINP
      PMAX = MAXP
C
C MAKE DIFFERENCE IN EXPONENTS OF MAX & MIN A MULTIPLE OF 1,2,3,4 OR 5
C
      IDIF = ABS(PMAX-PMIN)
      IF (IDIF.LE.6) GO TO 55
      IF (IDIF.EQ.7. OR .IDIF.EQ.9) PMAX = PMAX + 1
      IF (IDIF.LE.10) GO TO 55
      LDIF = 15
 45   IF (LDIF.GE.IDIF) GO TO 50
      LDIF = LDIF + 5
      GO TO 45
 50   PMAX = PMIN + LDIF
C
C DETERMINE NUMBER OF TICK MARKS PER AXIS
C
 55   IDIF = ABS(PMAX-PMIN)
      IF (IDIF.NE.1) GO TO 60
      TIC = 1
      GO TO 90
C
 60   IF (IDIF.GT.10) GO TO 65
      IDIF = IDIF*10
      GO TO 60
C
 65   IF (IDIF.LE.100) GO TO 70
      IDIF = IDIF/10
      GO TO 65
C
 70   TIC = 3
      IF (IDIF.EQ.100) GO TO 85
      IF (IDIF.EQ.80) GO TO 75
      IF (IDIF.GE.60) GO TO 90
      IF (IDIF.EQ.50) GO TO 85
      IF (IDIF.EQ.40) GO TO 75
      IF (IDIF.EQ.30) GO TO 90
      IF (IDIF.EQ.25) GO TO 85
      IF (IDIF.EQ.20) GO TO 80
      IF (IDIF.EQ.15) GO TO 90
C
 75   TIC = 4
      GO TO 90
C
 80   TIC = 2
      GO TO 90
C
 85   TIC = 5
C
 90   CONTINUE
      RETURN
      END
```

```
FTN4X,L
      SUBROUTINE LINAX(IAXIS,AMIN,AMAX,TIC,LUG,HW,HH)
      DIMENSION ITICM(6)
C
C LINEAR AXIS DRAWING ROUTINE
C
C IAXIS - 1=X AXIS , 2=Y AXIS
C AMIN  - MINIMUM VALUE OF AXIS
C AMAX  - MAXIMUM VALUE OF AXIS
C TIC   - NUMBER OF TICK MARKS ALONG AXIS
C LUG   - LOGICAL UNIT NUMBER FOR GRAPHICS OUTPUT
C HW    - CHARACTER CELL WIDTH
C HH    - CHARACTER CELL HEIGHT
C
C SET AXIS LENGTH IN WORLD COORDINATE SYSTEM (WCS)
C
      IF (IAXIS.EQ.1) ALEN = 114.
      IF (IAXIS.EQ.2) ALEN = 70.
C
C DEFINE ORIGIN
C
      XO = 19.
      YO = 10.
      TICM = AMIN
      JTICM = INT (TICM)
      CALL J2MOV (XO, YO)
      IF (IAXIS.EQ.2) GO TO 100
C
C DRAW X AXIS
C
      CALL J2DRW (ALEN+XO, YO)
C
C LABEL X ORIGIN
C
      CALL CMOVE (JTICM, N, ITICM)
      CALL J2MOV (XO, YO-1.3*HH)
      CALL JFONT(1)
      CALL JTEXH (N, ITICM)
C
C DRAW X TICK MARKS
C
      DO 50 K = 1,TIC
      TICK = ALEN*(FLOAT(K)/TIC)
      CALL J2MOV (TICK+XO, YO)
      CALL J2DRW (TICK+XO, YO+2.0)
C
C LABEL X TICK MARKS
C
      TICM = TICM + ((AMAX-AMIN)/TIC)
      JTICM = INT (TICM)
      CALL CMOVE (JTICM, N, ITICM)
      CALL J2MOV (TICK+XO, YO-1.3*HH)
      CALL JTEXH (N, ITICM)
 50   CONTINUE
      GO TO 200
C
C DRAW Y AXIS
C
 100  CALL J2DRW (XO, ALEN+YO)
```

```
C
C LABEL Y ORIGIN
C
      CALL CMOVE (JTICM, N, ITICM)
      RN = REAL (N)
      CALL J2MOV (XO-HW*(RN+3.)/2., YO)
      CALL JTEXH (N, ITICM)
C
C DRAW Y TICK MARKS
C
      DO 150 K = 1,TIC
      TICK = ALEN*(FLOAT(K)/TIC)
C
C LABEL Y TICK MARKS
C
      TICM = TICM + ((AMAX-AMIN)/TIC)
      JTICM = INT (TICM)
      CALL J2MOV (XO, TICK+YO)
      CALL J2DRW (XO+HW, TICK+YO)
      CALL CMOVE (JTICM, N, ITICM)
      RN = REAL (N)
      CALL J2MOV (XO-HW*(RN+3.)/2., TICK+YO)
      CALL JTEXH (N, ITICM)
  150 CONTINUE
  200 RETURN
      END
```

```
FTN4X,L
      SUBROUTINE LOGAX(IAXIS,PMIN,PMAX,TIC,LUG,HW,HH)
C
C LOGRATHMETIC AXIS DRAWING ROUTINE
C
C IAXIS - 1=X AXIS , 2=Y AXIS
C PMIN  - POWER TO BASE 10 OF MINIMUM VALUE OF AXIS
C PMAX  - POWER TO BASE 10 OF MAXIMUM VALUE OF AXIS
C TIC   - NUMBER OF TICK MARKS ALONG AXIS
C LUG   - LOGICAL UNIT NUMBER FOR GRAPHICS OUTPUT
C HW    - CHARACTER CELL WIDTH
C HH    - CHARACTER CELL HEIGHT
C
C SET AXIS LENGTH IN WORLD COORDINATE SYSTEM (WCS)
C
      DIMENSION IPMIN(3), IIEXP(3)
      INTEGER MIN
      MIN = INT (PMIN)
C
      IF(IAXIS.EQ.1)ALEN=114.
      IF(IAXIS.EQ.2)ALEN=70.
C
C DEFINE ORIGIN
C
      XO=19.
      YO=10.
      IEXP=MIN
      CALL J2MOV (XO,YO)
      IF(IAXIS.EQ.2)GOTO 25
C
C DRAW X AXIS
C
      CALL J2DRW (ALEN+XO,YO)
C
C LABEL X ORIGIN
C
      CALL CMOVE (IEXP, N, IIEXP)
      RN = REAL (N)
      CALL J2MOV (XO-RN*HW/2., YO-1.8*HH)
      CALL JTEXH (2, 2H10)
      CALL J2MOV (XO-(RN-2.)*HW/2., YO-1.3*HH)
      CALL JCSIZ (.75*HW, .75*HH, 0.)
      CALL JTEXH (N, IIEXP)
      CALL JCSIZ (HW, HH, 0.)
C
C DRAW X TICK MARKS
C
      DO20  K=1,TIC
      TICK=ALEN*(FLOAT(K)/TIC)
      CALL J2MOV (TICK+XO,YO)
      CALL J2DRW (TICK+XO,YO+2.0)
C
C LABEL X TICK MARKS
C
      IEXP=IEXP+(PMAX-PMIN)/TIC
```

```
C
      CALL CMOVE (IEXP, N, IIEXP)
      RN = REAL (N)
      CALL J2MOV (TICK+XO-RN*HW/2., YO-1.8*HH)
      CALL JTEXH (2, 2H10)
      CALL J2MOV (TICK+XO-(RN-2.)*HW/2., YO-1.3*HH)
      CALL JCSIZ (.75*HW, .75*HH, 0.)
      CALL JTEXH (N, IIEXP)
      CALL JCSIZ (HW, HH, 0.)
 20   CONTINUE
      GOTO 35
C
C DRAW Y AXIS
C
 25   CALL J2DRW (XO,ALEN+YO)
C
C LABEL Y ORIGIN
C
      CALL J2MOV (XO-(4.*HW), YO-0.5*HH)
      CALL JTEXH (2,2H10)
      CALL CMOVE (MIN, N, IPMIN)
      CALL J2MOV (XO-(3.*HW), YO)
      CALL JCSIZ (.75*HW, .75*HH, 0.)
      CALL JTEXH (N, IPMIN)
      CALL JCSIZ (HW, HH, 0.)
C
C DRAW Y TICK MARKS
C
      DO30   K=1,TIC
      TICK=ALEN*(FLOAT(K)/TIC)
C
C LABEL Y TICK MARKS
C
      IEXP = IEXP + (PMAX-PMIN)/TIC
      CALL J2MOV (XO,TICK+YO)
      CALL J2DRW (XO+2.5,TICK+YO)
      CALL J2MOV (XO-(4.*HW), TICK+YO-0.5*HH)
      CALL JTEXH (2, 2H10)
      CALL CMOVE (IEXP, N, IIEXP)
      CALL J2MOV (XO-(3.*HW), TICK+YO)
      CALL JCSIZ (.75*HW, .75*HH, 0.)
      CALL JTEXH (N, IIEXP)
      CALL JCSIZ (HW, HH, 0.)
 30   CONTINUE
 35   RETURN
      END
```

```
FTN4X,L
      SUBROUTINE CMOVE (SOURC, N, DEST)
C
      INTEGER ISOURC(3), DEST(3), CHAR, BLANK, N
      INTEGER SOURC
      DATA BLANK/2H  /
C
      ENCODE (6, 5, ISOURC) SOURC
 5    FORMAT (I6)
      N = 0
C
      DO 10 J = 0,5
      CALL MPUTC (BLANK, J, DEST)
 10   CONTINUE
C
C
      DO 20 I = 0, 5
      CHAR = MGETC (I, ISOURC)
      IF (CHAR.EQ.BLANK) GO TO 20
        CALL MPUTC (CHAR, N, DEST)
        N = N + 1
 20   CONTINUE
C
      RETURN
      END
```

PROGRAM TO PROGRAM DATA PASSING USING FIFO QUEUES IN SSGA

Matt Betts/Fisher Body Division
General Motors Corp.

A servo control application currently in development requires that one program (hereafter called the producer) should maintain a number of pre-calculated sets of data available for another program (hereafter called the consumer). The consumer program will operate in real time and be interrupt driven. The producer program will operate in background using a complicated algorithm to generate the sets of values. The problem is how to allow the two programs to run asynchronously while passing sets of data from the producer program to the consumer program. The producer program must stay sufficiently ahead of the consumer program to insure that at least one set of values be available at all times. The best solution is to pass the data through some form of First In First Out (FIFO) queue. Three alternatives were examined: using Class I/O, using System Common, and using Sub System Global Area.

FIFO USING CLASS I/O

There are several drawbacks associated with using class I/O for this application. First is that there has to be a mechanism for passing the class number from the program that allocated it to all other programs using that class. This requires either a Father/Son relationship or the use of a location in system common. Each of these alternatives has its own restrictions which may not be desirable. Second, there is no mechanism to limit the number of entries the producer may have waiting in the class queue. Either the producer runs until SAM is full or the consumer must tell the producer (using system common or another class) when it consumes an entry, thereby allowing the producer to monitor queue size. Finally, there is no way to clear the class queue of unwanted entries except by removing them one by one. These drawbacks make the class I/O approach undesirable.

FIFO IN SYSTEM COMMON

Subroutines can easily be written to implement FIFO queues in system common. The only problem with this approach is that any other program that uses system common must know the location and size of the queue area. This becomes difficult in multiuser development systems; also, if the queue size is changed all other programs using system common may have to be edited, recompiled and reloaded. These problems made the system common approach undesirable.

FIFO In SSGA

Placing the queue area in Sub System Global Area (SSGA) effectively eliminates the drawbacks of both the class I/O method and the system common method. Space for all queues and their associated pointers is reserved at system generation time. In order for a program to access SSGA, it must have been loaded with SSGA included (using the loadr OP,SS command) and know the entry point for the data desired. This makes the queues relatively inaccessible to programs not specifically using them.

IMPLEMENTATION OF FIFO QUEUES IN SSGA

A short piece of assembler code was written to define the queue area. The queues were laid out as a linked list. This was done to allow changes to be made to the queue lay out without requiring software changes. Each queue area has the following: a constant which is the size of the queue in words, a put pointer variable, a get pointer variable and a space for the queue. A dummy entry with zero as the size parameter after the last queue signifies the end of the queue. The following is an example queue layout:

```
ASMB,R,L,C
      NAM QQSSG,30 QUEUE LIBRARY <811121.1118>
      ENT QQSSG
*     STARTING ADRESS OF QUEUE AREA
QQSSG DEF #QQS1
*
*     QUEUE # 1
#QQS1 DEC 300         SIZE
#QQP1 NOP             PUT POINTER
#QQG1 NOP             GET POINTER
#QQQ1 BSS 300         QUEUE
*
*     QUEUE # 2
#QQS2 DEC 100         SIZE
#QQP2 NOP             PUT POINTER
#QQG2 NOP             GET POINTER
#QQQ2 BSS 100         QUEUE
*
*     QUEUE # 3
#QQS3 DEC 50          SIZE
#QQP3 NOP             PUT POINTER
#QQG3 NOP             GET POINTER
#QQQ3 BSS 50          QUEUE
*
*     QUEUE # 4
#QQS4 DEC 0           END MARK
*
*     END
      END
```

The above routine was included in the system generation. It reserved space for the queues, set up the queue parameters, and provided the entry point QQSSG.

Since Fortran cannot use entry points directly, an assembly language subroutine QQADR was written to provide the address of the queue based on the queue number. This routine was called by the Fortran subroutine QQPAR to provide the parameters required for queue manipulation. The routines QQADR and QQPAR follow.

```
ASMB,R,L,C
      NAM QQADR,7 QUEUE LIBRARY <811121.1118>
      ENT QQADR
      EXT QQSSG,.ENTR
A     EQU 0
B     EQU 1
QQNO  BSS 1           .ENTR PARAMETER DEST (QUEUE #)
QQADR NOP             RETURN ADDRESS STORED HERE AS USUAL
      JSB .ENTR       GET NUMBER OF DESIRED QUEUE
      DEF QQNO
*
      LDA QQSSG       GET ADDRESS OF FIRST QUEUE SIZE
      STA CURAD       SAVE IT IN CURRENT ADDRESS
```

```
*
      LDA QQNO,I    IS QUEUE NUMBER
      SZA,RSS       = 0 ?
      JMP ERRET     YES - ERROR
      SSA           < 0 ?
      JMP ERRET     YES - ERROR
      CMA,INA       NEGATE QUEUE NUMBER
*
LOOP  ISZ A         AT DESIRED QUEUE ?
      JMP NEXT      NO  - LOOP
*
      LDA CURAD     YES - AT QUEUE - POINT AT QUEUE ITSELF
      ADA =B3
      JMP QQADR,I   RETURN
*
NEXT  LDB CURAD,I   GET SIZE OF THIS QUEUE
      ADB CURAD     MOVE CURRENT ADDRESS TO NEXT QUEUE SIZE
      ADB =B3
      STB CURAD     SAVE CURRENT ADDRESS
      LDB CURAD,I   IF NEXT QUEUE HAS ZERO LENGTH
      SZB           THEN IT IS THE END MARKER
      JMP LOOP      OTHERWISE - TRY AGAIN
*
ERRET CLA           SET ERROR RETURN
      JMP QQADR,I   RETURN
*
CURAD BSS 1
      END


C
C     SUBROUTINE TO GET QUEUE PARAMETERS FROM SSG (QUEUE NUMBER,
C     ============  QUEUE ADDRESS, PUT POINTER ADDRESS, GET POINTER
C                   ADDRESS, QUEUE SIZE ADDRESS, ERROR FLAG)
C                 ================================================
      SUBROUTINE QQPAR (QUENO, QUEAD, PUTAD, GETAD, SIZAD, ERROR)
     *,QUEUE LIBRARY <811121.1118>
      IMPLICIT INTEGER (A - Z)
      LOGICAL ERROR
C
C     CLEAR ERROR FLAG
         ERROR = .FALSE.
C
C     GET QUEUE ADDRESS
         QUEAD = QQADR (QUENO)
C
C     IF(ERROR)
         IF(.NOT. (QUEAD .EQ. 0)) GO TO 10
C
C        SET ERROR FLAG
            ERROR = .TRUE.
            GO TO 20
C
C     ELSE
10       CONTINUE
C
C        CALCULATE OTHER ADDRESSES
            GETAD = QUEAD - 1
            PUTAD = GETAD - 1
            SIZAD = PUTAD - 1
```

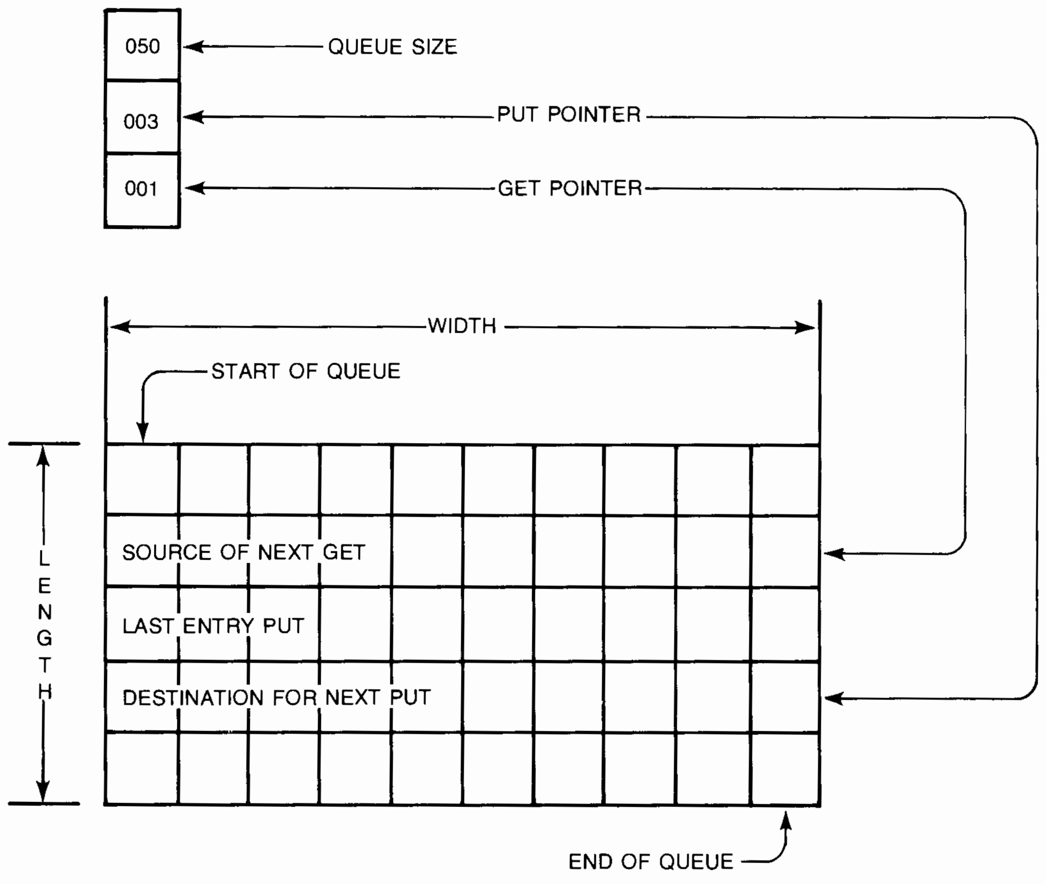
```
C
C      FI
20         CONTINUE
C      RETURN
           RETURN
           END
```

QQADR returns zero as the queue address if the queue doesn't exist. If this occurs, QQPAR sets the error flag.

Knowing the addresses of the queue and its parameters allow queue manipulating subroutines to be written using the HP supplied routines IPUT and IGET. These routines read from and write to address parameters.

QUEUE MANIPULATION SUBROUTINES

The queue manipulation routines were written to allow the queues to have width as well as length. The following is an illustration of the queue concept.



NOTE:
1. WIDTH = WIDTH IN WORDS
2. A FULL QUEUE WILL HAVE LENGTH — 1 ENTRIES
3. $2 \leq$ LENGTH $\leq$ SIZE
4. $1 \leq$ WIDTH $\leq$ (SIZE/2)

This arrangement allows real variables, groups of variables, or strings to be passed with one call. The length of a queue can be determined for a given width in the following manner.

```
          CALL QQPAR (QUENO, QUEAD, PUTAD, GETAD, SIZAD, ERROR)
          LENTH = IGET (SIZAD) / WIDTH
```

All subsequent queue manipulations must use the same length and width. By specifying the width and calculating the length, the size of the queue area can be altered without need of recompilation; therefore, programs will make maximal use of the queue area available.

Listings of the queue manipulating routines written to date along with appropriate comments are following.

Clear the specified queue. The Error flag is set if the queue doesn't exist.

```
C     SUBROUTINE CLEAR QUEUE (QUEUE NUMBER, ERROR)
C     =============================================
      SUBROUTINE QQCLR (QUENO, ERROR)
     *,QUEUE LIBRARY <811121.1118>
      IMPLICIT INTEGER (A - Z)
      LOGICAL ERROR
C
C     GET QUEUE PARAMETERS
         CALL QQPAR (QUENO, QUEAD, PUTAD, GETAD, SIZAD, ERROR)
C
C     IF(NOT ERROR)
         IF(ERROR) GO TO 10
C
C        PUT POINTER <== GET POINTER
            CALL IPUT(PUTAD,0)
            CALL IPUT(GETAD,0)
C
C     FI
10       CONTINUE
C
C     RETURN
         RETURN
         END

C*
```

Is the specified queue empty? The Error flag is set if the queue doesn't exist. The Empty flag is set if the queue is empty.

```
C     SUBROUTINE CHECK FOR EMPTY QUEUE (QUEUE NUMBER, ERROR, EMPTY)
C     ================================================================
      SUBROUTINE QQEMT (QUENO, ERROR, EMPTY)
     *,QUEUE LIBRARY <811121.1118>
      IMPLICIT INTEGER (A - Z)
      LOGICAL ERROR, EMPTY
C
C     GET QUEUE PARAMETERS
         CALL QQPAR (QUENO, QUEAD, PUTAD, GETAD, SIZAD, ERROR)
C
C     IF(NOT ERROR)
         IF(ERROR) GO TO 30
C
C        IF(PUT POINTER == GET POINTER)
            IF(.NOT. (IGET (PUTAD) .EQ. IGET (GETAD))) GO TO 10
C
C           EMPTY <== TRUE
               EMPTY = .TRUE.
C
            GO TO 20
C        ELSE
C
C           EMPTY <== FALSE
10             EMPTY = .FALSE.
C
C        FI
20          CONTINUE
C
C     FI
30       CONTINUE
C
C     RETURN
         RETURN
         END
```

Is the specified queue full? The Error flag is set if the queue doesn't exist. The Full flag is set if the queue is full.

```
C     SUBROUTINE CHECK FOR FULL QUEUE (QUEUE NUMBER, ERROR, LENGTH,
C     ==============================  FULL)
C                                    =========================
      SUBROUTINE QQFUL (QUENO, ERROR, LENGTH, FULL)
     *,QUEUE LIBRARY <811121.1118>
      IMPLICIT INTEGER (A - Z)
      LOGICAL ERROR, FULL
C
C     GET QUEUE PARAMETERS
         CALL QQPAR (QUENO, QUEAD, PUTAD, GETAD, SIZAD, ERROR)
C
C     IF(NOT ERROR)
         IF(ERROR) GO TO 30
C
C        IF((PUT POINTER MOD LENGTH) + 1 == GET POINTER)
            IF(.NOT. ((MOD (IGET (PUTAD) + 1,LENGTH)) .EQ.
     *                 IGET (GETAD))) GO TO 10
C
C           FULL <== TRUE
               FULL = .TRUE.
C
            GO TO 20
C        ELSE
C
C           FULL <== FALSE
10             FULL = .FALSE.
C
C        FI
20          CONTINUE
C
C     FI
30       CONTINUE
C
C     RETURN
         RETURN
         END
```

How many entries are there currently in the specified queue? The Error flag is set if the queue doesn't exist.

```
C      SUBROUTINE POPULATION IN QUEUE (QUEUE NUMBER, ERROR, LENGTH,
C      ==============================  POPULATION)
C                                     ===========================
       SUBROUTINE QQPOP (QUENO, ERROR, LENGTH, POP)
      *,QUEUE LIBRARY <811121.1118>
       IMPLICIT INTEGER (A - Z)
       LOGICAL ERROR
C
C      GET QUEUE PARAMETERS
          CALL QQPAR (QUENO, QUEAD, PUTAD, GETAD, SIZAD, ERROR)
C
C      IF(NOT ERROR)
           IF(ERROR) GO TO 30
C
C         IF (PUT POINTER >= GET POINTER)
             IF(.NOT. (IGET (PUTAD) .GE. IGET (GETAD))) GO TO 10
C
C            POPULATION <== PUT POINTER - GET POINTER
                POP = IGET (PUTAD) - IGET (GETAD)
C
             GO TO 20
C         ELSE
C
C            POPULATION <== PUT POINTER - GET POINTER + LENGTH
10              POP = IGET (PUTAD) - IGET (GETAD) + LENGTH
C
C         FI
20           CONTINUE
C
C      FI
30        CONTINUE
C
C      RETURN
          RETURN
          END
```

Get a point (array) from the specified queue. The Error flag is set if the queue doesn't exist or the queue is empty. WIDT2 is the width of the array receiving the data. WIDTH or WIDT2 words will be transferred, whichever is less.

```
C     SUBROUTINE GET A POINT FROM QUEUE (QUEUE NUMBER, ERROR, LENGTH,
C     ================================= WIDTH, ARRAY, WIDT2)
C                                       ===========================
      SUBROUTINE QQGET (QUENO, ERROR, LENGTH, WIDTH, ARRAY, WIDT2)
     *,QUEUE LIBRARY <811121.1118>
      IMPLICIT INTEGER (A - Z)
      LOGICAL ERROR, EMPTY
      DIMENSION ARRAY (1)
C
C     GET QUEUE PARAMETERS
         CALL QQPAR (QUENO, QUEAD, PUTAD, GETAD, SIZAD, ERROR)
C
C     IF(NOT ERROR)
         IF(ERROR) GO TO 40
C
C        IF(EMPYT QUEUE (QUEUE NUMBER, ERROR, EMPTY))
            CALL QQEMT (QUENO, ERROR, EMPTY)
            IF(.NOT. EMPTY) GO TO 10
C
C           ERROR <== TRUE
               ERROR = .TRUE.
C
            GO TO 30
C        ELSE
C
C           ERROR <== FALSE
10             ERROR = .FALSE.
C           LOOPSIZ <== MIN (WIDTH, WIDT2)
               LSIZ = MIN0 (WIDTH, WIDT2)
C
C           DO (I = 1: I <= LOOPSIZ: I = I + 1)
               DO 20 I = 1, LSIZ, 1
C
C              ARRAY (I) <== QUEUE (GET POINTER, I)
                  ARRAY (I) = IGET (QUEAD + IGET (GETAD) *
     *                           WIDTH + (I - 1))
C
20          CONTINUE
C           ENDDO
C
C           GET POINTER <== (GET POINTER MOD LENGTH) + 1
               CALL IPUT (GETAD, (MOD (IGET (GETAD) + 1, LENGTH)))
C
C        FI
30          CONTINUE
C
C     FI
40       CONTINUE
C
C     RETURN
         RETURN
         END
```

Write a point (array) to the specified queue. The Error flag is set if the queue doesn't exist or the queue is full. WIDTH and WIDT2 are the same as in get a point.

```
C     SUBROUTINE PUT A POINT TO QUEUE (QUEUE NUMBER, ERROR, LENGTH,
C     ===============================  WIDTH, ARRAY, WIDT2)
C                                     ============================
      SUBROUTINE QQPUT (QUENO, ERROR, LENGTH, WIDTH, ARRAY, WIDT2)
     *,QUEUE LIBRARY <811121.1118>
      IMPLICIT INTEGER (A - Z)
      LOGICAL ERROR, FULL
      DIMENSION ARRAY (1)
C
C     GET QUEUE PARAMETERS
         CALL QQPAR (QUENO, QUEAD, PUTAD, GETAD, SIZAD, ERROR)
C
C     IF(NOT ERROR)
         IF(ERROR) GO TO 40
C
C        IF(FULL QUEUE (QUEUE NUMBER, ERROR, LENGTH, FULL))
            CALL QQFUL (QUENO, ERROR, LENGTH, FULL)
            IF(.NOT. FULL) GO TO 10
C
C           ERROR <== TRUE
               ERROR = .TRUE.
C
            GO TO 30
C        ELSE
C
C           ERROR <== FALSE
10             ERROR = .FALSE.
C           LOOPSIZ <== MIN (WIDTH, WIDT2)
               LSIZ = MIN0 (WIDTH, WIDT2)
C
C           DO (I = 1: I <= LOOPSIZ: I = I + 1)
               DO 20 I = 1, LSIZ, 1
C
C              QUEUE (PUT POINTER, I) <== ARRAY (I)
                  CALL IPUT (QUEAD + IGET (PUTAD) *
     *                      WIDTH + (I - 1), ARRAY(I))
C
20             CONTINUE
C           ENDDO
C
C           PUT POINTER <== (PUT POINTER MOD LENGTH) + 1
               CALL IPUT (PUTAD,(MOD (IGET (PUTAD) + 1, LENGTH)))
C
C        FI
30          CONTINUE
C
C     FI
40       CONTINUE
C
C     RETURN
         RETURN
         END
```

OTHER CONSIDERATIONS

If more than one program will be using the Get subroutine on the same queue, care should be taken to ensure that the subroutine calls do not overlap. Similar care should be taken when using the Put and Clear subroutines. Overlapping calls using the same pointer will cause improper pointer operation. This problem can be prevented by using Resource Numbers.

THE FUNDAMENTALS OF HP-IB ADDRESSING

Neal Kuhn/DSD,Applications Development

HP-IB is a powerful interfacing concept that allows multiple instruments to be connected to the same bus. This concept helps to economize the system cost with respect to cabling, interface hardware, and user software effort. Programming devices with HP-IB is straightforward; however, if full utility is desired a person implementing an HP-IB system should understand how HP-IB addresses are created and used. The purpose of this article is to describe the HP-IB addressing scheme, and discuss how it is used in an HP 1000 Computer System.

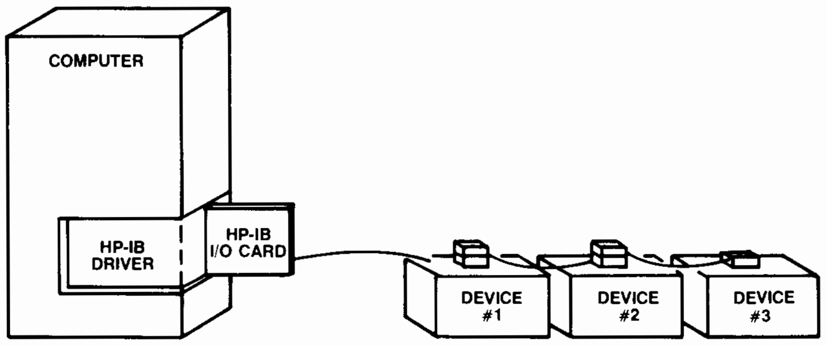


Figure 2

Figure 1 illustrates the typical connection pattern for a group of HP-IB instruments. The bus itself is a multidrop arrangement in which each device has its own address. The address is set with switches which are usually located on the back of the instrument, or with jumpers which are usually on the circuit board associated with the instrument's I/O. Each manufacturer installs the switches or jumpers in a different place, and labeling is not always clear. Therefore, care should be taken to ascertain which switch is the most significant bit, and which position is "1" or "0". Application Note Series 401 contains information regarding the setting of the HP-IB address switches for many of the HP-IB instruments and peripherals. A typical HP-IB address grouping is shown in Figure 2.

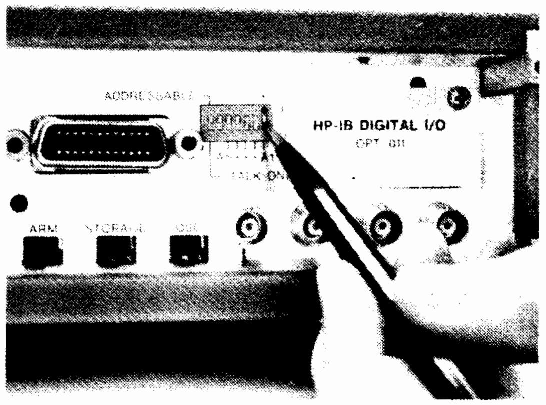


Figure 1

Five address bits are used to obtain the binary pattern for up to 32 device addresses, but only 30 different settings can be used for instruments. One address is withheld by the controller for use as its own address, since the I/O card itself is an HP-IB device. For the HP 1000, this address is address 0. HP-IB address 31 (37B) is defined by the HP-IB for the special purpose of telling everyone to stop talking (untalk) or stop listening (unlisten). All devices respond to the untalk and unlisten commands regardless of their addresses. Addresses 1 to 30 (36B) may be arbitrarily assigned to your device.

When the HP-IB's ATN (attention) management line is asserted, information is sent out from the controller over the bus data lines. This information tells the devices which HP-IB command to perform, or which device should talk and which devices should listen during a transfer of data. A coding scheme is established for HP-IB where a parity bit and two data bits are appended to the five bit address to form an eight bit "command" byte. The parity bit is not usually used. A device looks for its address in the command byte, and also looks at the two extra bits to determine what the command was. The choices for these two bits are:

WORD	MEANING
00 XXXXX	This is a bus "universal" command and xxxxx is the command to perform. The universal commands are used to clear devices, trigger them, and to set up polling sequences for service requests.
01 XXXXX	This is a "listen" command, and device xxxxx is to listen to the bus for data.
10 XXXXX	This is a "talk" command, and device xxxxx is to send data onto the bus.
11 XXXXX	This is "secondary" command, and xxxxx is a device dependent function code for the currently addressed device.

When the command bits are combined with the device address or function code, a seven bit ASCII character is formed. Note for universal commands that the characters are non-printing. Table 1 illustrates the ASCII characters formed for talker and listener combinations of device address and command bits.

For an example, if the ASCII command string "_?5L" is sent onto the bus from the controller, everyone would untalk (_), everyone would unlisten (?), the device set to address 5 is told to listen for data (5), and the device set to address 12 is told to talk (L). After the ATN management line is de-asserted, device 12 will begin talking, device 5 will listen, and everyone else will ignore the transaction. Note that device addresses 5 and 12 were arbitrarily selected.

According to the IEEE-488 standard, a device that is made a listener will continue to listen until it is made a talker, or the unlisten command is sent.* Assigning a new listener will not unlisten the previous one. This is the technique used by HP-IB to assign multiple listeners. The active talker will change when it is made a listener, or another device is made a talker, or the untalk command is sent. To assure that the proper devices are talking and listening, the convention usually adopted is to always send the untalk and unlisten commands first, and then set up the appropriate talkers and listeners.

Communication with devices over HP-IB can be accomplished in two different ways. One method is referred to as "automatic addressing", and the other is referred to as "direct addressing". Automatic addressing is a simple and straightforward technique which can be used for a majority (if not all) of HP-IB transactions.

Automatic addressing can utilize the standard READ,WRITE/PRINT contexts and formatting procedures. For READ statements, it is assumed that the HP 1000 is the listener, and the addressed device is the talker. For WRITE and PRINT statements, the HP 1000 is the talker and the addressed device is the listener.

**There are other techniques to unlisten or untalk an instrument (such as interface clear), but they are unaesthetic, and are used as bail out measures. Also, the untalk for my listen address is optional, but normally used. The only time that this option is not used is with terminals where the keyboard (Talker) and the display (Listener) are both active.*

Table 1. ASCII Character Equivalents for Talker and Listener Values

COMMAND BITS	SWITCH SETTINGS	OCTAL VALUE OF SWITCHES	LISTENER (XX=01)	TALKER (XX=10)
XX	0 0 0 0 0	0	SPACE	@
XX	0 0 0 0 1	1	!	A
XX	0 0 0 1 0	2	"	B
XX	0 0 0 1 1	3	#	C
XX	0 0 1 0 0	4	$	D
XX	0 0 1 0 1	5	%	E
XX	0 0 1 1 0	6	&	F
XX	0 0 1 1 1	7	'	G
XX	0 1 0 0 0	10	(	H
XX	0 1 0 0 1	11	)	I
XX	0 1 0 1 0	12	*	J
XX	0 1 0 1 1	13	+	K
XX	0 1 1 0 0	14	,	L
XX	0 1 1 0 1	15	-	M
XX	0 1 1 1 0	16	.	N
XX	0 1 1 1 1	17	/	O
XX	1 0 0 0 0	20	0	P
XX	1 0 0 0 1	21	1	Q
XX	1 0 0 1 0	22	2	R
XX	1 0 0 1 1	23	3	S
XX	1 0 1 0 0	24	4	T
XX	1 0 1 0 1	25	5	U
XX	1 0 1 1 0	26	6	V
XX	1 0 1 1 1	27	7	W
XX	1 1 0 0 0	30	8	X
XX	1 1 0 0 1	31	9	Y
XX	1 1 0 1 0	32	:	Z
XX	1 1 0 1 1	33	;	[
XX	1 1 1 0 0	34	<	\
XX	1 1 1 0 1	35	=	]
XX	1 1 1 1 0	36	>	^
XX	1 1 1 1 1	37	?	_

NOTE: The two shaded values have special uses. Address "0" is used by the computer. Address 37B is used by HP-IB to UNTALK or UNLISTEN all devices.

The Device Reference Table (DRT) structure in the HP 1000 lends itself well to HP-IB programming. When a READ, WRITE, or PRINT statement is executed, the LU specified is converted into the respective EQT number of the HP-IB and a subchannel number. The subchannel number is the HP-IB device's address (1 to 36B). The HP-IB driver "automatically" converts the subchannel number into the proper HP-IB talker or listener code and performs the following:

For a READ statement:

1. ATN is asserted by the HP 1000
2. UNTALK (_) is sent
3. UNLISTEN (?) is sent
4. The TALKER character for the specified LU is sent (see table 1)
5. ATN is de-asserted
6. The device talks and the HP 1000 listens

For a WRITE or PRINT statement:

1. ATN is asserted by the HP 1000
2. UNTALK (_) is sent
3. UNLISTEN (?) is sent
4. The LISTEN character for the specified LU is sent
5. ATN is de-asserted
6. The HP 1000 talks and device listens

The following statement illustrates an example of how automatic addressing is used. If a device requires the characters "T3" to initiate a measurement, the statement:

```
      WRITE (28,101)
  101 FORMAT ("T3")
```

could be used if the device was assigned to LU 28. Unformatted (free field format) and binary readings can also be used.

There are times when it is desirable to continue a READ or WRITE function without altering talk or listen addresses. This can be accomplished by using an LU number that points to the bus itself (subchannel 0). Since talking or listening to subchannel 0 would mean talking to itself, the system assumes that you wish to perform the I/O transaction without disturbing prior addressing. The I/O task is performed without any command information (asserting ATN). This technique is useful for some buffered devices when a portion of the buffer is sent each time without re-addressing. Also, this technique can speed processing when the talkers and listeners do not need to be changed.

The direct addressing technique is a faster and more efficient method than auto-addressing, but places the burden of bus control on the user program. Under direct addressing, the user program provides two buffers to the HP-IB driver. One buffer contains the command information which will be sent over the bus while ATN is asserted. This is used to assign the appropriate talkers and listeners, or to send universal and secondary command characters. The other buffer either contains the data to be sent out, or is the place where incoming data will be stored. Remember to specify the computer's talk or listen address with direct addressing if the computer will participate in the data transfer.

There are two ways to directly address devices over the bus. HP-IB library routines CMDR and CMDW (command read and write) or EXEC calls can be used in FORTRAN. In BASIC, CMDR and CMDW must be used, since EXEC calls cannot be used.

The HP-IB Users Guide (HP part number 59310-90064) explains how to use CMDR and CMDW, but two vital points are often overlooked. First, CMDR and CMDW were intended for use with BASIC, and expect the command and data buffers to contain character strings. In FORTRAN, the format of these buffers must resemble a string. This means that the first word of the buffer must contain the number of characters in the string. After this first word, each additional word will contain two characters. The second point to remember is that direct addressing requires that the LU of the bus be used. The HP-IB driver checks to see if the subchannel specified is 0. If it isn't, an error occurs, the task is rejected, and the program is terminated.

The library commands CMDW and CMDR reformat the buffers and make calls to EXEC. If the language used allows calls to EXEC, they can be used directly. The format for the direct addressing EXEC call is:

```
        CALL EXEC(ICODE,ICNWD,IDBFR,IDLEN,ICBFR,ICLEN)
```

where:

ICODE = Function Code

- 1 = READ
- 2 = WRITE

ICNWD = Control word containing the bus LU and an indicator for type of transfer

10000B+LU	= ASCII data record with an end of record indicator.
10100B+LU	= Fixed length binary record with an end of record indicator.
12000B+LU	= ASCII data record without an end of record indicator.
12100B+LU	= Fixed length binary record without an end of record indicator.

IDBFR = Name of data buffer

IDLEN = Length of data buffer in either characters (bytes) or words.

- n = words
- −m = characters
- 0 = no data

ICBFR = Name of buffer containing the HP-IB commands

ICLEN = Length of command buffer

- n = words
- −m = characters
- 0 = no command buffer used

Note in the above EXEC call that the command and data buffers are optional parameters. If the EXEC call is to perform HP-IB commands only, no data buffer is used. The EXEC call expects these parameters to be specified, so remember to supply arguments for both the buffer name and length (set the values to 0's). If the talk and listen assignments are not altered, the command buffer is not used. It is either set to 0's, or can be omitted. If it is omitted, be sure to turn off bit twelve in the above control words (subtract 10000B from each of the commands).

Special bus related commands can also be sent to the HP-IB bus to clear devices, to trigger them, and to regulate polling for service requests. These commands, which are referred to as "universal" commands, are shown in table 2. They can be sent in command buffers under direct addressing, or with library routines. The HP-IB Users Guide explains the library calls in detail, and gives examples of their use.

The preceeding discussion covered addressing and talking to HP-IB devices. Extra processing power and efficiency can be obtained by utilizing a full understanding of the addressing scheme. Remember, it is important to know your address when getting on the bus.

Table 2. HP-IB Universal Commands

COMMAND	ACRONYM	OCTAL CODE	FUNCTION
Local Lockout	LLO	21	Causes all responding devices to disable their front panel local-reset buttons. Devices need not be addressed.
Device Clear	DCL	24	Causes all responding devices to return to a pre-determined state. Devices need not be addressed.
Selected Device Clear	SDC	04	Causes the current addressed devices to re-set to a predetermined state.
Group Execute Trigger	GET	10	Causes all current addressed devices to initiate a preprogrammed action.
Go To Local	GTL	01	Causes the currently addressed devices to return to local control.
Serial Poll Enable	SPE	30	Establishes serial poll mode, such that all co-operating devices, when addressed, will pro-vide status information. An ensuing read will take a single 8-bit byte of status.
Parallel Poll Configuration	PPC	05	Assigns an HP-IB DIO line to a cooperating device for the purpose of responding to a parallel poll.
Parallel Poll Unconfiguration	PPU	25	Resets all parallel poll devices to a prede-termined condition.
Take Control	TCT	11	This command is used to pass active control-ler function to another device on the bus. This function may NOT be used under the constructs of the current driver.
The remaining unspecified codes are reserved for future use and should not be used indiscriminately in any control buffer. This will avoid future difficulties.			

THE MOST POWERFUL RTE EVER

Jim Williams/DSD Product Marketing

On November 1, 1981, Data Systems Division made available for order RTE-6/VM, a major enhancement to the high end Real-Time Executing Operating System on the HP 1000. This new operating system contains many enhancements to existing features of RTE-IVB, and also includes three major new capabilities, which make the HP 1000 the most powerful 16-bit minicomputer available by far. These capabilities are:

1. The ability to handle very large data areas through a Virtual Memory scheme;
2. Shared data areas of up to 1.9 megabytes in size;
3. The ability to handle large code with minimal or no changes to the source program, and to have up to 1.9 megabytes of code in main memory.

The Virtual Memory for data system allows the user to access up to 128 megabytes of data by simply dimensioning the arrays normally and adding one control statement to the source program, the $EMA statement. When the user loads the program, he or she may choose between utilizing the VM system or the EMA memory resident subset of VM by issuing one command to the loader. SIMPLE! After the program is loaded, access to the data is totally transparent to the user, whether the data is in memory or on the disc. From the user's point of view, the data is accessed as if it were local data.

Typical applications requiring access to large data areas are CAD simulation, graphics, linear programming, and statistical and structural analysis. Applications dealing with large matrices, such as statistical and structural analysis, may also take advantage of the Vector Instruction Set, which works in conjunction with the Virtual Memory System to give a powerful large matrix manipulation tool. No other 16-bit computer has this extensive data handling capability.

The second new major feature refers to the sharing of large areas of data in memory among up to 64 programs. Upon generation or system boot-up, up to 8 memory partitions of any legal size may be specified as shareable. The operating system will manage the use of these partitions so that, if the user desires, a partition may be locked and remain in use for sharing data even when no active program is using the data. Also, when the partition is not being used to share data, it is used by the system just as any other partition of its type.

This capability is extremely popular in the process monitor and control environment. The existence of a large process table in main memory containing data that describes the process, with several programs accessing the data to monitor and analyze the process, while other programs are updating real-time displays describing the process, is essential to many applications. The HP 1000 now has the most powerful shared data scheme implemented on 16-bit minicomputers.

The last of the three major features of RTE-6/VM listed above is the capability to efficiently handle large programs. We call this capability Extended Code Space (ECS). There are two distinct advantages to this scheme over other implementations of solutions to the large code problem. The first advantage is the implementation of code segmentation with minimal or no changes to the source of the program. This allows the programmer to develop complex programs in modular form without excessive concern for the word size of the target machine. This capability also allows the user to convert programs written for larger, more expensive computers to the HP 1000, and realize a tremendous price/performance advantage. The normal problem when transporting a large program to a different machine is facing a programming language conversion prior to facing a system conversion. This problem is averted by the coordinated introduction of our new, full ANSI standard FORTRAN 77 compiler. This compiler, in combination with ECS make large program conversion to the HP 1000 an advantageous proposition.

The second advantage realized by the implementation of ECS on the HP 1000 is the extremely low overhead incurred by the operating system when switching control between code segments that are memory resident. This transition between segments, implemented through normal user subroutine calls, is accomplished by the Operating System and micro coded for speed and efficiency. Up to 1.9 megabytes of program code may take advantage of this speed by residing in memory, with the remainder of non-critical code residing on disc.

This capability will allow the HP 1000 to be utilized in applications previously restricted to larger, more expensive computers.

There are no less than 22 additional features in RTE-6/VM. I will not attempt to explain each one, but instead categorize them into four classes, and give examples of features in each category that have been particularly well received.

1. Input/Output Extensions
   Class I/O Clean-Up
   I/O Request Cancellation
   More peripherals capability (255 EQTs)

2. Extensions to RTE-IVB Resources
   More System Available Memory (SAM)
   Larger system discs

3. Friendlier on-line quick reference utilities
   Faster HELP
   User Extendable Help files

4. Performance
   With all these new features and functionality, you might think that the performance of the operating system would surely be degraded — NOT SO! In fact, in most cases the new system outperforms RTE-IVB! The reason is the addition of Operating System Accelerator Firmware; the microcoding of key areas of RTE-6/VM.

From this discussion of the BIG THREE and the many other features, you can see that this is truly a major revision to our high-end operating system family.

ORDERING INFORMATION

92084A:	$9,000 plus media for first time RTE buyers
Option 001:	−$3,600 upgrade discount from RTE-IVB, no services
Option 002:	−$6,300 upgrade discount from RTE-IVB if on CSS or SSS.

For out customers with more than 15 systems on support services, a special upgrade path has been defined.

92085A:	$6,000 RTE-6/VM Bulk Upgrade License (software only)
92086A:	$ 650 RTE-6/VM Upgrade Kit (firmware and manuals)

A customer wishing to upgrade more than 15 systems would purchase one 92085A product and a number of 92086A products equal to the number of CPUs to be upgraded.

Remember, RTE-6/VM is the most powerful Real-Time Operating System for automation, test and control applications.

NEW LANGUAGES EXTEND PROGRAMMING CAPABILITIES ON RTE-6/VM

Linda Haar/DSD Sales Development

To fully complement the introduction of RTE-6/VM, DSD has also announced a variety of new languages. As a result, a larger array of compilers is offered on RTE-6/VM than any previous HP 1000 operating system.

Leading the way on the RTE-6/VM offering are the new compilers: FORTRAN 77, an enhanced Pascal compiler, and the new Macro Assembler.

FORTRAN 77

The most significant new product offered with RTE-6/VM is FORTRAN 77.

First of all, FORTRAN 77 is a *complete* ANSI 77 standard compiler. Nonetheless, it also provides even more flexibility and compatibility by incorporating the complete MIL-Std-1753 implementation as well as major mainframe extensions. These are all powerful aids for moving programs over to the HP 1000. In addition, FORTRAN 77 is a superset of the ANSI 66 FORTRAN 4X compiler. In cases where incompatibilities exist between the ANSI 66 and ANSI 77 standards, FORTRAN 77 provides a software switch so the user may decide to which standard he will adhere. So, current FORTRAN 4X programs are totally *compatible!*

And, even with all these features, FORTRAN 77 running on RTE-6/VM provides no additional overhead above that found on FORTRAN 4X with RTE-IVB.

Thus, FORTRAN 77 provides a set of programming capabilities over and above that of many ANSI 77 compilers. Together, RTE-6/VM and FORTRAN 77 provide a winning combination for program development.

PASCAL

RTE-6/VM also supports an enhanced higher performance Pascal compiler. The virtual memory capabilities of RTE-6/VM made it possible to dramatically improve a common complaint about Pascal; slow compilation. By using VMA/EMA, the new Pascal for RTE-6/VM achieves twice the compilation speed of Pascal on RTE-IVB. Higher compile speeds translate to increased throughput for the single user or for several programmers doing compilation on a single system.

MACRO/1000

RTE-6/VM and RTE-XL will both include the new Macro Assembler as a standard part of the operating system. A fully-compatible, greatly-enhanced superset of the RTE-IVB assembler, Macro/1000 exhibits twice the compilation performance of its predecessor. The enhanced feature set includes file and string manipulation routines, more pseudo-operations, and more extensive assembly-time conditionals. However, perhaps the greatest enhancement is the full support of macros and macro libraries. These provide powerful programming capabilities as macro libraries can be built and searched at compilation time in much the same way as subroutine libraries are built today. This provides a means to reduce redundant code and implement high level constructs into assembly level programs.

FORTRAN 77, the new Pascal and the new Macro Assembler all support the new record format in RTE-6/VM. This means that externally referenced names can now be sixteen characters in length removing the restriction of five characters found in our old compilers.

To round out our new language offerings, we are also introducing Pascal on RTE-XL, providing a powerful structured programming language on our L-Series microsystems.

We feel the new compilers will strongly complement RTE-6/VM in providing the strongest programming development capabilities ever found on a 16-bit minicomputer. But those languages do not complete the list. BASIC-D and FORTRAN 4X (effective revision code 2140 or later) will be supported under RTE-6/VM; and a third party, Corporate Computer Systems (CCS), offers the C Compiler and a just released COBOL compiler for the new RTE-6/VM systems. An array of languages never found before on an HP 1000 computer system; all designed to fully meet with any programmer's needs.

ORDERING INFORMATION

		Price
92836A	FORTRAN 77 for M,E,F computers operating under RTE-6/VM (must order a media option)	$4,500
-001	Upgrade discount (from previous version of 92836A for users not on CSS/SSS)	−1,800
-003	Upgrade discount (from 92834A FORTRAN 4X for users on CSS/SSS as of November 1, 1981)	−3,000
-004	Upgrade discount (from 92834A FORTRAN 4X for users not on CSS/SSS)	−1,000
92833A	Pascal/1000 for M/E/F-Series computers with 384k bytes memory operating under RTE-6/VM. (Must order media option)	$4,500
-001	Upgrade discount from 92832A a previous version of 92833A for user not on CSS/SSS	−1,800
-002	Upgrade discount from 92832A for user on CSS/SSS	−4,500
92854A	Pascal/1000 for L-Series computer with at least 128k byte of memory and hard disc operation under RTE-XL (Must order media option)	$1,500
-001	Upgrade discount from previous version of 92834A for user not on CSS/SSS	−600

JOIN AN HP 1000 USER GROUP!

Here are the groups that we know of as of December 1980. (If your group is missing, send the Communicator/1000 editor all of the appropriate information, and we'll update our list.)

NORTH AMERICAN HP 1000 USER GROUPS

Area	User Group Contact
Arizona	Jim Drehs 7120 E. Cholla Scottsdale, Arizona 85254
Boston	LEXUS P.O. Box 1000 Norwood, Mass. 02062
Chicago	David Olson Computer Systems Consultant 1846 W. Eddy St. Chicago, Illinois 60657 (312) 525-0519
Greenville/S. C.	Henry Lucius III American Hoechst Corp. P.O. Box 1400 Greer, South Carolina 29651 (803) 877-8471
Huntsville/Ala.	John Heamen ED35 George C. Marshall Space Flight Ctr. Nasa Marshall Space Flight Ctr., AL. 35812
Montreal	Erich M. Sisa Siemens Electric Ltd. 7300 Trans Canada Highway Pointe Claire, Quebec H9R 1C7
New Mexico/El Paso	Guy Gallaway Dynalectron Corporation Radar Backscatter Division P.O. Drawer O Holloman AFB, NM 88330
New York/New Jersey	Paul Miller Corp. Computer Systems 675 Line Road Aberdeen, N.J. 07746 (201) 583-4422

NORTH AMERICAN HP 1000 USER GROUPS (CONTINUED)

Area	User Group Contact
Philadelphia	Dr. Barry Perlman RCA Laboratories P.O. Box 432 Princeton, N.J. 08540
Pittsburgh	Eric Belmont Alliance Research Ctr. 1562 Beeson St. Alliance, Ohio 44601 (216) 821-9110 X417
San Diego	Jim Metts Hewlett-Packard Co. P.O. Box 23333 San Diego, CA 92123
Toronto	Nancy Swartz Grant Hallman Associates 43 Eglinton Av. East Suite 902 Toronto M4P1A2
Washington/Baltimore	Mal Wiseman Hewlett-Packard Co. 2 Choke Cherry Rd. Rockville, MD. 20850
General Electric Co. (GE employees only)	Stu Troop Special Purpose Computer Ctr. General Electric Co. 1285 Boston Ave. Bridgeport, Conn. 06602

OVERSEAS HP 1000 USER GROUPS

Belgium	J. Tiberghien Vrije Universiteit Brussel Afdeling Informatie Pleinlaan 2 1050 Brussel Belgium Tel. (02) 6485540

OVERSEAS HP 1000 USERS GROUPS (CONTINUED)

Area	User Group Contact
France	Jean-Louis Rigot Technocatome TA/DE/SET Cadarache BP.1 13115 Saint Paul les Durance France Tel. (042) 253952
Germany	Hermann Keil Vorwerk + Co Elektrowerke Abt. TQPS Rauental 38-40 D-5600 Wuppertal 2 W. Germany Tel. (0202) 603044
Netherlands	Albert R. Th. van Putten National Institute of Public Health Antonie van Leeuwenhoeklaan 9 Postbox 1 3720 BA Bilthoven The Netherlands Tel. (030) 742344
Singapore	W. S. Wong Varta Private Ltd. P.O. Box 55 Chai Chee Post Office Singapore Tel. 412633
Switzerland	Graham Lang Laboratories RCA Ltd. Badenerstrasse 569 8048 Zurich Switzerland Tel. (01) 526350
United Kingdom	Mike Bennett Riva Turnkey Computer Systems Caroline House 125 Bradshawgate Bolton Lancashire United Kingdom Tel. (0204) 384112

Printed in U.S.A. 12/81

Part No. 5951–6111
E1281