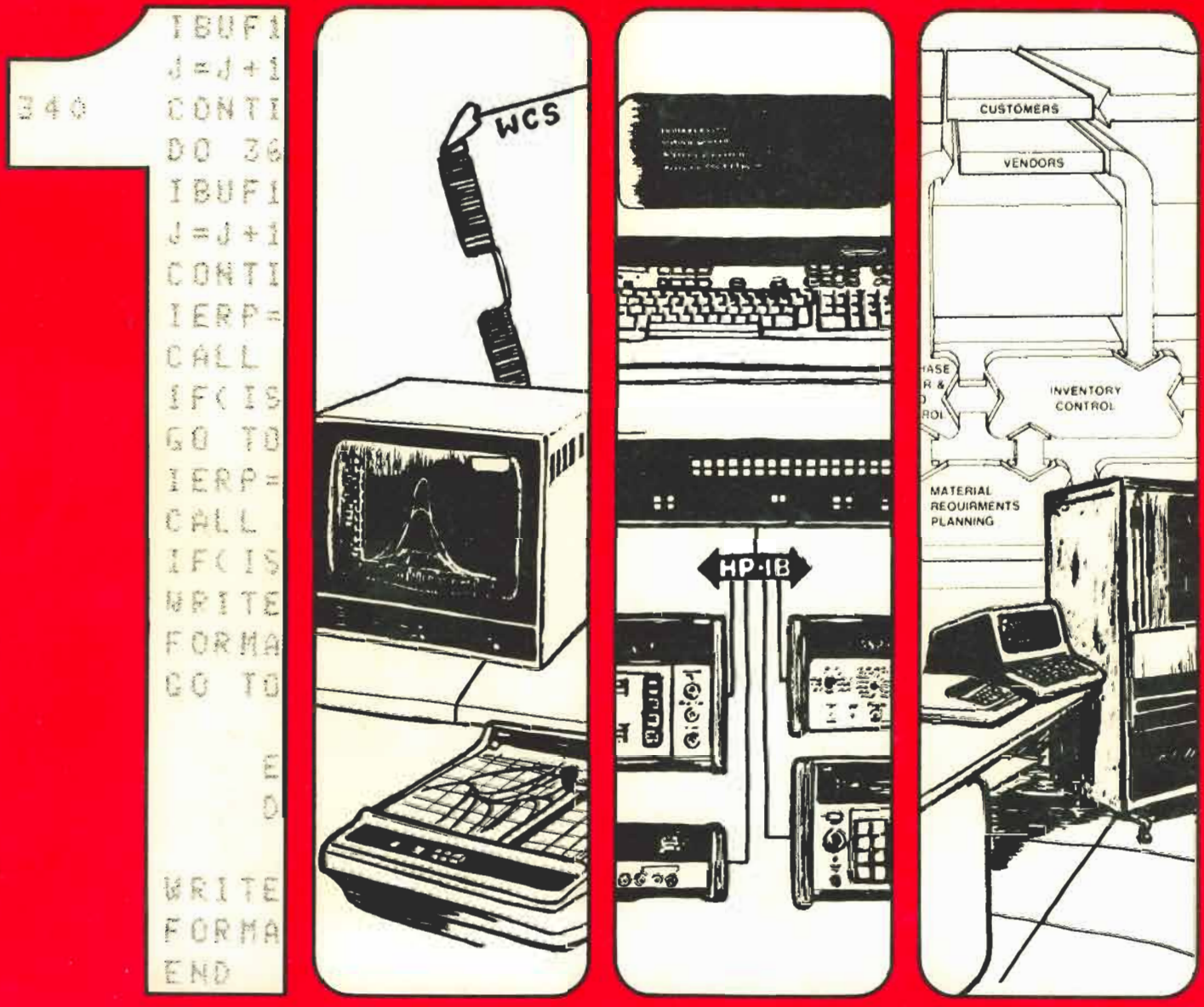
HEWLETT PACKARD

Hewlett-Packard Computer Systems

COMMUNICATOR



1983

Volume V Issue 6

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HEWLETT-PACKARD
COMPUTER SYSTEMS

Volume V
Issue 6

COMMUNICATOR/1000

Computer Museum

Feature Articles

Departments

EDITOR'S DESK

YES, VIRGINIA, THERE IS A COMMUNICATOR/1000

Reports of the Communicator/1000's demise have been greatly exaggerated! Late — Maybe! Dead — No!

Hewlett-Packard in general, and Data Systems Division specifically, experienced a very active year of rapid growth and the introduction of an impressive number of new products. Unfortunately, the Communicator/ 1000 does not have a full-time editor, and the division's acceleration was at the expense of time available to work on the magazine.

We indeed recognize the importance of the Communicator/1000 as a tool for those of you who rely upon HP 1000 Computer Systems in your daily operations. With this thought in mind, we are placing increased emphasis on getting the magazine back on schedule.

After studying the monthly workload and the volume of submissions received, it has been decided that the number of issues per year should go from six to four. So, we are setting a goal of going to press quarterly, counting from the publication date of this current issue. That means you can expect an issue in early May, August, November 1983 and February, 1984. The tentative deadlines for Volume VI submissions work out to be:

Issue 1 15 March 1983
Issue 2 13 June 1983
Issue 3 14 September 1983
Issue 4 12 December 1983

In this issue, we have a variety of contributions from around the world on a myriad of topics, but with a slant toward industrial automation. Data processing manager Michael Miller explains how HP 1000 Computer Systems have been successfully applied to industrial automation at the Bendix Corporation, in the Operations Management section. In the Languages section, Diana Bare of HP's Roseville Division discusses how to cope with language variations when downloading subroutines to an HP 2250 Measurement and Control Processor. The process monitor and control theme continues as Hewlett-Packard announces Process Monitoring and Control/1000 applications software, in the Bulletin section.

Since most of us use Edit/1000 extensively for text processing or file manipulation, don't miss Michael Wiesenberg's article, "Tricks with Edit/1000", in the Operations Management section. His tricks are so practical that it's difficult to call them fun — but they are.

This month's feature in the Operating Systems section is "Simplified Development of Custom I/O Drivers", by John Trueblood, a senior systems analyst with the Sony Corporation of America. John did it the hard way by discussing driver writing to handle communication protocol, I/O transfer, and peripheral control for the entire spectrum of active RTE operating systems! And for lighter reading, the Bit Bucket is filled with goodies.

With this being the last issue of Volume V, we have also included the traditional index of articles for the entire set of six issues.

And now for this month's winners of an HP32E calculator for best feature article. With no articles from HP Field Employees, we only have two winners.

Best feature article by an HP Customer:	**"Simplified Development of Custom I/O Drivers"** John Trueblood, Senior Systems Analyst Sony Corporation of America, San Diego
Best feature article by an HP Division employee:	**"2250 Buffer Management from Downloaded Subroutines"** Diana Bare HP Roseville Division, California

Needless to say, we encourage you to continue to send in contributions and share those great ideas you've discovered. The selection of articles always seems biased by the convenience of transferring it to a disc file. On the following page, we explain the desire for a minicartridge or other machine readable medium. The ideal format is a basic text file (without RUNIT commands, security codes, etc.) that can be loaded with a :STore command. If your format differs, please explain how to read and store it. Also, files that are in all-capital letters have to be converted (then we have to guess the capitalization of your special terminology, such as acronyms).

Another way to enhance the acceptance of your contribution is to establish a reasonable balance between the volume of your explanatory text and the program code. The theory is that readers are unwilling to type in a long program without a good appreciation of "what, why and how." (We'll concede that generous comment statements within the program do help.)

The programs that are published have been carefully reviewed by the authors themselves and responsible people here at Data Systems Division. But we cannot guarantee that they will work on your system. We consider the risk of your damaging your system to be at an absolute minimum, but remember that the listings will have gone through multiple iterations by the time they appear in print.

Enough said. Keep those cards and letters coming in!

Editor

EDITOR'S DESK

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 Editor:

I am very pleased with the responses and alternative methods to the one I proposed for dynamic use of memory behind your FORTRAN program. I would like to point out for those using FTN7X (the FORTRAN 77 Compiler) that none of the previously mentioned methods is required.

FTN7X has a simple method described on page 7-17 of the reference manual. By defining an array that begins at the beginning of absolute memory (location 0), one can use LIMEM to find the first word after your program which would be ARRAY (FWAM).

Hats off to the compiler writer for adding this feature!

Best regards,

John A. Pezzano

Dear Mr. Pezzano,

Thanks for the additional input.

Sincerely

The Editor

CORRECTIONS TO PREVIOUS ISSUES

In a recent Bit Bucket article (Volume V, Issue 4), Elaine Mosakowski and Terry O'Neal presented a FORTRAN program for automatic scaling and logarithmic plotting for Graphics/1000-II. Unfortunately, two subroutines were omitted. MGETC extracts a character from a string, and MPUTC puts a single character in an integer array. These subroutines are listed below.


```
FTN4X,L
      INTEGER FUNCTION MGETC (PLACE, STRING)
     *, GET A CHAR FROM A STRING

      IMPLICIT NONE
      INTEGER  PLACE, STRING
      DIMENSION STRING(1)

C*************************************************************************
C  PURPOSE:
C
C        THIS ROUTINE EXTRACTS A CHARACTER FROM A STRING
C
C  PARAMETERS:
C
C        PLACE - ORIGIN 0 OFFSET INTO STRING
C
C        STRING- ARRAY THAT FROM WHICH CHARACTER IS EXTRACTED
C
C*************************************************************************
C
C *****  LOCAL VARIABLES:
C
C        WORD   -- HOLD THE WORD THAT CHAR RESIDES IN
C
C*************************************************************************

      INTEGER WORD

C
C *** GET WORD OUT OF STRING THAT CONTAINS CHARACTER
C *** ASSUMPTION: 2 CHARACTERS PER WORD
C
      WORD = STRING( (PLACE+2)/2 )
C
C ***  NOW EXTRACT THE CHAR FROM THE WORD.  WHAT CHARACTER POSITION IS IT?
C
      IF ( MOD(PLACE,2) .EQ. 0 ) GO TO 10
C
C *** CHAR IS IN LOW BYTE
C
         MGETC= IOR(ISHFT( WORD, 8),40B)
         GO TO 9999
C
C
   10 CONTINUE
```

```
C
C *** CHAR IS IN HIGH BYTE
C
         MGETC= IOR( IAND ( WORD, 177400B), 40B)
C
C
9999  RETURN

      END



FTN4X,L
C
      SUBROUTINE MPUTC (CHAR, PLACE, STRING)
     *, PUT A SINGLE CHARACTER IN AN INTEGER ARRAY
C
      IMPLICIT NONE
C
      INTEGER CHAR, PLACE, STRING
      DIMENSION STRING(1)

C*****************************************************************************
C
C  PARAMETERS:
C
C     CHAR   --  THE CHARACTER TO PUT INTO STRING IN GRAPHICS 1000-II
C                PACKED ASCII FORMAT
C
C     PLACE  --  THE ORIGIN 0 OFFSET INTO STRING THAT INDICATES WHERE TO
C                PLACE THE CHARACTER
C
C     STRING --  THE ARRAY TO PLACE A CHARACTER IN
C
C  PURPOSE:
C
C     THIS ROUTINE PUTS CHARACTER "CHAR" INTO "STRING" AT OFFSET "PLACE".
C
C
C*****************************************************************************
C*****************************************************************************
C
C
C       LOCAL VARIABLES:
C
C       WORD   -- THE WHOLE WORD OF THE ARRAY THAT "CHAR" IS LOCATED IN
C
C       WPLACE -- THE INDEX OF THE WORD THAT THE CHARACTER IS CONTAINED IN
C
C*****************************************************************************
C
      INTEGER WORD, WPLACE
C
C *** FIND WHAT WORD THE CHAR IS IN
```

```
C
      WPLACE= ((PLACE +2) /2)
C
C *** GET WORD FROM ARRAY
C
      WORD = STRING( WPLACE )
C
C *** FIND THE BYTE THE CHAR IS IN
C
      IF( MOD ( PLACE, 2 ).EQ. 0) GO TO 10
C
C *** PUT THE CHAR IN THE LOW BYTE
C
          STRING( WPLACE ) = IOR(
     1                          IAND ( WORD,177400B),
     2                          ISHFT( CHAR, -8))
          GO TO 999
C
   10 CONTINUE
C
C *** PUT THE CHAR IN THE HIGH BYTE
C
      STRING( WPLACE ) = IOR(
     1                     IAND ( WORD, 377B ),
     2                     IAND ( CHAR, 177400B))
 999  CONTINUE
      RETURN
C
      END
```

FOR/NEXT LOOPS IN FORTRAN

by Alf Lacis, c/o Chatads P/L.,
P.O. Box 846, Traralgon, Vic., 3844
Australia

Here is a little library which takes the hassle out of devising explicit incremental instructions for loops. The subroutines shown can be adapted equally well for type "real" or type "double precision" (3 or 4 word).

The FOR/NEXT loops work with positive or negative increments, and like DO loops, will execute at least once. On completion, the loop variable value is always "outside" the loop limits.

A pair of FOR/NEXT subroutines may be used many times throughout the programme, but a difficulty arises if they need to be nested. The nesting difficulty is overcome by having pairs of routines called "FOR1/NEXT1", "FOR2/NEXT2", "FOR3/NEXT3", etc, or as many as are required. I have found that three sets suffice for even the most amazing programmes.

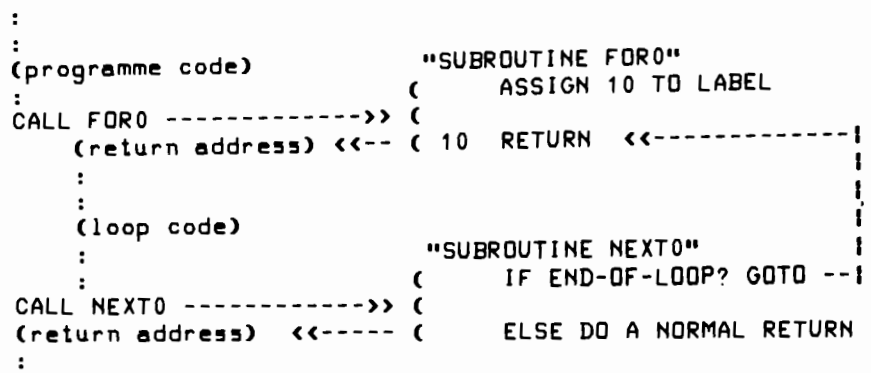
EXAMPLE 1: A single pair of routines used in different parts of a programme.

```
FROM = 1.
TO   = 100.
STEP = 2.
SUM  = 0.
CALL FOR0 ( VALUE, FROM, TO, STEP )
    :
    (calculations)
    :
CALL NEXT0 ( VALUE, FROM, TO, STEP )
:
B = 25.
C = -25.
D = -5.
CALL FOR0 ( A, B, C, D )
    :
    (calculations)
    :
CALL NEXT0 ( A, B, C, D )
```

EXAMPLE 2: Nested loops, using FOR0/NEXT0 and FOR1/NEXT1.

```
CALL FOR0 ( A, PI, -PI, -PI/10. )
    :
    :
    FROM = 0.
    TO   = 100.
    STEP = 2.
    CALL FOR1 ( BIT, FROM, TO, STEP )
        :
        :
    CALL NEXT1 ( BIT, FROM, TO, STEP )
    :
    :
CALL NEXT0 ( A, PI, -PI, -PI/10. )
```

Notice that no line numbers are necessary! Here is a brief schematic of how they work. Notice that the CALL to FOR0 is actually executed only once, while a normal exit to NEXT0 again only occurs once.



These routines were originally developed as part of a sub-division plotting routine, where we needed to draw segments of arcs incrementally at any orientation and in both clockwise and anticlockwise directions. They reduced the amount of IF'ing and ELSE'ing to nill.

Here is the code:

```
FTN4
C-------------------------------------------------------------
      SUBROUTINE FOR1( A, B, C, D ),16-10-81 S: For...
      COMMON /FRNX1/ LABEL, E
      A = B
      E = C*D
      ASSIGN 10 TO LABEL
10    RETURN
      END
C-------------------------------------------------------------
      SUBROUTINE NEXT1( A, B, C, D ),16-10-81 S: ...Next
      COMMON /FRNX1/ LABEL, E
      A = A + D
      IF ( A*D .LE. E ) GOTO LABEL
      RETURN
      END
C-------------------------------------------------------------
      SUBROUTINE FOR2( A, B, C, D ),16-10-81 S: For...
      COMMON /FRNX2/ LABEL, E
      A = B
      E = C*D
      ASSIGN 10 TO LABEL
10    RETURN
      END
C-------------------------------------------------------------
      SUBROUTINE NEXT2( A, B, C, D ),16-10-81 S: ...Next
      COMMON /FRNX2/ LABEL, E
      A = A + D
      IF ( A*D .LE. E ) GOTO LABEL
      RETURN
      END
```

```
C-------------------------------------------------------------
      BLOCK DATA FN,16-10-81 B: Return address
      COMMON /FRNX1/ LAB1, E1
      COMMON /FRNX2/ LAB2, E2
      END
```

The NEXT call could be simplified by leaving out all the parameters and passing the parameters through COMMON, eg:

```
      FTN4
C-------------------------------------------------------------
      SUBROUTINE FOR1( A, B, C, D ),16-10-81 S: For...
      COMMON /FRNX1/ LABEL, E1, B1, C1, D1
      A = B
      B1 = B
      C1 = C
      D1 = D
      E1 = C*D
      ASSIGN 10 TO LABEL
10    RETURN
      END
C-------------------------------------------------------------
      SUBROUTINE NEXT1( A, B, C, D ),16-10-81 S: ...Next
      COMMON /FRNX1/ LABEL, E1, B1, C1, D1
      A = A + D1
      IF ( A*D1 .LE. E1 ) GOTO LABEL
      RETURN
      END
```

RESTORING PURGED FILES

by Paul M. Dunphy
Nova Scotia Department of Health

Have you ever hit the return key after purging an obsolete file, then had the sudden realization that the file was not as obsolete as you had thought, and then have a File Manager Directory Listing confirm that the purge command was indeed doing its job? I have had this happen several times, even when the files were protected with security codes. If the file is a user-generated relocatable program (type 5) or an absolute program (type 6), this is usually not a problem, as they can be recompiled, reloaded, etc. However, other file types leave the operator with several less desirable options:

1. There is a copy of the file on a cartridge or magnetic tape unit. This can be restored to the cartridge rather easily.

2. It is a type 4 file (source program). There may be a recent listing that can be retyped in.

3. It is a data file (e.g. type 1 or 2). The file may be recreated and the data recollected.

4. It is a data file, and the data is unique and cannot be replaced.

5. It is a program, and there is no recent listing.

None of these options is particularly appealing, especially the last two. To solve this problem, we have developed the FORTRAN program GETBK, which can restore purged files. Each cartridge has a file directory located on one or more of its tracks. This directory consists of a 16-word entry for each of the files on that cartridge. The first three words in each entry contain the ASCII name of the file. The end of the directory is marked by a binary zero in the first word following the last active file entry.

When File Manager purges a file, the first word of the corresponding entry in the file directory is replaced by minus one (−1). The rest of the directory and the file itself are unchanged. This action leaves the file unprotected and inaccessible by FMP. If a write request occurs for a file of equal or smaller size, the purged file will be written over. Likewise, if the cartridge is packed, files below the purged one will be moved up and it will also be written over. In either case, the directory is modified to conform to the new file. If this happens, the purged file is permanently lost, and there is no way to recover it.

However, if neither of these events has occurred, it is relatively easy to restore the file. Changing the first word of the corresponding file directory entry from minus one back to its original state is all that is necessary. In fact, this word may be changed to any two ASCII characters that File Manager will recognize as valid, thus giving the restored file a new name. However, I would recommend that you not use this method to rename files!

Program GETBK scans the directory tracks of a specified cartridge, looking for purged files. If it is not known which cartridge contains the file in question, all cartridges can be checked. When a purged file is encountered, the remaining portion of the file directory entry is displayed on the operator's terminal. The first two file name characters can then be specified. After checking the cartridge to ensure that a duplicate file name is not being created, and that no illegal characters have been used, GETBK will restore the file. If this is not the file sought, it can be bypassed and GETBX will proceed to the next purged file. This process is repeated until a file is restored or an end of directory is encountered.

END OF DIRECTORY

As mentioned above, the end of directory mark is a binary zero in the next word following the last active file entry. If the purged file was the last file on the cartridge, File Manager realizes this and places a zero where the minus one purge indicator would normally go. GETBK checks the format of the next 15 words, and if they conform to file directory rules, this entry is considered a purged file. If it is restored, the file directory is returned to its original size automatically, as an end of directory mark must have existed beyond the restored file, prior to its being purged. This process also takes place when restoring extents.

FILE EXTENSIONS (EXTENTS)

A special situation can exist with file types greater than 2. When a write request points to a location beyond the range of the currently defined file, a file extension or extent is created by FMP. This extent has the same name and size as the original file. FMP repeats this process as often as necessary as file size increases. Each extent is given an extent number that is stored in the left byte of word 5 in the the corresponding entry of the file directory. There may be up to 255 extents for each file. It is a good practice to keep the number of extents to a minimum. Alan K. Housley and Clark Johnson/HP Data Systems Division, describe an excellent program for doing so in a previous Communicator, Volume IV, Issue 4.

In any event, file extensions do exist and must be taken into consideration when restoring files. When a File Manager PU, (namr) command is executed, the cartridge directory is searched for all entries with the specified name. Thus, any extents that may exist are purged, along with the original file.

Conversely, when GETBK restores a file, it must also ensure that all extents are restored. Beginning at the original file, GETBK searches down the directory tracks for additional purged entries. Since extents have directory entries that are the same as the original file, except for words 4 and 5, they are readily identified and restored with the same name as the original. This process continues until up to 255 extents are restored or no further entries exist. In the event that some of the extents were lost by being written over, GETBK will still restore any that do exist. This will allow the operator to at least recover as much of his file as possible.

TYPE 6 FILES

When a program is SP'ed by File Manager, it is assigned a setup code word that is unique to each system generation. This word is determined by summing the contents of words 1650 through 1657, and words 1742 through 1747, and words 1755 through 1764 in the base page. It is then stored in word 34 of the first sector of the file. File Manager verifies this word when a program is RP'ed. This is designed to prevent incompatible programs from being run on systems other than that on which they were loaded. When such a program is purged, this number is also removed from the file. Thus, if type 6 files were to be restored by GETBK, they would give a File Manager 19 error (Program not set up by SP on current system).

It would be possible to determine this code word by summing the contents of these locations and then restore it to the file. However, this could lead to trouble. Consider the following sequence of events. A type 6 file has been purged, and this action goes unnoticed for several days. Meanwhile, the system manager generates a new system. The purged file is discovered and restored, with a new setup code word that has been determined from the new system base page. The program can now be RP'ed because File Manager does not find an incorrect code word. Since the program is not compatible with the new system, numerous system errors can occur. Therefore, GETBK does not attempt to recover type 6 files, as they are user-loaded programs, and can easily be recovered by reloading.

CONCLUSION

Accidently purging a file can be quite common where disc space is limited, as with Session Monitor. In such cases, one does not have the luxury of keeping unused files indefinitely, and must constantly be purging them as they become obsolete. GETBK will be particularly useful to operators in this situation. You will find that the sooner you try to recover your file after it is purged, the greater your success will be. Every time another file is written on the cartridge, the danger exists that your program will be lost, particularly with files at the end of the directory. I have found it good practice to scan the cartridge with GETBK before packing it to ensure that there are no accidently purged files that have gone unnoticed.


```
FTN4
      PROGRAM GETBK(3,99),PURGED FILE RECOVERY PROGRAM <821208.0842>
C
C*********************************************************************
C                                                                     *
C     PROGRAM NAME: GETBK (GET PURGED FILE BACK)                      *
C                                                                     *
C     DATE: JAN. 5, 1982                                              *
C                                                                     *
C     AUTHORS: PAUL DUNPHY AND JAMES MATHERS                          *
C              VICTORIA GENERAL HOSPITAL                              *
C              HALIFAX, NOVA SCOTIA                                   *
C                                                                     *
C     DISCRIPTION: PROGRAM TO RECOVER ACCIDENTIALLY PURGED            *
C                  DISC FILES. DIRECTORY TRACKS OF SPECIFIED          *
C                  CARTRIDGE OR CARTRIDGES ARE SCANNED FOR PURGED     *
C                  FILES. AT OPERATOR'S DISCRETION, DIRECTORY IS      *
C                  MODIFIED TO RESTORE FILE AND ANY EXTENTS THAT      *
C                  MAY EXIST.                                         *
C                                                                     *
C*********************************************************************
C
      IMPLICIT INTEGER(A-Z)
      INTEGER STATUS(125),BUFFER(4),DIRECT(16)
      INTEGER CARTDR(136),CTEMP(136),BUFF(3),EXTENT(16)
      EQUIVALENCE (CARTDR(129),ENTRY),(CARTDR(130),DTRACK)
      EQUIVALENCE (CARTDR(132),I),(CARTDR(133),SECTOR)
      EQUIVALENCE (CARTDR(134),NUMBER),(CARTDR(135),COUNT)
      EQUIVALENCE (CARTDR(136),KOUNT),(CARTDR(131),BLOCK)
      DATA K/0/
C
C     GET LU # OF SCHEDULING TERMINAL
C
      LU=LOGLU(I)
      ASSIGN 900 TO NEXT
      ASSIGN 9000 TO END
C
C     USE FMP ROUTINE 'FSTAT' TO OBTAIN LISTING OF ALL
C     MOUNTED CARTRIDGES.
C
 2    CALL FSTAT(STATUS)
      WRITE(LUQ,350)
 350  FORMAT(/," Cart. LU#   Last Track      Cart. Ref. # "/)
      DO 20 I=1,120,4
      DO 10 J=0,3
      BUFFER(J+1)=STATUS(J+I)
      IF(BUFFER.EQ.0) GOTO 50
 10   CONTINUE
      K=K+1
C
C     CONVERT CARTRIDGE REFERENCE NUMBER TO ASCII
C
      LBYTE=IAND(BUFFER(3),77400B)
      RBYTE=IAND(BUFFER(3),377B)
      IF(RBYTE.LT.40B.OR.RBYTE.GT.176B) RBYTE=40B
      IF(LBYTE.LT.20000B.OR.LBYTE.EQ.77400B) LBYTE=20000B
      ASCII=IOR(LBYTE,RBYTE)
C
C     LIST MOUNTED CARTRIDGES
```

```
C
 20   WRITE(LU,45) (BUFFER(N),N=1,3),ASCII
 45   FORMAT(2(I6,6X),4X,I6" = "A2)
C
C     DETERMINE WHICH CARTRIDGE CONTAINS THE PURGED FILE.
C
 50   WRITE(LU,300)
 300  FORMAT(/," Enter cartridge LU # to be processed (0 = all, 9999"
     1         " = exit) _")
      READ(LU,*) DISC
      IF(DISC.EQ.9999) GOTO END
C
C     ENSURE SPECIFIED CARTRIDGE IS MOUNTED.
C
      DO 15 I=1,120,4
      IF(STATUS(I).EQ.DISC) GOTO 4
 15   CONTINUE
      WRITE(LU,500)
 500  FORMAT(/" Cartridge not mounted.")
      GOTO 2
C
C     DETERMINE IF ALL CARTRIDGES ARE TO BE SEARCHED. IF NOT,
C     SET LOOP COUNTER AT 1
C
 4    CARTDS=1
      IF(DISC.EQ.0) CARTDS=K
      DO 90 J=1,CARTDS
      IF(CARTDS.NE.1) DISC=STATUS(((J-1))*4+1)
C
C     LOCATE DIRECTORY TRACKS OF CURRENT CARTRIDGE
C
 75   WRITE(LU,120) DISC
 120  FORMAT(/" Cartridge LU # "I3)
      IF(CARTDS.EQ.1) DTRACK=STATUS(I+1)
      IF(CARTDS.NE.1) DTRACK=STATUS(((J-1)*4)+2)
C
C     INITIALIZE FLAGS
C
      FLAG=0
      FLAG1=0
      FLAG2=0
C
C     DETERMINE THE NUMBER OF DIRECTORY TRACKS AND THE NUMBER
C     OF SECTORS PER TRACK.
C
 80   CALL EXEC(1,DISC,DIRECT,16,DTRACK,0)
      NUMBER=-DIRECT(9)
      SECTOR=DIRECT(7)
      KOUNT=0
 600  COUNT=0
 750  BLOCK=COUNT*14
 800  IF(BLOCK.LT.SECTOR) GOTO 810
      BLOCK=BLOCK-SECTOR
      GOTO 800
 810  ENTRY=1
C
C     READ IN THE DIRECTORY TRACK, ONE BLOCK AT A TIME.
```

```
C
      CALL EXEC(1,DISC,CARTDR,128,DTRACK,BLOCK)
      DO 30 I=1,8
C
C     CHECK EACH 16 WORD DIRECTORY RECORD FOR PUGRED FILES
C     FLAG  -- END OF DIRECTORY FOUND
C     FLAG1 -- CHECKING FOR DUPLICATE FILE NAME
C     FLAG2 -- LOOKING FOR EXTENTS
C     DO NOT ATTEMPT TO PROCESS TYPE 6 FILES
C
      IF(CARTDR(ENTRY+3).EQ.6) GOTO 400
      FWORD=CARTDR(ENTRY)
      IF(FWORD.LT.1.AND.FLAG2.EQ.1) GOTO 710
      IF(FWORD.EQ.0) FLAG=1
      IF(FWORD.GE.1) GOTO 400
      IF(FWORD.EQ.0.AND.FLAG1.EQ.1) GOTO 430
      IF(FWORD.EQ.-1.AND.FLAG1.EQ.1) GOTO 400
      IF(FWORD.NE.-1.AND.FWORD.NE.0) GOTO 400
C
C     DECODE DIRECTORY ENTRY
C
 710  CALL SGET(CARTDR(ENTRY+5),1,LBYTE)
      SIZE=CARTDR(ENTRY+6)/2
      SECRTY=CARTDR(ENTRY+8)
C
C     IF END OF DIRECTORY HAS BEEN ENCOUNTERED, CHECK TO
C     SEE IF NEXT ENTRY CONFORMS TO DIRECTORY RECORD FORMAT
C     IF SO, TREAT IT AS A PURGED FILE.
C
      IF(FLAG.EQ.0) GOTO 720
      LAT=DTRACK-NUMBER
      IF(CARTDR(ENTRY+3).LT.1.OR.SIZE.LT.1.OR.(ENTRY+4).GT.LAT)
     1 GOTO 210
      DO 43 M=1,4
      CALL SGET(CARTDR,(ENTRY*2+M),BYTE)
      IF(BYTE.EQ.53B.OR.BYTE.EQ.54B.OR.BYTE.EQ.55B.OR.
     1   BYTE.EQ.72B.OR.BYTE.EQ.0) GOTO 210
 43   CONTINUE
C
C     DETERMINE IF WE ARE LOOKING FOR EXTENTS, AND IF SO
C     SEE IF THE DIRECTORY ENTRY IS THE SAME AS THE
C     ORIGINAL FILE.
C
      IF(FLAG2.NE.1) GOTO 40
      DO 41 M=1,15
      IF(M.EQ.4.OR.M.EQ.5) GOTO 41
      IF(CARTDR(ENTRY+M).NE.EXTENT(M+1)) GOTO NEXT
 41   CONTINUE
C
C     IF EXTENT IS FOUND, RESTORE USING THE SAME NAME
C     AS THE ORIGINAL FILE
```

```
C
      POSN1=ENTRY*2-1
      POSN2=POSN1+31
      CALL SMOVE(CARTDR,POSN1,POSN2,EXTENT,1)
      CARTDR(ENTRY)=BUFF
      CALL EXEC(2,DISC,CARTDR,128,DTRACK,BLOCK)
      NOFEXT=NOFEXT+1
      WRITE(LU,150) NOFEXT
  150 FORMAT(/" Extent #"I3," restored.")
      GOTO NEXT
C
C     PRINT END OF DIRECTORY MESSAGE AND TRY TO PROCESS NEXT RECORD
C
 40   WRITE(LU,100)
 100  FORMAT(/," End of directory encountered, next record conforms"
     1       /," to directory record format.")
      GOTO 720
C
C     PRINT END OF DIRECTORY MESSAGE AND GET NEXT CARTRIDGE LU #
C
 210  WRITE(LU,130)
 130  FORMAT(/," End of directory encountered, next record does not"
     1       /," conform to directory record format.")
      GOTO 35
C
C     PURGED FILE ENCOUNTERED. LIST DIRECTORY RECORD AND
C     DETERMINE IF IT SHOULD BE RESTORED.
C
 720  WRITE(LU,170) (M,CARTDR(ENTRY+M),M=0,15)
 170  FORMAT(/" Directory word "I2,3X,K6"B",2(/16X,I2,3X,A2,5X,"(ASCII)"
     1       ),13(/,16X,I2,3X,K6,"B"))
      WRITE(LU,180) (CARTDR(M),M=ENTRY+1,ENTRY+3),LBYTE,SIZE,SECRTY
 180  FORMAT(/" File: --"2(A2),"     Type "I3,"      Extent number"I3,
     1       /" Size(BLKS):"I5,",    Security Code:"I6)
 620  FLAG1=0
      WRITE(LU,260)
 260  FORMAT(/" First word ASCII characters? ( 'return' = continue,"
     1        " '//'  = end) _")
      READ(LU,250) NAME
 250  FORMAT(A2)
      IF(NAME.EQ.2H//) GOTO END
      IF(NAME.EQ.2H  ) FLAG1=0
      IF(NAME.EQ.2H  ) GOTO NEXT
C
C     IS IT IS NOT TO BE RESTORED, CONTINUE LOOKING FOR ANOTHER.
C     IF IT IS, ENSURE THAT THE TWO ASCII CHARACTERS ENTERED
C     WILL NOT CREATE AN ILLEGAL OR DUPLICATE FILE NAME.
```

```
C
      DO 700 M=1,2
      CALL SGET(NAME,2*M-1,BYTE)
      IF(BYTE.GE.60B.AND.BYTE.LE.71B) GOTO 70

     IF(BYTE.EQ.53B.OR.BYTE.EQ.54B.OR.BYTE.EQ.55B.OR.
     1   BYTE.EQ.72B.OR.BYTE.EQ.40B) GOTO 70
 700  CONTINUE
      GOTO 200
 70   WRITE(LU,65)
 65   FORMAT(/" FMGR -15 illegal name")
      GOTO 620
 200  FLAG1=1
      ERROR=0
C
      REMEMBER WHERE WE ARE, THEN START OVER AND LOOK FOR
C     DUPLICATE FILE NAMES.
C
      CALL SMOVE(CARTDR,1,272,CTEMP,1)
      BUFF=NAME
      BUFF(2)=CARTDR(ENTRY+1)
      BUFF(3)=CARTDR(ENTRY+2)
      GOTO 80
C
C     INITIALIZE FOR FILE ENTENSION SEARCH
C
 310  FLAG1=0
      FLAG2=1
      NOFEXT=0
      POSN1=ENTRY*2-1
      POSN2=POSN1+31
      CALL SMOVE(CARTDR,POSN1,POSN2,EXTENT,1)
      WRITE(LU,140) BUFF
 140  FORMAT(/" Checking for extensions of file "3(A2))
      GOTO NEXT
 430  CALL SMOVE(CTEMP,1,272,CARTDR,1)
      IF(ERROR.EQ.1) GOTO 620
C
C     VERIFY THAT DIRECTORY IS TO BE MODIFIED
C
      WRITE(LU,190) NAME,(CARTDR(M),M=ENTRY+1,ENTRY+2)
 190  FORMAT(/" Restored name: "3(A2),", Confirm? (YE)S OR (NO) _")
      READ(LU,250) ANSWER
      IF(ANSWER.EQ.2H//) GOTO END
      IF(ANSWER.NE.2HYE) GOTO 620
C
C     RESTORE FILE
C
 760  CARTDR(ENTRYQ(=NAME
      CALL EXEC(2,DISC,CARTDR,128,DTRACK,BLOCK)
      WRITE(LU,220) DISC,BUFF
 220  FORMAT(/," File directory of cartridge LU #"I3" has been "
     1       /," modified to include ",3(A2))
      GOTO 310
C
C     DUPLICATE FILE NAME ROUTINE
```

```
C
 400  IF(FLAG1.EQ.0) GOTO NEXT
      ERROR=0
      DO 410 M=0,1
      IF(BUFF(M+1).NE.CARTDR(ENTRY+M)) GOTO NEXT
 410  CONTINUE
      WRITE(LU,420) BUFF
 420  FORMAT(/" FMGR -02 duplicate file name: "3(A2))
      ERROR=1
      GOTO 430
C
      CHECK NEXT RECORD
C
 900  ENTRY=ENTRY+16
 30   CONTINUE
C
C     SEE IF WE ARE AT THE END OF THE TRACK, IF SO DETERMINE
C     IF THERE IS ANOTHER ONE. IF SO CHECK IT AS WELL
C     OTHERWISE GET LU # OF NEXT CARTRIDGE.
C
      COUNT=COUNT+1
      IF(COUNT.NE.SECTOR/2) GOTO 750
      KOUNT=KOUNT+1
      IF(KOUNT.GT.NUMBER) GOTO 35
      DTRACK=DTRACK-1
      GOTO 600
C
C     INDICATE TO OPERATOR THE NUMBER OF EXTENTS FOUND
C     FOR THE RESTORED FILE.
C
 35   IF(FLAG2.EQ.1.AND.NOFEXT.EQ.0) WRITE(LU,160)
 160  FORMAT(/" No extents found")
      IF(FLAG2.EQ.1.AND.NOFEXT.GT.0) WRITE(LU,1000) BUFF,NOFEXT
 1000 FORMAT(/,X,3(A2)," restored with"I3" extents.")
      IF(FLAG2.EQ.1) GOTO END
 90   CONTINUE
      IF(CARTDS.EQ.1) GOTO 50
C
C     TERMINATE
C
 9000 WRITE)LU,110)
 110  FORMAT(2/" %END GETBK")
      END
      END$
```

HELLO — TWO SESSIONS AT ONE LU

by James Donahue/Naval Weapons Support Center
Crane, Indiana

How often have you found yourself needing some other user's capability and/or resources? Some users find this to be a recurring need, especially if you use WRITT to back up the system (including private cartridges). Sure, you could just log off from your session and log onto the other. Doing this, however, destroys the user command stack, aborts temporary loaded programs, and creates other inconveniences which may not be acceptable. What is needed is a method which allows you to log onto another session, at the same terminal, without first logging off from your session. Moreover, it should preserve the command stack and temporary loaded programs, while preventing those other inconveniences. Essentially, it should leave your session intact.

HELLO is a program which provides a quick and simple solution to the problem described in the previous paragraph. You need only type 'RU,HELLO user.group/password' or 'RU,HELLO user.group' and you will find yourself logged onto another session containing the capability and resources defined by the user.group account structure! In the latter method of scheduling HELLO, the user will be prompted for the password. When HELLO logs you onto another session, the system message file will not be printed. It is a quick, clean log on procedure. You have a fresh copy of FMGR, complete with station configuration, ready and waiting to go. And when FMGR receives the EX command, HELLO will log that session off and return control to your original session. The log off message will not be printed.

Let's take a look at the program flow of HELLO. There are three main phases to the program:

1. Gathering the user.group, password, verifying that this information is valid, and performing the log on.

2. Repairing the session control block (SCB), placing any station LUs into the session switch table (SST), renaming the program itself, and scheduling a copy of FMGR through the cloning routine XQPRG.

3. Releasing the copy of FMGR, restoring HELLO to its original name, performing the log off, and halting.

An EXEC 14 call is used to retrieve the account string, which takes the form previously mentioned. The 'RU,HELLO ' is discarded and the remaining data is checked for validity. The account string is scanned to determine whether or not a password was supplied in the runstring. If it was not a password and the supplied user.group is valid, then a prompt for the password is issued.

The validity of the user.group and password is checked by the subroutine CHKUSR (Check User). This module opens the Accounts File (+@CCT!) and finds the location of the Accounts File Directory in the Accounts File Header (record one). The directory is scanned to see if it contains the user.group that has been supplied. If it is found, the location of account entry for that user is retrieved. The password is found in the user account entry and passed back to the main program. After this, the Accounts File Header is again accessed to find the location of the Configuration Table. This table is scanned to see if there are any station LUs assigned to the user's terminal.

Once HELLO is satisfied that all is in order, DTACH is called. The call to DTACH is a requirement of XLGON. The log on and log off are performed indirectly by the Assembly routines XLGON and XLGOF. These were written by Ron Williams of HP/ Pittsburgh. These routines appeared in a very early publication of WHZUP, distributed locally in Pittsburgh.

Assuming that XLGON is successful in its attempt to perform the log on, ATACH is called to link to the new session. The next step is to repair the SCB. The actual log on and log off is performed by the system routine .CLGN, which is called from XLGON and XLGOF. During this process, the SCB is built, but it contains a small flaw. The entry for session LU 1 points back to the system console rather than to the user's terminal. The ID segment contains a pointer to the SST in word 32 (origin 0). It points to the SST length word in the SCB. The location of this ID segment address is XEQT(1717B) of the system communications area.

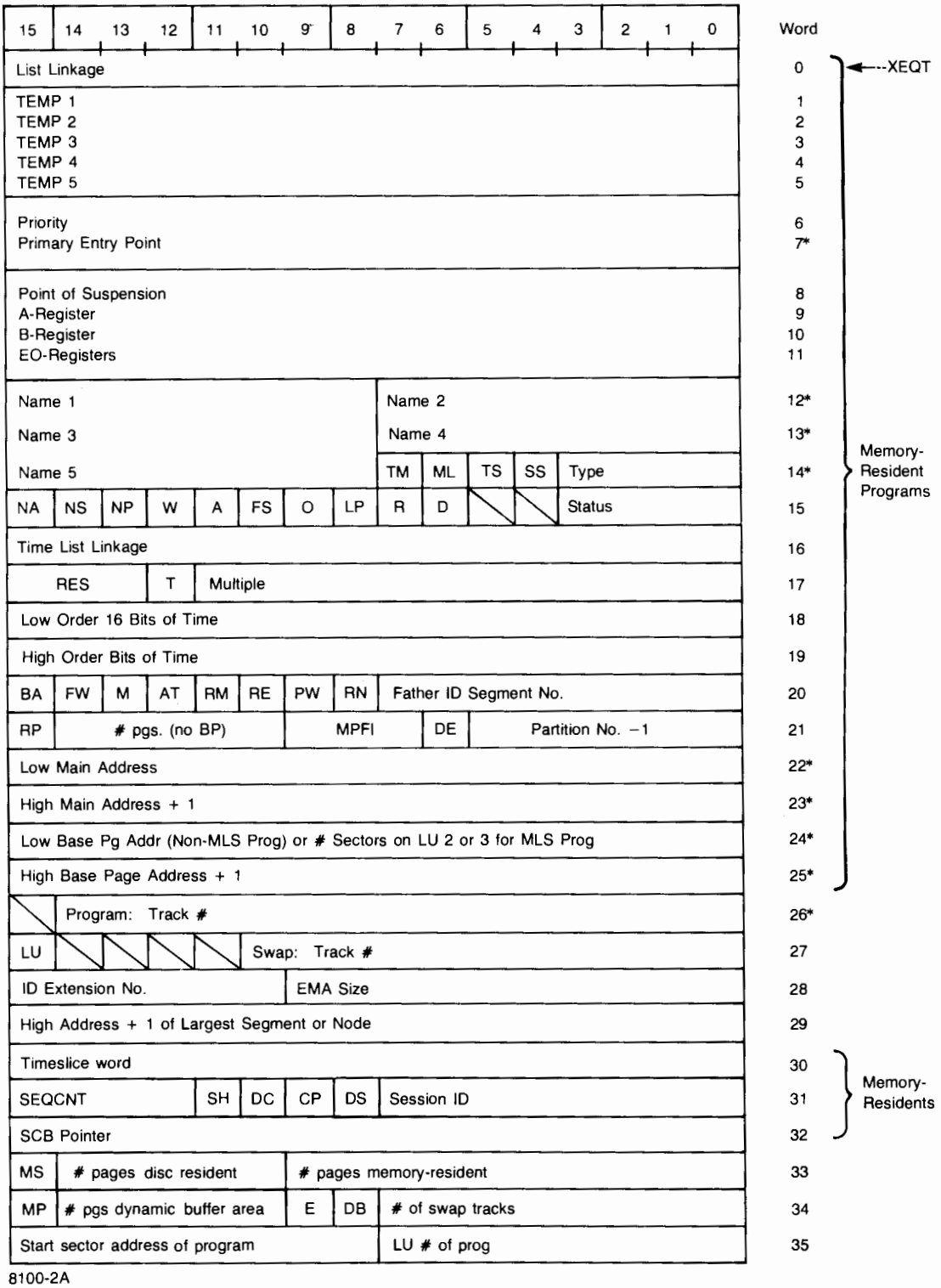

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Word
List Linkage
0
XEQT
TEMP 1
TEMP 2
TEMP 3
TEMP 4
TEMP 5
1
2
3
4
5
Priority
Primary Entry Point
6
7*
Point of Suspension
A-Register
B-Register
EO-Registers
8
9
10
11
Name 1
Name 2
12*
Name 3
Name 4
13*
Name 5
TM ML TS SS Type
14*
NA NS NP W A FS O LP R D Status
15
Memory-Resident Programs
Time List Linkage
16
RES T Multiple
17
Low Order 16 Bits of Time
18
High Order Bits of Time
19
BA FW M AT RM RE PW RN Father ID Segment No.
20
RP # pgs. (no BP) MPFI DE Partition No. −1
21
Low Main Address
22*
High Main Address + 1
23*
Low Base Pg Addr (Non-MLS Prog) or # Sectors on LU 2 or 3 for MLS Prog
24*
High Base Page Address + 1
25*
Program: Track #
26*
LU Swap: Track #
27
ID Extension No.
EMA Size
28
High Address + 1 of Largest Segment or Node
29
Timeslice word
30
SEQCNT SH DC CP DS Session ID
31
Memory-Residents
SCB Pointer
32
MS # pages disc resident # pages memory-resident
33
MP # pgs dynamic buffer area E DB # of swap tracks
34
Start sector address of program
LU # of prog
35
8100-2A

Figure A

Using the system routines IXGET and IXPUT, the system LU of the user's terminal is patched into the SCB so that session LU 1 will point to it. The mapping of the SST is not quite straightforward.

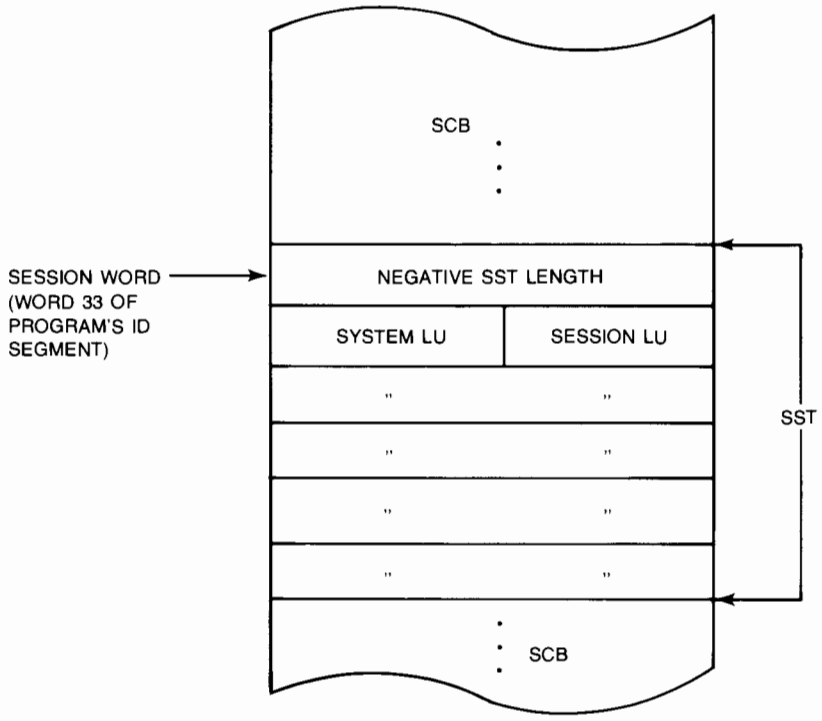


Figure B

In an entry in the SST, a single word represents a mapping of a system LU to a session LU. Bits 0 thru 7 contain the session LU and bits 8 thru 15 contain the system LU that it should be mapped to. Well, it's almost that way. Let us consider that the user's terminal is system LU 84. Then we would expect to find an entry containing an 84 in its upper byte and a 1 in its lower byte (See Figure C).

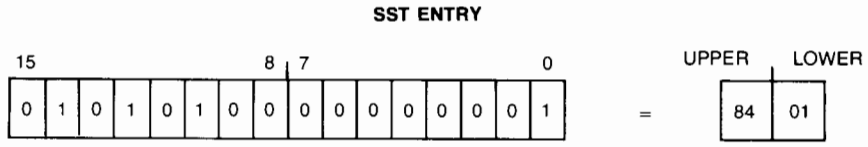


Figure C

Actually the values contained in each byte of the entry would be one less than the corresponding logical unit number. That is, we would find an 83 in the upper byte and a 0 in the lower byte (See Figure D).

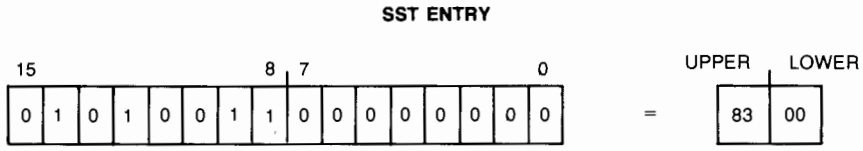


Figure D

Also, entries in the SST are placed in the table from the bottom up. For example, if the SST is 24 entries long and the user only needs to have 10 mappings, these entries would be found in the last ten table words. Any unused entries are found at the beginning of the table and are denoted by a −1 (177777B).

In addition to the SCB patch, if there are any station LUs, they are also patched into the SST by IXPUT. The name of HELLO is changed in the ID segment from HELXX (XX=system LU of user's terminal) to HEL.XX. This is done so that HELP can be run in the new session. This is necessary because, if a person ran HELLO, it would be renamed by FMGR as HELXX. If he then tried to run HELP, a FMGR 023 (Duplicate program name) would occur. This happens when FMGR also tries to rename HELP to HELXX. Then FMGR is cloned with XQPRG and the new session is ready to go. This copy will be named FM.XX. This is the reason for choosing HE.XX as the new name for HELLO.

When the user is ready to exit the new session, he simply enters EX to FM.XX. This releases the copy of FMGR. HELLO is then renamed back to HELXX to maintain the proper father-son links. XLGOF is called to perform the log off. You will then find yourself back in your original session. All of your previous processing will be intact. Problem solved!

There are a few remaining features of HELLO which may be of some interest. Multiple copies of HELLO may not be run from the same terminal. That is, schedulings of the program may not be nested. Rather, all calls to HELLO must be independent of each other. This limitation is forced by the method by which HELLO chooses the session identifier for the new session. XLGON requires that this identifier be between 100 and 200. The number used is derived from the system LU of the scheduling terminal. It is simply the system LU plus 100 (new session identifier=1XX). This should create a number within XLGON's required range and also serves as a method of identifying the location of the new session (as through WHZAT, etc.). By using this method, if HELLO were run twice in a row, the second attempt would be in error, because there would already be a session 1XX. If HELLO should be aborted abnormally before the call to XLGOF is made, the new session 1XX will be left hanging by itself without any link back to the user. It will then be necessary for the system manager to run ACCTS and shut down that session via a SD,1XX command.

HELLO was written in FTN7X on an RTE-6/VM system in order to take advantage of the DO WHILE construct. Care was taken, however, to keep the variable and subroutine names to a six character maximum. This was done so that users not operating off an RTE-6/VM system can use HELLO by simply simulating the DO WHILE construct with standard FORTRAN. If there are any questions regarding HELLO, my phone number and address are listed in the source code.

```
FTN7X,L
      PROGRAM HELLO(),Non-interactive LOGON and LOGOFF. <821208.1050>

* This program allows a user to log onto another person's account while
*   remaining logged onto his own.  This gives the user the resources
*   and capability of the other user until the program is terminated.
*   Once the HELLO is terminated the user returns to his first session
*   with all earlier processing still intact, i.e., TR STACK, edits, etc.
*   If HELLO has been used to log onto another account then it cannot
*   be used again until the user logs off this account.  That is, HELLO
*   calls cannot be nested among each other.

* HELLO is invoked by:
*                       RU,HELLO user.group
*                                or
*                       RU,HELLO user.group/password
*                                or
*                       RU,HELLO user          '.GENERAL'  implied
*                                or
*                       RU,HELLO user/password  '.GENERAL' implied
*
* The program only requires an eight page partition and must be loaded with
*   the assembly routines XLGON and XLGOF which appeared in a very early
*   publication of WHZUP.

* Author:  Bill Donahue
*          NWSC 7071
*          Crane, IN 47522
*          (812) 854-3206                                              25 Aug 82

      IMPLICIT INTEGER(A-Z)
      DIMENSION Acct(16),Fmgr(3),Code(2),Parm(5),Pass1(5),Pass2(5)
      DIMENSION Table(5),Genrl(4)
      LOGICAL Found,Pass,Statn,Group
      DATA fmgr/'FMGR  '/
      DATA Code/2*0/
      DATA Genrl /'.GENERAL'/

* Retrieve the runstring and the length of the runstring (in characters)
*   and check to see that a 'user.group' was specified in the runstring.
*   If no user was specified then inform the user and stop execution.

      CALL EXEC(14,1,Acct,-32)
      CALL ABREG(A,Lacct)
      IF(Lacct.LE.9) THEN
        CALL EXEC(2,1,22HHi! No user specified.,11)
        CALL EXEC(6)
      ENDIF

* Calculate the length (in characters) of the 'user.group/password' string
*   by subtracting the 9 characters ('RU,HELLO ') retrieved by EXEC from the
*   runstring.  Also move the Acct string to the beginning of the buffer.

      Lacct=Lacct-9
      CALL SMOVE(Acct,10,10+Lacct,Acct,1)

* Test the last character for a space and decrement
*   Lacct one if it is.

      Call Sget(Acct,Lacct,Count)
      IF(Count.eq.40B) Lacct=Lacct-1
```

```
* Scan the runstring to see if the password was supplied in it.  Set the
*   flag Pass accordingly.  The '/' character signifies that a password
*   was supplied

      Pass=.FALSE.
      Count=0
      DO WHILE(.NOT.Pass.AND.Count.LE.Lacct)
        Count=Count+1
        IF(JSCOM(Acct,Count,Count,2H/ ,1,Error).EQ.0) Pass=.TRUE.
      ENDDO

* Scan the runstring to see if a GROUP was supplied in it.  Set the
*   flag Group accordingly.  The character '.' signifies that a
*   group was supplied.  If no group was supplied, insert the char-
*   acters '.GENERAL' in the runstring.

      Group = .FALSE.
      Count2 = 0
      DO WHILE(.NOT.Group.AND.Count2.LE.Lacct)
        Count2 = Count2 + 1
        if(JSCOM(Acct,Count2,Count2,2H. ,1,Error).EQ.0) Group=.TRUE.
      ENDDO

      IF(.NOT. Group) THEN
        IF(Pass) CALL SMOVE(Acct,Count,Lacct,Pass2,1)
        CALL SMOVE(Genrl,1,8,Acct,Count)
        Lacct = Lacct + 8
        Count = Count + 8
        IF(Pass) CALL SMOVE(Pass2,1,10,Acct,Count)
      ENDIF

* Call the routine that scans the accounts file to see if the 'user.group'
*   is a valid one.  Acct and Lacct are passed and the subroutine will
*   set the flag Found and if the 'user.group' is valid it will
*   return the correct password in Pass1.

      Lu=LOGLU(Syslu)
      CALL Chkusr(Acct,Lacct,Found,Pass1,Syslu,Statn,Table,Ltable)

* If the 'user.group' is invalid then inform the user and stop execution
*   of the program.  If the password was supplied then the length of the
*   'user.group' will be just the value in Count-1.  Otherwise it just be
*   the value of Count.

      IF(.NOT.Found) THEN
        CALL EXEC(2,1,30H Logon Error.  No such user: _,15)
        IF(Pass) Count = Count - 1
        CALL EXEC(2,1,Acct,-Count)
        CALL EXEC(6)
      ENDIF
```

```
* If the 'user.group' supplied by the user is valid and no password was
*   supplied AND a password is required then prompt the user for the
*   password.  To insure that the
*   password given will be in capital letters AND each word of the user
*   supplied password with 157737B.  This is done because the password
*   returned from the Accounts file will be in capitals.  After this is
*   done move the password into Acct in preparation for the call to XLGON.

      IF(Pass1(1) .NE. 0)THEN              !  Password required?
        IF(.NOT.Pass) THEN                 !      Y E S
          CALL EXEC(2,1,12H Password: _,6)
          CALL EXEC(1,1,Pass2,-10)
          CALL ABREG(A,Lpass2)
          DO I=1,Lpass2/2+1
            Pass2(I)=Pass2(I).AND.157737B
          ENDDO
          CALL SMOVE(2H/ ,1,1,Acct,Lacct+1)
          CALL SMOVE(Pass2,1,Lpass2,Acct,Lacct+2)
          Lacct=Lacct+Lpass2+1
        ELSE
          CALL SMOVE(Acct,Count+1,Lacct,Pass2,1)
          Lpass2=Lacct-Count
        ENDIF

* Compare the user supplied password to the one retrieved from the Accounts
*   file.  If they do not match then inform the user that an invalid
*   password has been supplied and stop execution.

        IF(JSCOM(Pass2,1,Lpass2,Pass1,1,Error).NE.0) THEN
          CALL EXEC(2,1,35H Illegal access.  winvalid password!,-35)
          CALL EXEC(6)
        ENDIF
      ENDIF

* Since the user has supplied a valid 'user.group' and the appropriate
*   password continue onward.  Use the session and system lu of the
*   terminal and then call DTACH to remove the program from session.
*   Set up the code for XLUEX.  Create a unique session id by adding
*   100 to the system lu from which the program was scheduled.

      Code(1)=Syslu
      Sesslu=Syslu+100

      CALL DTACH

* Attempt a non-interactive logon and inform user of the status of
*   this attempt.  If this attempt was successful clone a copy of FMGR
*   with XQPRG.  If the error return is zero then the logon was
*   successful.  On a successful logon call ATACH to attach to the
*   new session.

      CALL XLGON(Error,Sesslu,Acct,Lacct)

      IF(Error.EQ.0) THEN
        CALL ATACH(Sesslu)
```

```
* Find the SCB of the newly logged on session via the id-segment and
*   patch the terminal lu into the new SCB.  This is done by scanning
*   the session switch table for the entry containing session lu 1 and
*   adding the terminal lu to the map.

        Idseg=IXGET(1717B)
        Sst=IXGET(Idseg+32)
        Lsst=-IXGET(Sst)
        Count=1
        DO WHILE(Count.LE.Lsst.AND.IXGET(Sst+Count).NE.0)
          Count=Count+1
        ENDDO

        IF(IXGET(Sst+Count).eq.0) THEN
          CALL IXPUT(Sst+Count,(Syslu-1)*256)

* Change the name of the program in the ID segment to HE.XX so that when
*   HELLO is running the user can also run HELP.  This is done because
*   if HELXX is already running then FMGR does not know how to rename
*   HELP.

          Name=IXGET(Idseg+13)
          Name=(Name.AND.377B).OR.027000B
          CALL IXPUT(Idseg+13,Name)

* Check to see if there are any station lus to be assigned to this
*   session.  If there are then put the lu assignments passed back from
*   the subroutine Chkusr into the SST of this session.
*   the subroutine Chkusr into the SST of this session.

          IF(Statn) THEN
            Offset=1
            Entry=IXGET(Sst+Offset)
            DO WHILE(Entry.EQ.-1)
              Offset=Offset+1
              Entry=IXGET(Sst+Offset)
            ENDDO
            DO I=1,Ltable
              CALL IXPUT(Sst+Offset-I,Table(I))
            ENDDO
          ENDIF
```


```
* If the SCB patch was successful then inform the user that he has been
*   logged on and clone the copy of FMGR with XQPRG.  If FMGR cannot be
*   cloned then inform the user that FMGR is not available and stop
*   execution.  If the patch was not successful then inform the user of
*   the status and exit.

          CALL EXEC(2,1,11H Logged on!,-11)

          CALL XQPRG(Dcb,9,Fmgr,Parm,0,0,Parm,Error)
            IF(Error.NE.0) THEN
              CALL EXEC(2,1,20H FMGR not available!,10)
            ENDIF
        ELSE
          CALL EXEC(2,1,24H SCB entry patch failed!,12)
          CALL EXEC(6)
        ENDIF
      ELSE
         CALL CITA(Error,Acct)
         CALL XLUEX(2,Code,16H XLGON error # _,8)
         CALL XLUEX(2,Code,Acct(3),1)
         CALL EXEC(6)
         CALL EXEC(6)
      ENDIF

* When the user exits FMGR call XLGOF to log off the active session logged
*   into the account Acct.  Also inform the user of the status of the
*   logoff attempt.  Change the name in the id segment back to its original
*   form so that the id segment will be properly found by FMGR.

      Name=IXGET(Idseg+13)
      Name=(Name.AND.377B).OR.046000B
      CALL IXPUT(Idseg+13,Name)

      CALL XLGOF(Error)

      IF(Error.EQ.0) THEN
        CALL XLUEX(2,Code,25H Successfully logged off.,-25)
      ELSE
        CALL CITA(Error,Acct)
        CALL XLUEX(2,Code,16H XLGOF error # _,8)
        CALL XLUEX(2,Code,Acct(3),1)
      ENDIF

      END
```

```
      SUBROUTINE Chkusr(Acct,Lacct,Found,Passwd,Syslu,Statn,Table,
     +                  Ltable)

* The purpose of this subroutine is to scan the Accounts file and determine
*   if the user has supplied a valid 'user.group'.  Found is a flag which
*   will be true if the 'user.group' is valid and will be false otherwise.
*   The password will also be retrieved so that the user supplied password
*   may be checked before a call is made to XLGON.  If there are any station
*   lus they will be passed back and the flag Statn will be set.

      IMPLICIT INTEGER(A-Z)
      DIMENSION Acct(20),Acctfl(3),Buffer(128),Dcb(144),User(5),Group(5)
     +          ,Passwd(5),Table(5)
      LOGICAL Found,Statn
      DATA Acctfl/'+@CCT!'/,Lu2/-2/,Secur/(see note)/

* Open the accounts file.  It must be opened with non-exclusive access.

      CALL OPEN(Dcb,Error,Acctfl,1,Secur,Lu2)
      IF(Error.LT.0) CALL FMPerr(Error)

* Read the first record of the accounts file and determine the location
*   of the account directory.

      CALL READF(Dcb,Error,Buffer,128,Lread,1)
      IF(Error.LT.0) CALL FMPerr(Error)
      Recnum=Buffer(5)

* Scan through the user entries of the account directory to see if the
*   supplied 'user.group' is valid.  If it is valid then extract the
*   password for that user to be returned for later comparison.

      CALL READF(Dcb,Error,Buffer,128,Lread,Recnum)
      IF(Error.LT.0) CALL FMPerr(Error)
      Posn1=1
      Found=.FALSE.
      Statn=.FALSE.
      DO WHILE(.NOT.Found.AND.Buffer(Posn1).NE.0)
        IF(Buffer(Posn1+11).NE.0) THEN
          Luser=(Buffer(Posn1)/256).AND.377B
          Lgroup=Buffer(Posn1).AND.377B
          CALL SMOVE(Buffer(Posn1+1),1,10,User,1)
          CALL SMOVE(Buffer(Posn1+6),1,10,Group,1)

* If the search is successful then get the record number which points to
*   the location of the user's account definition which contains the
*   password.

          IF(JSCOM(User,1,Luser,Acct,1,Error).EQ.0.AND.JSCOM(Group,1,
     +    Lgroup,Acct,Luser+2,Error).EQ.0.AND.JSCOM(2H. ,1,1,Acct,
     +    Luser+1,Error).EQ.0) THEN
            Found=.TRUE.
            Recnum=Buffer(Posn1+14)
```

NOTE: See your system manager to obtain the Accounts file security code and insert it in the data statement for SECUR.

```
* Check to see if the account definition is in the first or second half of
*   the block. Then get the length of the password and move the correct
*   password from the ACCTs file to the buffer.

            Offset=0
            IF(Recnum.LT.0) THEN
              Offset=64
              Recnum=Recnum.AND.77777B
            ENDIF
            CALL READF(Dcb,Error,Buffer,128,Lread,Recnum)
            IF(Error.LT.0) Call FMPerr(Error)
            Lpass=Buffer(Offset+1).AND.377B
            CALL SMOVE(Buffer(Offset+2),1,Lpass,Passwd,1)

* Since the 'user.group' is valid then check the configuration table to
*   see if there are any station Lus to be assigned to this location.

            CALL READF(Dcb,Error,Buffer,128,Lread,1)
            IF(Error.LT.0) CALL FMPerr(Error)
            Recnum=Buffer(2)
            CALL READF(Dcb,Error,Buffer,128,Lread,Recnum)
            IF(Error.LT.0) CALL FMPerr(Error)
            Posn2=1
            Length=Buffer(Posn2)
            DO WHILE(Length.NE.0.AND..NOT.Statn)
              IF(Buffer(Posn2+1)/256+1.EQ.Syslu) THEN
                Statn=.TRUE.
                DO I=1,Length-1
                  Table(I)=Buffer(Posn2+I+1)
                ENDDO
                Ltable=Length-1
              ENDIF
              Posn2=Posn2+Length+1
              Length=Buffer(Posn2)
            ENDDO
          ENDIF
        ENDIF

* Check to see if there is a need to read another block of account
*   definitions from the Accounts file.

        Posn1=Posn1+16
        IF(Posn1.GT.113.AND..NOT.Found) THEN
          Posn1=1
          Recnum=Recnum+1
          CALL READF(Dcb,Error,Buffer,128,Lread,Recnum)
          IF(Error.LT.0) CALL FMPerr(Error)
        ENDIF
      ENDDO
```

```
* Close the Accounts file and return to the calling module.

      CALL CLOSE(Dcb,Error)
      IF(Error.LT.0) CALL FMPerr(Error)
      RETURN
      END


      SUBROUTINE FMPerr(Error)
      IMPLICIT INTEGER(A-Z)
      DIMENSION Err(5)
      DATA Err(1)/2HFM/,Err(2)/2HGR/
      CALL CITA(Error,Err(3))
      CALL EXEC(2,1,22H Error in FMP call!  _,11)
      CALL EXEC(2,1,Err,5)
      CALL EXEC(6)
      END
```

```
ASMB,R,L
      NAM XLGON,7 Non-interactive LOGON procedure   <810514.1605>
*
*     Subroutine to allow a program to log itself onto
*     another account.  The program must be non-session at
*     the time the call to XLGON is made, I.E. the program
*     must have logged itself off of its original account
*     via a call to the counter part routine XLGOF or a call
*     to DTACH
*
*     CALL XLGON(IE,ID,ACCT,LACCT)
*
*     IE    = Error code return
*               0 = Normal return, no error
*              >0 = Logon error number
*              -1 = Program already in session
*              -2 = Session not initialized or not installed
*              -3 = Class I/O error on logon reply
*              -4 = SCB not built properly
*              -5 = Session id not in range 100-200
*     ID    = Session id to be used (100-200)
*     ACCT  = ASCII array designating the account to log onto
*               "USER.GROUP/PASSWORD" (Password must be supplied!)
*     LACCT = Length of acct array in characters
*
*     AUTHOR: Ron Williams          Version 1.0
*             Systems Engineer      <810514.1605>
*             HP-Pittsburgh
*
B     EQU 1
      EXT SESSN,.CLGN,.ENTR,EXEC,LUSES,$LIBR,$LIBX
      ENT XLGON
IE    NOP
ID    NOP
ACCT  NOP
LACCT NOP
XLGON NOP           Entry point / Exit address
      JSB .ENTR     Get parameter addresses
      DEF IE
```

```
        LDA ID,I        Check for negative or bad session ID
        AND =B177400
        SZA
        JMP BADID
        LDA =D-100      Check to see if session ID >= 100
        ADA ID,I
        SSA
        JMP BADID
        LDA ID,I        Check for session ID > 200
        CMA,INA
        ADA =D200
        SSA
        JMP BADID
        JSB SESSN       See if already attached to session
        DEF *+2
        OCT 1717
        SEZ,RSS
        JMP INSES       Program in session
        LDA ID,I        Call LOGON
        STA CODE
        LDB ACCT
        LDA LACCT,I
        CMA,INA         Convert to -# characters
        JSB .CLGN
CODE    NOP
        SSA             Skip if no error
        JMP NOSES       Session not initialized or not installed
        STA CLASS       Save class no. for LOGON reply
LRSPN   JSB EXEC        Get logon reply
        DEF *+8
        DEF DS21
        DEF CLASS
        DEF BUFFR
        DEF ZERO
        DEF IP1
        DEF IP2
        DEF IP3
        JMP CLERR       Class I/O error
        LDA IP3         Fetch call type
        CPA =D1         Must be read or write/read
        RSS
        JMP LRSPN       Try again
        LDA IP2         Fetch LOGON status
        SSA,RSS         If negative or zero, then continue
        SZA,RSS
        RSS
        JMP LRSPN       Else get next message
        STA IE,I        save logon error
        LDA CLASS       Clear save class buffer bit
        XOR =B20000
        STA CLASS
AGAIN   JSB EXEC        Release LOGON reply class #
        DEF *+5
        DEF DS21
        DEF CLASS
        DEF BUFFR
        DEF ZERO
        RSS
        JMP AGAIN
        LDA IE,I        Complete processing id-segment if IE=0
        SZA
```

```
        JMP LGERR       Go take LOGON error return
        JSB $LIBR       Go patch id-segment
        NOP
        JSB LUSES       Get pointer to SCB
        DEF *+2
        DEF ID,I
        SZA,RSS         Skip if found
        JMP NOSCB
        LDB 1717B       Get id-segment address
        ADB =D32        Offset to session word
        XSA B,I         Store session word
        ADB =D-1        Backup to sesion id field
        XLA B,I         Mask out old session id
        AND =B177400
        IOR ID,I        Replace with new session id
        XSA B,I         Put back word
        JSB $LIBX       Turn system back on
        DEF *+1
        DEF *+1
        JMP XLGON,I     Normal return
LGERR   ALF,ALF         Extract LOGON error code
        RAL,RAL
        AND =B77
        JMP RTN
NOSCB   LDA =D-4        Scb not built properly
        JSB $LIBX
        DEF *+1
        DEF RTN
CLERR   LDA =D-3        Bad class I/O get on LOGON reply
        JMP RTN
BADID   LDA =D-2        Session id not in range 100-200
        JMP RTN
INSES   CCA             Already logged on error return
        JMP RTN
NOSES   LDA =D-2        Session not installed or uninitialized error return
RTN     STA IE,I
        JMP XLGON,I
CLASS   NOP             Logon reply class number
DS21    OCT 100025
BUFFR   NOP
ZERO    DEC 0
IP1     NOP
IP2     NOP
IP3     NOP
        END
```

```
ASMB,R,L
      NAM XLGOF,7 Non-interactive LOGOFF procedure  <810514.1631>
*
*     Routine to force a logoff for the calling program
*
*     CALL XLGOF(IE)          This form of call uses the session
*                             ID from the program's id-segment
*
*     CALL XLGOF(IE,ID)       This form of the call uses the argument
*                             "ID" as the session id.  It must be in
*                             the range of 100-200
*
*     IE = Error return parameter
*          >0 = Logoff error return
*           0 = Logoff successful, normal return
*          -1 = Session id not in range 100-200
*          -2 = SCB not found for given session ID
*          -3 = Session not installed or uninitialized
*          -4 = Bad class I/O get on LOGOF reply
*
*     Author: Ron Williams          Version 1.1
*             Systems Engineer      <810514.1631>
*             HP-Pittsburgh
*
B     EQU 1
      EXT .ENTR,EXEC,DTACH,$DSCS,LUSES,$LGOF
      ENT XLGOF
IE    NOP
ID    OCT 100000
XLGOF NOP           Entry point / exit address
      JSB .ENTR     Get parameter addresses
      DEF IE
      LDA ID        check for single argument
      SSA,RSS
      JMP DOUBL
      LDB 1717B     Fetch this program's id-segment address
      ADB =D31      Offset to session id word
      XLA B,I       Fetch word
      AND =B377     Mask out garbage
      STA .ID       Store for local use
      JMP CHECK
DOUBL LDA ID,I      Fetch session id locally
      STA .ID
CHECK LDA .ID       Check for negative or bad session ID
      AND =B177400
      SZA
      JMP BADID
      LDA =D-100    Check to see if session id >= 100
      ADA .ID
      SSA
      JMP BADID
      LDA .ID       Check for session id > 200
      CMA,INA
      ADA =D200
      SSA
      JMP BADID
      LDB 1717B     Fetch calling program's session ID
      ADB =D31
      XLA BU,I
      AND =B377
```

```
       CPA .ID        If requested session id matches that of calling
       JMP *+2        Then detach it from the session
       JMP DOIT
       ASB DTACH
       DEF *+1
       JMP DOIT
BADID  CCA            Session ID not in range 100-200
RTN    STA IE,I       Store error parameter
       LDB =B100000   Reset ID address to null

      STB ID
       JMP XLGOF,I    Return
BADSM  LDA =D-2       Session not installed or unitialized error return
       JMP RTN
NOSES  LDA =D-3       SCB does not exist for session ID
       JMP RTN
CLERR  LDA =D-4       Bad class I/O get on LOGOF reply
       JMP RTN
DOIT   LDA .ID        Add in kill active programs bit
       IOR =B20000
       STA .ID
       XLA $DSCS      Fetch disc pool pointer (up/down)
       SSA
       JMP NOSES      Session not init. or not installed
       LDA .ID        Mask out all butr session id
       AND =B377
       STA BUFFR
       JSB LUSES      Find SCB address
       DEF *+2
       DEF BUFFR
       SZA,RSS        See if found it
       JMP BADSM
       STA .SCB
       XLA $LGOF      Get lgoff's class #
       SZA,RSS        See if session up yet
       JMP NOSES
       IOR =B20000    Set no deallocate bit
       STA .CLAS
       JSB EXEC       Make sure LGOFF exists & is executing
       DEF *+3
       DEF DS10
       DEF LGOFF
       NOP
       CPB =A05       Check for SC05 error
       JMP NOSES
       CLA            Get class # for LGOFF reply
       STA CLASS
       JSB EXEC       Do zero length class write to reply class
       DEF *+8
       DEF D18
       DEF ZERO
       DEF ZERO
       DEF ZERO
       DEF ZERO
       DEF ZERO
       DEF CLASS
```


```
        JSB EXEC        Issue class write/read to LGOFF
        DEF *+8
        DEF DS20
        DEF ZERO
        DEF CLASS
        DEF D1
        DEF .ID
        DEF .SCB
        DEF .CLAS
        JMP NOSES
        JSB EXEC        Make sure LGOFF IS executing
        DEF *+3
        DEF DS10
        DEF LGOFF
        NOP
        LDA CLASS       Insert save class bit into reply class #
        IOR =B20000
        STA CLASS
LRSPN   JSB EXEC        Get LOGON reply
        DEF *+8
        DEF DS21
        DEF CLASS
        DEF BUFFR
        DEF ZERO
        DEF IP1
        DEF IP2
        DEF IP3
        JMP CLERR       Class I/O error
        LDA IP3         Fetch call type
        CPA =D1         Must be read or write/read
        RSS
        JMP LRSPN       Try again
        LDA IP2         Fetch LOGON status
        SSA,RSS         If negative or zero, then continue
        SZA,RSS
        RSS
        JMP LRSPN       Else get next message
        STA IE,I        Save LOGON error
        LDA CLASS       Clear save class buffer bit
        XOR =B20000
        STA CLASS
AGAIN   JSB EXEC        Release LOGON reply class #
        DEF *+5
        DEF DS21
        DEF CLASS
        DEF BUFFR
        DEF ZERO
        RSS
        JMP AGAIN
        LDA IE,I        Complete processing
        JMP RTN
CLASS   NOP             LOGOF reply class number
DS10    OCT 100012
D18     DEC 18
DS20    OCT 100024
LGOFF   ASC 3,LGOFF
```

```
      .CLAS NOP           LGOFF class # storage
      .SCB  NOP           Session SCB address storage
      .ID   NOP           LGOFF control parameter storage
      DS21  OCT 100025
      BUFFR NOP
      IP1   NOP
      IP2   NOP
      IP3   NOP
      ZERO  DEC 0
      D1    DEC 1
            END
```

```
WAITING FOR INPUT
:wh

13:34:13:920
-------------------------------------------------------------------------
  PRGRM T PRIOR PT  SZ DO.SC.IO.WT.ME.DS.OP.     .PRG CNTR.  .NEXT TIME.
-------------------------------------------------------------------------
**FMG84 3 00090 14  12 * * * *  3,WHZAT * * * * * P:36136
  WHZAT 1 00001  0   . .  1,  . . . . . . . . . . P:36540
.
-------------------------------------------------------------------------
ALL LU'S OK
ALL EQT'S OK
LOCKED LU'S (PROG NAME)  63(EDI63), 64(EDI64), 87(EDI87),117(DBD59),
-------------------------------------------------------------------------
13:34:15: 20

:sl
SLU   1=LU  # 84 = E  34
SLU   2=LU  #  2 = E   1
SLU   3=LU  #  3 = E   1 S 1
SLU   4=LU  # 85 = E  34
SLU   5=LU  # 86 = E  34
SLU   6=LU  #  6 = E   7
SLU   7=LU  #116 = E  53
SLU   8=LU  #105 = E  38 S 5
SLU   9=LU  #  9 = E   8
SLU  10=LU  # 10 = E   1 S 2
SLU  11=LU  # 11 = E   1 S 3
SLU  12=LU  # 12 = E   1 S 4
SLU  13=LU  # 13 = E   1 S 5
SLU  14=LU  # 14 = E   1 S 6
SLU  17=LU  # 17 = E   1 S 9
```

```
SLU   22=LU   # 22 = E    2 S 1
SLU   28=LU   # 28 = E    2 S 7
SLU   33=LU   # 33 = E    2 S12
SLU   44=LU   # 44 = E    2 S23
SLU   57=LU   # 57 = E    9
:edit
EDI84 : Use ? for help
FI,namr specifies file to edit.
EOF
/ru,hello bill_d.co_op    <<----- Running HELLO from an EDIT!
 Password:
 Logged on!       <<----- Logged onto the new session!
:wh

13:34:38:890
-------------------------------------------------------------------------
  PRGRM T PRIOR PT   SZ DO.SC.IO.WT.ME.DS.OP.      .PRG CNTR.   .NEXT TIME.
-------------------------------------------------------------------------
**FMG84 3 00090 14   12 * * * *  3,EDI84 * * * * * P:36136
  EDI84 4 00051 50   22 . . . .  3,HE.84 . . . . . P:36364
  HE.84 3 00099 36    8 . . . .  3,FM.84 . . . . . P:34233
  FM.84 3 00090 40   12 . . . .  3,WHZAT . . . . . P:36136
  WHZAT 1 00001  0    . .  1,  . . . . . P:36540
.
-------------------------------------------------------------------------
ALL LU'S OK
ALL EQT'S OK
LOCKED LU'S (PROG NAME)  63(EDI63), 64(EDI64), 87(EDI87),117(DBD59),
-------------------------------------------------------------------------
13:34:40:300

:sl
SLU    1=LU   # 84 = E   34
SLU    2=LU   #  2 = E    1
SLU    3=LU   #  3 = E    1 S 1
SLU    4=LU   # 85 = E   34     <<----- Station lus 4 and 5 (ctu)!
SLU    5=LU   # 86 = E   34
SLU    6=LU   #  6 = E    7
SLU    7=LU   #104 = E   38 S 4
SLU    8=LU   #105 = E   38 S 5
SLU    9=LU   #  9 = E    8
SLU   10=LU   # 10 = E    1 S 2
SLU   11=LU   # 11 = E    1 S 3
SLU   12=LU   # 12 = E    1 S 4
SLU   13=LU   # 13 = E    1 S 5
SLU   14=LU   # 14 = E    1 S 6
SLU   17=LU   # 17 = E    1 S 9
SLU   22=LU   # 22 = E    2 S 1
SLU   28=LU   # 28 = E    2 S 7
SLU   33=LU   # 33 = E    2 S12
SLU   44=LU   # 44 = E    2 S23
SLU   57=LU   # 57 = E    9
:ex
 $END FMGR
 Successfully logged off.
  Resume EDI84 on
/a/                    <<----- Returned to edit
EDI84 aborted by user
end of edit
:
```

OPTIMIZING MLS-LOC LOADER PERFORMANCE

by Julie Leon

The MLS-LOC loader's performance can be optimized by using the the following techniques:

1. Don't use the SGMTR's "A" option when generating the MLS-LOC command file. When the A option is not used, NA and SY commands are not emitted for leaf node modules that would be loaded anyway during default searches. Therefore, the MLS loader accesses the command file on disc fewer times.

2. Use one indexed library. This saves time doing multiple opens and when searching. The program INDXR can be used to merge and index the library.

3. Delete the one line comments in the MLS command file. This reduces the number of disc accesses required to read the command file. This can be done easily with the following EDIT commands:

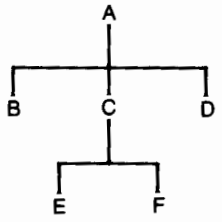
```
:RU,EDIT,#PROG          edit the command file
/SEREON                 turn on regular expression option
/1$X/ //Q               remove spaces quietly
/1$D/^[A-Z]/AQ          delete all comments quietly
/EC#PROG2               create a second output file
```

4. Of course using the largest possible path will allow SGMTR to build the smallest number of nodes. Since there is loader overhead associated with each node, the fewer the nodes, the faster the load.

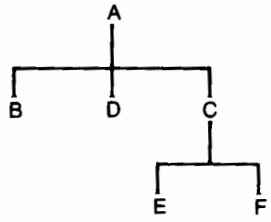
5. If all else fails, the tree structure produced by SGMTR can be manipulated by hand. If the tree has leaves (bottom nodes) that are not all on the same level, it is possible that the tree structure can be changed to save more loading time. Rotating base page when the loader runs out of links, and the corresponding overhead can be optimized by grouping brother nodes that are leaves at the same level.

   See example below:

Not Optimized



Optimized



These techniques can substantially decrease loading time. We tested two medium size programs to determine the savings. One was the Pascal demo PSDEM, the other was the SPICE program. See chart below for savings when loading the SPICE program.

MLS-LOC TIMING TESTS

SPICE	#1	#2	#3	#4
SGMTR "A"	Yes	No	No	No
INDEX LIB	No	No	Yes	Yes
COMMENTS	Yes	Yes	Yes	No
TIME(MINUTES)	60	43	24	21
SAVINGS	—	28	44	4

TROUBLE FREE SEGMENTED DGL GRAPHICS PROGRAMS

by Anjali Magana/HP ESO

A number of points must be considered when a segmented program using Graphics/1000-II DGL communicates with a single workstation. If all calls to DGL are included in one program segment, then users should simply follow the rules described in the DGL Programmer's Reference Manual, under Segmented DGL Programs. However, if all of the calls to DGL are not contained within one program segment, then using the following guidelines in addition to rules in the manual will eliminate potential execution problems.

1. When loading any program segment in which a locator device will be used in conjunction with a display device (e.g. tracking the position of the locator on the graphics display) then both the locator library and the display library should be searched.

2. When operating in system buffering mode, ZMCUR should always be the last graphics call made in segments which reference the graphics display device.

3. If an application program takes advantage of display devices which support hardware clipping, then all calls which modify the viewing system of the display (ZBEGN, ZASPK, ZDLIM, ZVIEW, and ZWIND) can only be made in a program segment which calls ZDINT. The exception to this is when there is no display device enabled at the time these calls are made.

4. All devices should be explicitly terminated (e.g. ZLEND) instead of relying on ZEND to implicitly terminate them.

COMMUNICATOR/1000 INDEX

With this issue, we close out Volume V of the Communicator/1000. The index below covers the six issues of Volume V in chronological order, with each article coded according to its category.

Legend

BI	Bit Bucket
OS	Operating Systems
OM	Operations Management
LA	Languages
IN	Instrumentation
BU	Bulletins

COMMUNICATOR INDEX BY ISSUE, VOLUME V

BIT BUCKET

ISSUE	CAT	TITLE	AUTHOR	
4	IN	The Fundamentals of HP-IB Addressing	Neal Kuhn	53
4	BU	The Most Powerful RTE Ever	Jim Williams	59
4	BU	New Languages Extend Programming Capabilities on RTE-6/VM	Linda Haar	61
5	BI	Who's Logged On?	Dan Wagner	14
5	BI	Who Am I?	J. L. DeSchutter	20
5	BI	How Long Have I Been Here?	William J. Loye	23
5	BI	Executing a Procedure After Logoff	Bob Desinger	
5	BI	Fast FORTRAN — An Update	John Pezzano	30
5	BI	Accessing Physical Memory	Stephen Botzko	
5	BI	More Notes on the Use of Undeclared Memory	Jeff Wynne	38
5	BI	Short Formatted I/O for LUs in Pascal/1000	Dave Redmond	42
5	BI	Pascal Error Trapping and Reporting	Jeffrey Hirschl	53
5	BI	MVDIR — The Case of the Moving Directory	John McCabe	59
5	BI	How to Build System Utilities Using a Disc Directory and Edit/1000 Subsystem	Bob Gordon	61
5	BI	Restart Spooled Printing	Jay McCanta	74
5	BI	Set Up Your 2608 Line Printer	Linnea L. Fort	76
5	BI	1351A Graphics Generator With a 21MXM Computer in RTE-IVB	K.H. Kitching, J. Robinson	80
6	BI	For/Next Loops in FORTRAN	Alf Lacis	10
6	BI	Restoring Purged Files	Paul Dunphy	13
6	BI	HELLO — Two Sessions at One LU	James Donahue	21
6	BI	Optimizing MLS-LOC Loader Performance	Julie Leon	40
6	BI	Trouble Free Segmented DGL Graphics Programs	Anjali Magana	42
6	BI	Communicator/1000 Index for Volume V	Editor	43
6	BI	Performance Concepts for Software Design and Implementation	Marc Katz	45
6	OS	Simplified Development of Custom I/O Drivers	John Trueblood	48
6	OM	Tricks with Edit/1000	Michael Wiesenberg	64
6	OM	Manufacturing Productivity Automation at Bendix	Michael Miller	73
6	LA	2250 Buffer Management from Downloaded Subroutines	Diana Bare	77
6	BU	Customer Courses for ATS/1000 Users	Editor	85
6	BU	New Product Announcements	Editor	86
6	BU	Join an HP 1000 User Group!	Editor	89

PERFORMANCE CONCEPTS FOR SOFTWARE DESIGN AND IMPLEMENTATION

by Marc Katz/HP Data Systems Division

This paper will define some basic concepts that may be used to produce software which performs optimally. Good design can eliminate potential performance problems. The techniques outlined here should help designers produce good designs. Implementation techniques are also discussed. Although these concepts are simple, they are often overlooked, as system complexity increases or functionality changes.

LOOK AT THE SYSTEM AS A WHOLE

When a large software system is developed, the project is generally divided into parts, each part treated separately. This is the classic divide and conquer strategy. Although this strategy is often the only way for a large project to be reduced into manageable pieces, the resulting interface between these pieces can be a source of problems.

There must be coordination between the parts of the system. The design must extend to the interconnection of the parts, insuring that data is in a consistent format and that information is not lost. A balance must be maintained between the local and global view, to insure cohesiveness throughout the whole system.

MINIMIZING DATA TRANSFORMATION

One problem we have when different parts of a system are designed independent of each other is data transformation. In one graphics system, data items were transformed four times in their short lifetime. The first transformation was from the user's format to an internal format, then from internal format to a communications format, then to a database storage format, then to communication format to another graphics system!

Each transformation when viewed separately seemed justified and logical. When viewed as a whole, however, it became clear that too much transformation was occurring, and that system performance would suffer as a result. The overall system complexity grew also, since different mechanisms were needed to access and convert among the different data formats. This problem is solvable through a global look at the data flow in the system.

THE TIME/SPACE TRADEOFF

Time/space tradeoff is an underlying concept in any discussion involving performance. It could also be phrased, "use the correct algorithm for the given performance and storage constraints." In most cases, several different algorithms can be used to solve a particular problem. Each algorithm has different storage requirements and different performance properties. It should not be assumed that an algorithm which requires more space is a faster algorithm. This tends to be the case with established algorithms since algorithms which require more space and run slower generally are not popular.

An interesting example of this phenomenon can be found in the subroutine mechanism. The subroutine mechanism can be a great space saving device. Code that is common to many routines is stored in one place and is shared by routines which need it. The downside of this is that subroutine calling overhead takes time. A small subroutine could spend more execution time in the subroutine call overhead than in the body of the subroutine. In one time-critical application, we found that 35% of the execution time was spent in subroutine overhead. By changing the subroutine structure, we reduced this overhead considerably.

In this case, there was another tradeoff involved — performance vs. structure. This sticky subject must be carefully treated, to balance out the real concerns on both sides. One must be careful not to sacrifice structure and maintainability in areas which do not justify such sacrifice.

OPTIMIZE FOR THE COMMON CASE

This section could also be phrased, "worry about the things which need worry". Optimization always involves tradeoffs. Optimizing all operations for improved performance will result in programs which may be too large and too hard to understand. An analysis of the most common operations will allow the most useful optimization.

A good example of this concept is the method of optimizing instruction sets of computers. A small number of instructions commonly account for the majority of processing time. By identifying those instructions and optimizing them, the overall performance of the computer is improved. Another example is found in a graphics system. There are many different graphics instructions which must be processed. The most common, however, are moves and draws (or polylines). These are the operations which should be optimized in order to increase the overall system performance.

OPTIMIZING FOR CONSTRAINTS

The constraints of a system are the properties of inputs and outputs which are known to be restricted. We may know that some data will never be accessed or that some functionality will never be required.

For example, if a user needs a computer for one specific use, certain models will be better than others. Some models are better for real-time operations, and some are better for number crunching operations. The model is chosen according to the constraints of the environment. Optimization would not be possible if the user did not know what he would be using the computer for.

There are numerous examples of constraints which may allow more efficient design. The downside, of course, is that if constraints change, performance may suffer and large amounts of code may need to be modified.

MICRO-OPTIMIZATIONS

This is the class of optimizations which can be applied to time-intensive or often-called routines to decrease execution time. These optimizations tend to be machine-dependent and rather tricky. The following are some examples that are common:

- Reduce the use of floating point operations, where possible
- Optimize the order of comparisons so that the most likely situation is tested first
- Reduce parameter overhead by minimizing the number of parameters and the number of subroutine levels
- Optimize register usage and use of hardware and firmware features

As we have shown in this paper, these optimizations form only a small part of the overall effort. Historically, these classes of optimizatons have been well treated to the neglect of other more global optimizations.

UTILITIES FOR INCREASING PERFORMANCE

Last is the class of machine-dependent solutions for producing more optimal routines. These utilities allow access to hardware and firmware capabilities which can be used to obtain optimal performance. Some common capabilities are bit and byte manipulation instructions, move and compare instructions, hardware or firmware math and arithmetic subroutines, vector instruction sets and custom microcode. Assembly language routines can often make better use of hardware registers than can higher level languages. They also produce more optimal code.

As always, there are tradeoffs. The use of special hardware features produces programs which are more machine-dependent (and less portable) and may be hard to understand.

The use of profiling and timing routines can help identify problem areas. The use of assembly language listings of higher level language code can help to identify the possibilities for optimization, using some of the techniques above. A knowledge of the way that compilers generate code can also help in writing more efficient higher level code.

CONCLUSION

The use of all these techniques is the art and elegance of software design and implementation. A careful balance must be maintained between generality and specificity, simplicity and complexity, machine independence and dependence. These concepts can be extremely useful in making the decisions which will produce efficient systems.

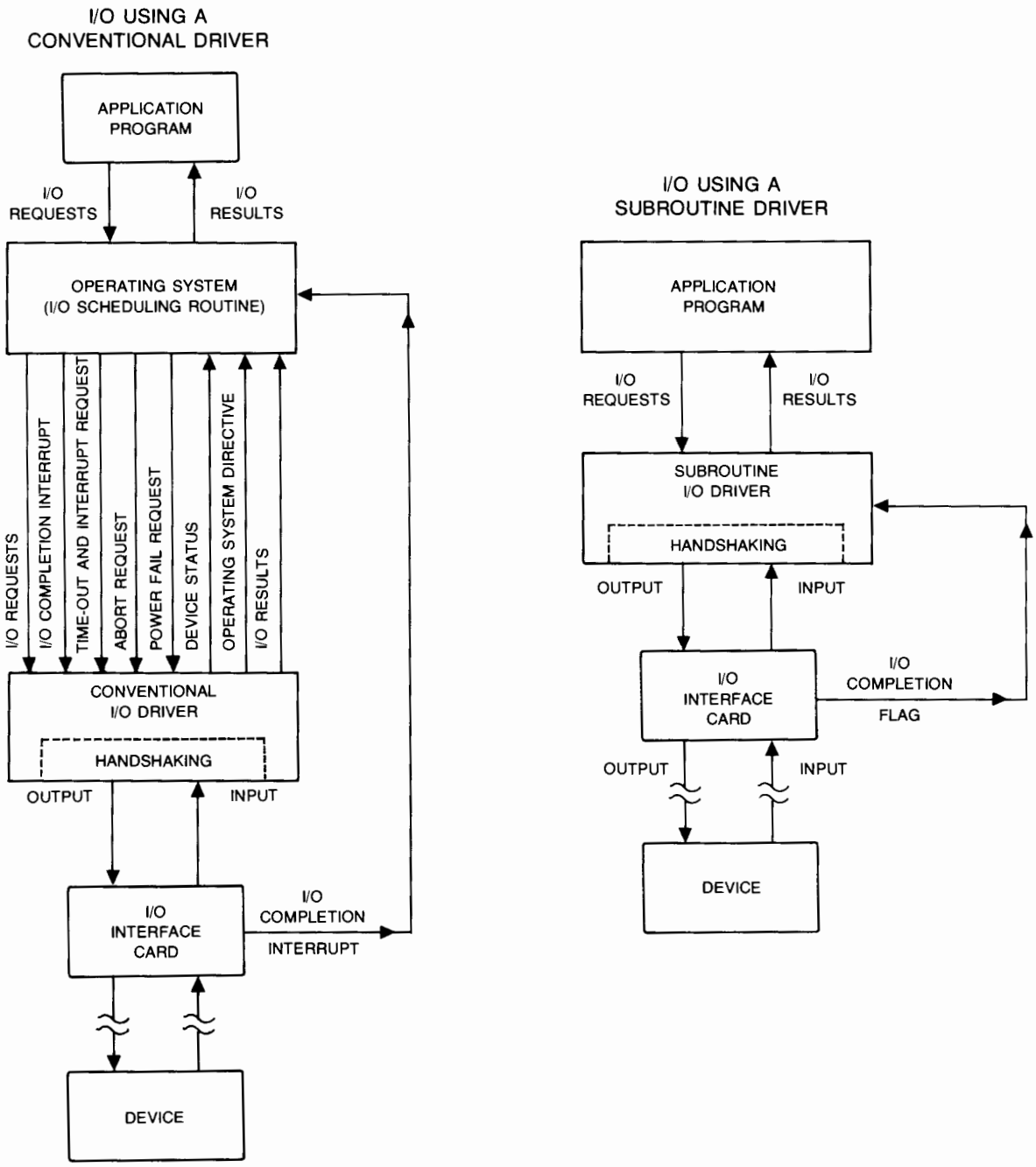


Figure 1. Information Flow Between an Application Program and a Peripheral Device

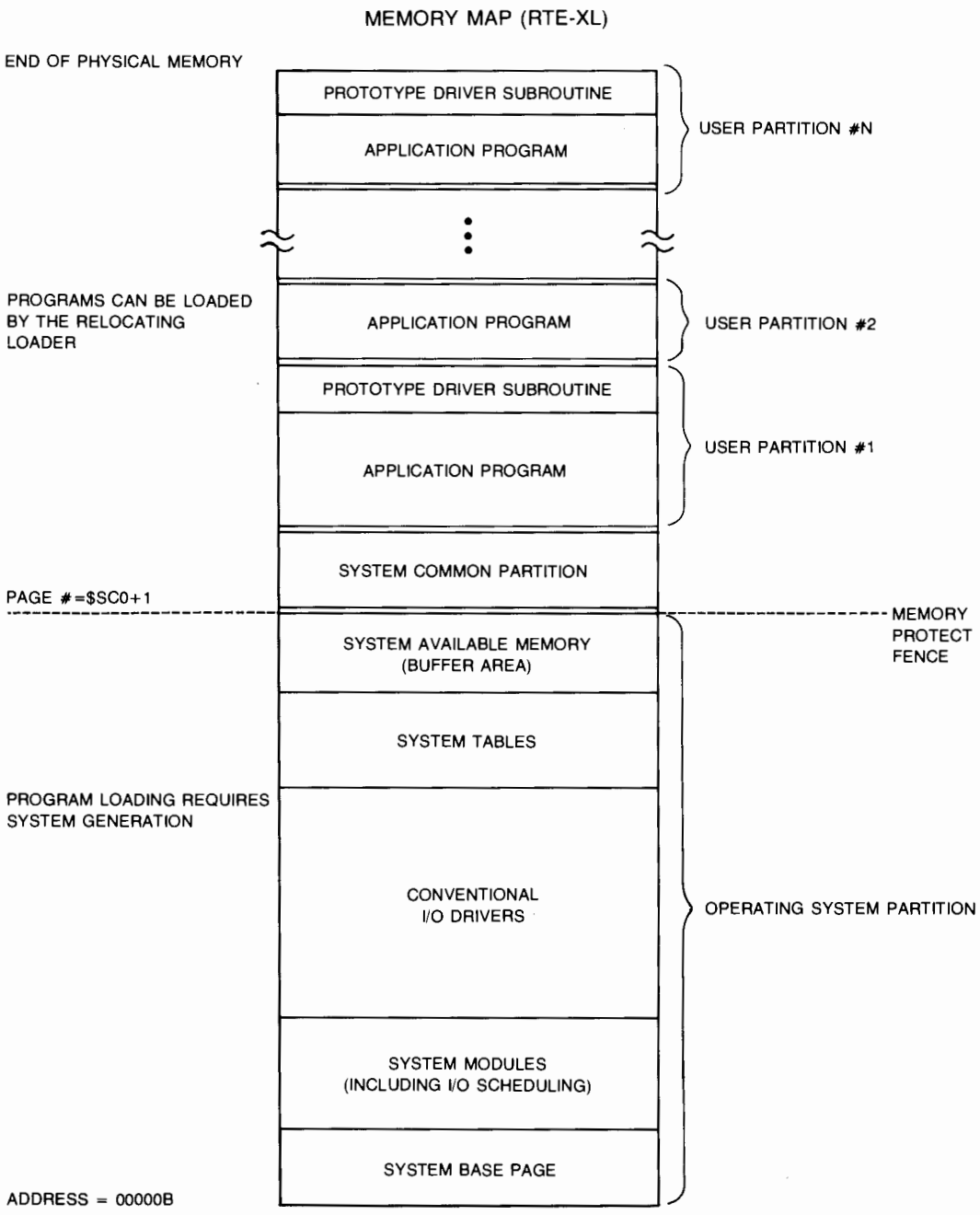


Figure 2. Relative Memory Locations of Prototype Subroutine vs Conventional Drivers

WRITING A SUBROUTINE DRIVER

A subroutine driver basically consists of four parts:

1. Parameters passed from the calling program
2. I/O transfer preparation
3. I/O transfer execution
4. System restoration

Status reporting, multi-buffered requests, control requests, etc., can be added if desired, but are not necessary in the prototype. The flowchart in Figure 3 illustrates typical subroutine driver logic.

Four parameters are required to process general I/O requests. First, the direction of the transfer must be indicated. Next, either the physical or logical unit number of the I/O device must be specified. Using the physical unit number (select code) will simplify the routine, but use of the logical unit number (LU) adds flexibility and is recommended. The I/O buffer address and length for the data transfer comprise the remaining parameters.

Preparation for the data transfer constitutes the major portion of a subroutine driver. A significant complication occurs on 21MX type computers if DMA is needed, since only two DMA channels exist. For instance, the programmer must ensure that a DMA channel is free before it is used, a function normally managed by the operating system. Therefore, a subroutine driver in an RTE-IVB system is better off not utilizing DMA for data transfers. However, in A- and L-Series computers, the distributed processing architecture provides a separate DMA channel for each I/O port. Thus, for those systems, self-configured, chained DMA is the most efficient method of data transfer, which is therefore the technique henceforth discussed.

First, the value and status of the global register must be saved (to be restored at the end of the routine). Next, determine the select code of the device and load it into the global register, then enable the global register (the global register is normally managed by the operating system prior to driver entry, but must be handled by the programmer in a subroutine driver). If the host is running RTE-XL, the relocation register corresponding to the select code must be set to the starting page number of the user partition so the I/O card can properly map into the I/O buffer in the user partition. For RTE-A.1, each select code has a complete set of mapping registers, rather than a single relocation register. In that case, the mapping register set of the desired port must be calculated (subtract 10B from the select code) and the 32 mapping registers must be copied to that I/O mapping register set using DMS instructions (i.e. MWW). Depending on which is used, either the relocation register number or the mapping register set number must be included in the lower bits of the DMA control register 21B. An operating system routine ($SETR) normally manages the relocation register and the mapping register sets, but because that routine cannot be called from a user partition, the programmer must oversee that function himself. (See the example for the method used.)

Also, for RTE-XL and -A.1, a portion of system common must be saved so it can be overwritten and subsequently restored, and pointers must be set up to direct DMA quad storage into the system common area. (See example — note that system common is accessed by the physical address instead of mapping into the user partition, since DMA register 20 must contain a physical address.) System common is used in RTE-XL and -A.1 for quad storage because the DMA self-configuration feature can access only the first 32 pages of memory, thus precluding access of user partition areas. If the target system is very large so that system common begins beyond page 32, then an eight-word storage area must be generated into the first 32 pages of the operating system with a unique entry point (i.e., $QUAD). The subroutine driver must then be altered to access that area for quad storage instead of system common. In most cases, however, the method used in the example enables execution of a subroutine driver on the existing operating system without modification.

Now the appropriate DMA quad must be built according to the direction of the I/O transfer. For RTE-XL, the relocation register number (the mapping register set number for RTE-A.1) must be logically added (inclusive "OR") to the DMA control word. Any additional characters to be transferred (i.e., a carriage return/line feed) require a contiguous DMA quad utilizing chaining of self-configured DMA. Everything is now set to begin the I/O transfer.

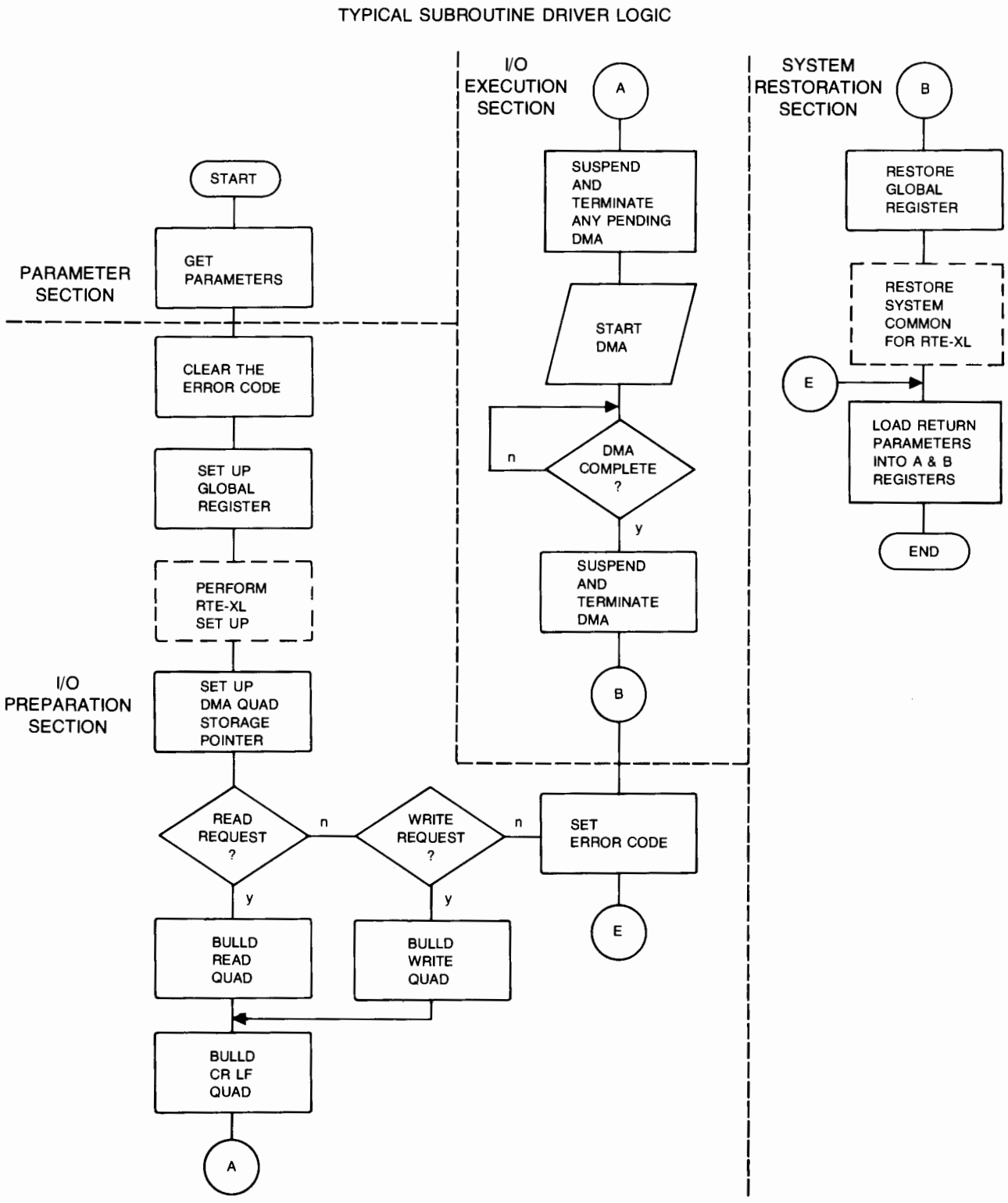


Figure 3. Example flowchart for a prototype subroutine driver. Any necessary protocol may be added to the I/O Execution Section

Communication protocol is of primary concern in the development of a custom driver and is normally an integral part of I/O transfer execution. However, since handshaking requirements are device specific and vary greatly, the topic of protocol development is beyond the scope of this discussion. Rather, attention is focused on how to locate a driver in the user partition. Therefore, the example driver included uses no handshaking so that it may be used on a variety of serial terminals for demonstration purposes.

The execution of the DMA transfer, once the protocol is satisfied, is very simple. Suspend and terminate any pending DMA by clearing the flags associated with DMA registers 21 and 23 (i.e., CLC 21B, C). Now load DMA register 20 with the physical address of the first word of the DMA quads and initiate DMA with the STC 20B,C instruction. Wait for the register 20 flag to be set, indicating completion of the DMA chain. Although interrupt processing is conceivable in a subroutine, the procedure gets rather involved and emulation of this particular operating system function is not practical in a prototype. Upon DMA completion, suspend and terminate DMA as before and set the I/O card to a known state. (See example.) Finally, the number of bytes actually transferred may be calculated if desired.

The last step consists of returning the system to the condition it was in prior to subroutine entry. This task is difficult for RTE-IVB systems because system time is lost for the duration of privileged execution, so rebooting may be a simpler solution. For A- and L-Series systems, up to 2^{16} 10-millisecond TBG tics (about 10.92 minutes) are held during privileged execution to be added to the system clock automatically (by RTIOL) on subroutine exit, thus maintaining time management and facilitating system restoration. First, restore the contents of the global register, then enable or disable the register according to its previous state. For RTE-XL and -A.1, the contents of system common that were overwritten must be restored. The subroutine is now complete.

Although some routines are required in a subroutine driver that are not necessary in a conventional driver (operating system emulations such as global register management), these routines are straightforward and easy to write. The subroutine prototype remains simpler than a conventional driver, in addition to all the other advantages heretofore discussed.

AN EXAMPLE OF A SUBROUTINE DRIVER

Included here is an example subroutine driver that performs simple input and output to a serial RS-232 device using an asynchronous serial interface card (HP 12005A). The parameters are compatible with EXEC 1 and 2 call formats. For example:

```
CALL DRIVR (CODE,CNTWD,BUFR,LEN)
where:
CODE  = an integer 1 or 2 for read or write reqests respectively
CNTWD = the integer LU number
BUFR  = an integer buffer for the transfer
LEN   = the positive number of words or negative number of bytes
        for the transfer.
```

On return from subroutine, the A register contains an error code and the B register contains the positive number of bytes or words transferred (according to the sign of LEN in the EXEC request). The driver is compatible with both RTE-L and -XL. This is accomplished by determining which system is running (compare an XL entry point with 0). If the system is RTE-L, NOP's are written over the JSB's to the XL setup and restoration routines. In either case, a NOP is written over the JSB to the test routine so the system will only be checked the first time the subroutine executes. If the target system is known, this function may be performed by changing the code and removing unnecessary routines. If the subroutine is to be loaded without change on an RTE-L system, a dummy library must be searched to satisfy external references unique to RTE-L. (See example dummy library included in the listing that follows.) For RTE-XL, even though system common is used, it should only be specified to the loader if required by the main program, since the subroutine accesses system common by its physical address. Application programs using both system common and the subroutine driver are permitted since the driver restores all system common that it uses.

Note the use of system entry points in determining the select code from a logical unit number, calculating the relocation register number, determining the partition starting page by use of the ID segment and partition table, and in the use of system common. This enables the subroutine driver to run on nearly any system configuration for RTE-L and -XL.

A simple FORTRAN program to demonstrate the example driver is as follows:

```
FTN4,L
      PROGRAM DEMO(), DEMONSTRATE THE EXAMPLE SUBROUTINE DRIVER
      DIMENSION IBUFR (40)
      LU=LOGLU (IDUMY)
C Prompt the user for input
      CALL DRIVR (2,LU,15HEnter some data,-15)
C Receive user input
      CALL DRIVR (1,LU,IBUFR,-80)
C The B register contains the positive number of bytes read
      CALL ABREG (IA,IB)
C Prompt the return data
      CALL DRIVR (2,LU,25HThis is what was entered:,-25)
C Send the data to the user terminal
      CALL DRIVR (2,LU,IBUFR,-IB)
C Now demonstrate formatted I/O
      CALL DRIVR (2,LU,34HNow enter an integer less than 180,-34)
C Receive input from the terminal
      CALL DRIVR (1,LU,IBUFR,-80)
      CALL ABREG (IA,IB)
C Use the formatter to interpret the ASCII number received
      CALL CODE(IB)
      READ (IBUFR,*) I
C Perform a calculation to verify proper formatting
      J=I*I
C Format the output buffer into ASCII data
      CALL CODE(20)
      WRITE (IBUFR,10) I,J
10    FORMAT (I4," squared=",I6)
C Output the formatted buffer to the terminal
      CALL DRIVR (2,LU,IBUFR,-20)
      END
```

This program requests input from the user's terminal and displays what the user sends, utilizing only the subroutine driver for I/O. It then demonstrates how to perform formatted I/O using the prototype.

In conclusion, by using the example subroutine driver prototype as a skeleton, one can develop a custom subroutine driver with minimal effort. The author has used this technique to set up a communication link between an HP 1000-XL and a Z-80 based microcomputer, thus even complex protocol requirements may be satisfied with increased programming efficiency using the prototype method herein described.

EXAMPLE PROTOTYPE DRIVER SUBROUTINE

```
EXAMPLE PROTOTYPE DRIVER SUBROUTINE

ASMB,L,Q,R
      HED Example Subroutine Driver for RTE-L and -XL Systems
**************************************************************************
**                    Subroutine Terminal Driver                        **
**    This is an example subroutine driver to execute simple            **
**  input and output to an RS232 device using an Asynchronous           **
**  Serial Interface Card (ASIC HP 12005A).  The parameters follow       **
**  the format of EXEC 1 and 2 calls for compatability: simply           **
**  replace the name EXEC with DRIVR.  This routine tests whether        **
**  it is being run on an RTE-L or -XL system the first time it is       **
**  executed and changes the machine code as necessary, so it may        **
**  be run as is on either system.  The driver performs the function     **
**  of both device and interface driver, though it is possible to        **
**  separate the two.                                                    **
**    The A and B register returns are the same as with EXEC calls:      **
**  the A reg contains an error/status code (0 if successful) and        **
**  the B reg contains the positive number of words or bytes             **
**  transferred (depending on the sign of the length parameter as        **
**  with EXEC calls).                                                    **
**************************************************************************
      SPC 2
      NAM DRIVR,7,30  Subroutine Terminal Driver
      ENT DRIVR
      EXT $LIBR,.ENTP,$LIBX,$LUTA,.XLA,.XLB,.XSB,$SC
      SPC 1
***   The following externals are only used by the RTE-XL routines.
***   If the target system is RTE-L, simply search the dummy library
***   listed following this routine.
      SPC 1
      EXT $SCO,.MWF,.MWI,$MATA
      EXT $MASZ,$BASE,.LDX
      SPC 1
**************************************************************************
**               Parameters Passed from Calling Program                 **
**************************************************************************
      SPC 1
CODE  NOP
CNTWD NOP
BUFR  NOP
LEN   NOP
      SPC 1
DRIVR NOP           ENTRY POINT FOR THIS ROUTINE
      JSB $LIBR     LEVEL 1 SUBROUTINE ENTRY
      NOP           TURN OFF INTERRUPTS & MEM PROTECT
      JSB .ENTP     PICK UP PARAMETERS
      DEF CODE      AND PUT THEM HERE
      SPC 1
**************************************************************************
**                       I/O Transfer Preparation                       **
**************************************************************************
      SPC 1
      LDA LEN,I     GET LENGTH OF BUFFER
      SZA,RSS       ZERO LENGTH TRANSFER?
      JMP EXIT      YES, DON'T DO ANYTHING
```

Computer Museum

```
***
*** The following JSB alters the code for RTE-L & -XL compatability
***
FIXIT JSB LXLFX      SET UP DRIVR FOR L OR XL SYSTEM
      SPC 1
      CLB
      STB ERR        START W/ NO ERRORS
      JSB GLBST      SET UP GLOBAL REG
      LDA CODE,I     DETERMINE DIRECTION OF TRANSFER
      CPA B1         READ?
      JMP READ       YES, READ DATA
      CPA B2         WRITE?
      JMP WRITEI     YES, WRITE DATA
      LDA D56        NO,ERR=56 (BAD PARAMETER)
      STA ERR        STORE ERROR CODE
      JMP QUIT       RETURN
READ  LDA RELRG      GET RELOCATION REGISTER
      IOR RCTWD      PUT REL REG IN DMA CONTROL
      STA *+2        STORE DMA CONTROL FOR QUAD
      JSB QUAD       BUILD READ QUAD (RETURN *+5)
      NOP            DMA CONTROL WORD
      DEF RASIC      ASIC CONTROL WORD
      DEF BUFR       BUFFER ADDRESS
      DEF LEN,I      BUFFER LENGTH
      LDA DMAAD      GET DMA POINTER
      STA RDUE       STORE AS RESIDUE POINTER
      JMP SDMA       START DMA
      SPC 2
WRITE LDA RELRG      GET RELOCATION REGISTER
   C  IOR WCTWD      PUT REL REG IN DMA CONTROL
      STA *+2        STORE CONTROL FOR QUAD
      JSB QUAD       BUILD WRITE QUAD (RETURN *+5)
      NOP            DMA CONTROL WORD
      DEF WASIC      ASIC CONTROL WORD
      DEF BUFR       BUFFER ADDRESS
      DEF LEN,I      BUFFER LENGTH
      LDA DMAAD      GET DMA POINTER
      STA RDUE       STORE AS RESIDUE POINTER
      SPC 2
SDMA  LDA RELRG      GET RELOCATION REGISTER
      IOR CRWD       PUT IN CRLF DMA CONTROL WORD
      STA *+2        STORE IT
      JSB QUAD       BUILD CRLF QUAD
      NOP            CRLF DMA CONTROL WORD
      DEF WASIC      ASIC WRITE CONTROL WORD
      DEF CRLFA      CARRIDGE RETURN / LINE FEED
      DEF DN2        TWO BYTE OUTPUT
      SPC 1
*****************************************************************
**                    I/O Execution Section                    **
*****************************************************************
      SPC 1
      CLC 21B,C      SUSPEND AND
      CLC 23B,C      TERMINATE DMA
```

```
***
***   This is where any communicatior protocol required must be satisfied
***   before continuing.  In this example, no handshaking is used so as to
***   not restrict the devices on which this driver may be demonstrated.
***
      LDB QUADA      GET QUAD STARTING ADDRESS
      OTB 20B        PUT IN I/O CARD REG
      STC 20B,C      START DMA
***
***   For a conventional driver, an interrupt would signal DMA completion.
***   For simplity in the prototype driver, we will wait for completion.
***
      SFS 20B        DMA COMPLETE?
      JMP *-1        NO, CHECK AGAIN
      SPC 1
      CLC 21B,C      SUSPEND AND
      CLC 23B,C      TERMINATE DMA
      CLB
      OTB 31B        CLEAR CARD CONTROL
      OTB 21B        CLEAR DMA CONTROL
      STC 30B,C      ENABLE BREAK
      JSB .XLB       GET RESIDUE
      DEF RDUE,I     FROM QUAD STORAGE AREA
      ADB DN1        SUBTRACT TERMINATION CHARACTER
      LDA LEN,I      GET ORIGINAL BUFFER LENGTH
      SSA            NEGATIVE?
      JMP SDMA1      YES, MAKE POSITIVE & ADD TO RDUE
      ALS            NO, DOUBLE FOR # BYTES
      JMP SDMA1+1    ADD TO RESIDUE
      SPC 1
SDMA1 CMA,INA        MAKE LENGTH POSITIVE
      ADA B          CALCULATE # OF BYTES IN XFER
      STA L          STORE IT
      LDB LEN,I      GET ORIGINAL BUFFER LENGTH
      SSB            BYTE MODE?
      JMP QUIT       YES, QUIT
      CLE,ERA        NO, DIVIDE BY TWO
      SEZ            AND ROUND
      INA            UPWARD
      STA L          STORE IT
      SPC 1
*****************************************************************************
**                    System Restoration Section                           **
*****************************************************************************
      SPC 1
QUIT  JSB GLBRT      RESET GLOBAL REG
      LDA ERR        GET ERROR CODE FOR RETURN
      LDB L          GET LENGTH OF XFER
EXIT  JSB $LIBX      END THE ROUTINE
      DEF DRIVR
      SPC 2
```

```
*************************************************************************
**  The routines that follow are for setting up the system and I/O     **
**  card for the DMA transfers                                         **
*************************************************************************
      SPC 2
GLBST NOP             SET UP THE GLOBAL REGISTER
      LIB 2B          GET CURRENT GLOBAL SETTING
      STB GLOBL       STORE IT
      CLA
      SFC 2B          GET THE FLAG STATUS
      INA
      STA GFLG        STORE IT
      STF 2B          DISABLE GLOBAL REG
      JSB LUSC        GET SELECT CODE
      OTA 2B          PUT IT IN GLOBAL REG
      CLF 2B          ENABLE GLOBAL
***
***   The following JSB is only for RTE-XL set-up
***
JXL1  JSB XL1         RTE-XL SET UP
      SPC 1
      LDB QUADA       GET ADDR FOR QUAD STORAGE
      ADB DN1         SUBTRACT 1
      STB DMAAD       STORE IN DMA POINTER
      JMP GLBST,I     RETURN
      SPC 2
GLBRT NOP             RESTORE GLOBAL REGISTER
      LDB GLOBL       GET ORIGINAL GLOBAL
      STF 2B          DISABLE GLOBAL
      OTB 2B          PUT ORIGINAL VALUE IN GLOBAL
      LDA GFLG        GET GLOBAL FLAG
      SLA,RSS         ENABLED?
      CLF 2B          YES, RE-ENABLE
***
***    The following JSB is only for RTE-XL housekeeping
***   (may be deleted for RTE-L)
***
JXL2  JSB XL2         RTE-XL CLEAN UP
      SPC 1
      JMP GLBRT,I     RETURN
      SPC 2
```

```
***
***   This routine finds the select code given a logical unit number.
***   By using this routine, the operator is able to redirect the
***   I/O with the LA command.
***
LUSC  NOP            FIND THE SELECT CODE
      LDA CNTWD,I    GET CONTROL WORD
      AND B77        GET LU #
      ADA DN1        SUBTRACT 1
      JSB .XLB       GET ADDRESS OF
      DEF $LUTA      LU TABLE
      ADB A          ADE DISPLACEMENT FOR LU
      JSB .XLA       GET DVT ADDR
      DEF B,I        FOR LU
      ADA D4         SET FOR IFT ADDR  (DVT5)
      JSB .XLB       GET IFT ADDR
      DEF A,I
      ADB D5         SET FOR IFT6
      JSB .XLA       GET IFT6
      DEF B,I
      AND B77        TAKE SELECT CODE ONLY
      STA SC         STORE IT
      JMP LUSC,I     RETURN (SELECT CODE IS IN A REGISTER)
      SPC 2
***
***   The next two routines are modified versions of routines from
***   ID.00.  They build contiguous quads for self-configured DMA into
***   either system common or a user area depending on how the DMAAD
***   pointer is set up.
***
QUAD  NOP            BUILD DMA QUADS
      LDA DMAAD      GET DMA POINTER
      INA            INCREMENT POINTER
      LDB QUAD,I     GET DMA CONTROL WORD
      JSB .XSB       STORE IT
      DEF A,I
      JSB NEXT       ASIC CONTROL WORD
      JSB NEXT       BUFFER ADDRESS
      JSB NEXT       BUFFER LENGTH
      ISZ QUAD       FIX RETURN ADDRESS
      STA DMAAD      FIX DMA POINTER
      SSB            CHARACTERS?
      JMP QUAD,I     YES, QUAD COMPLETE
      BLS            NO, SAVE
      CMB,INB        BUFFER LENGTH
      JSB .XSB_      IN CHARACTERS
      DEF A,I
      JMP QUAD,I     RETURN
      SPC 2
NEXT  NOP            SET UP A WORD IN THE QUAD
      INA            INCREMENT ADDR POINTER
      ISZ QUAD       INCREMENT POINTER TO QUAD WORD
      LDB QUAD       GET ADDR OF PTINTER TO NEXT WORD
      LDB B,I        GET POINTER TO NEXT WORD
      RBL,CLE,SLB,ERB   INDIRECT ADDR?
      JMP *-2        YES, INTERPRET AS INDIRECT POINTER
      LDB B,I        NO, GET QUAD WORD
      JSB .XSB       STORE IN QUAD
      DEF A,I
      JMP NEXT,I     RETURN
      SPC 2
```

```
***
***   This routine adjusts the machine code for RTE-L or -XL execution
***   (may be omitted if the target system is known and appropriate
***   changes are made by hand to the source code)
***
LXLFX NOP             FIX DRIVR FOR L OR XL SYSTEM
      CLA             THIS ROUTINE WILL BE BRANCHED TO ONCE
      STA FIXIT       CHANGE FIXIT TO A NOP  (WAS "JSB LXLFX")
      JSB .XLB        GET POINTER TO
      DEF $SC0        XL SYSTEM COMMON (RTE-L DUMMY LIBRARY)
      SZB             RTE-XL SYSTEM? (RTE-L RETURNS 0)
      JMP LXLFX,I     YES, NO PROGRAM CHANGES
      STA JXL1        NO, DON'T SET UP FOR RTE-XL
      STA JXL2        DON'T HOUSEKEEP FOR RTE-XL
      JMP LXLFX,I     RETURN
      SKP
**************************************************************************
**     The following group of routines takes care of setting up         **
**  for an RTE-XL system.  This consists of setting the proper          **
**  relocation register to the starting page of the user partition,     **
**  saving a portion of system common, setting up the quads to be       **
**  placed into system common, and restoring the used portion of        **
**  system common at the end of the routine.  None of this is           **
**  necessary in RTE-L.                                                 **
**************************************************************************
      SPC 2
XL1   NOP             RTE-XL SET UP
      JSB .XLA        GET POINTER TO
      DEF $SC0        SYSTEM COMMON
      ADA B1          ADD 1 FOR BEGINNING PAGE #
      LSL 10          CHANGE PAGE # TO PHYSICAL ADDR
      STA QUADA       STORE ADDRESS
      LDB SCHA        GET ADDR OF HOLD AREA
      JSB .LDX        LOAD COUNTER IN X REG
      DEF D8          (COUNTER = 8)
      JSB .MWF        SAVE SYCOM AREA TO BE OVERLAID
      JSB SETR        SET UP RELOCATION REGISTER
      JMP XL1,I       RETURN
      SPC 2
XL2   NOP             RTE-XL HOUSEKEEPING
      LDA SCHA        ADDR OF SYSCOM HOLD AREA
      LDB QUADA       ADDR OF SYSCOM OVERLAID
      JSB .LDX        LOAD COUNTER IN X REG
      DEF D8          (COUNTER=8)
      JSB .MWI        RESTORE CONTENTS OF SYSCOM
      JMP XL2,I       RETURN
      SPC 2
SETR  NOP             SET RELOCATION REGISTER
      JSB .XLA        GET IDSEG ADDR
      DEF $SC         OF CURRENT PROGRAM
      ADA D25         INCREMENT TO 26TH WORD
      JSB .XLA        GET CONTENTS OF
      DEF A,I         26TH WORD OF IDSEG
      AND B377        GET PARTITION #
      ADA DN1         SUBTRACT 1
      JSB .XLB        FIND LENGTH OF EACH
      DEF $MASZ       PARTN TABLE ENTRY
      MPY B           FIND DISPLACEMENT TO PRGM PARTN
      JSB .XLB        GET ADDR OF
      DEF $MATA       PARTITION TABLE
      ADA B           ADD PRGM DISPLACEMENT
```

```
       ADA B2          INCREMENT TO PARTN PAGE # ENTRY
       JSB .XLB        GET PARTN STARTING
       DEF A,I         PAGE #
       JSB .XLA        GET ADDR OF
       DEF $BASE       RELOCATION REGS
       ADA DN15        SUBTRACT 15 FOR ADDING SELECT CODE
       ADA SC          ADD SELECT CODE
       JSB .XSB        STORE PARTN PAGE # INTO
       DEF A,I         RELOCATION REGISTER
       STA RELRG       STORE REL REG #
       JMP SETR,I      RETURN
       SPC 2
*******************************************************************************
**                          Symbol Definitions                               **
*******************************************************************************
       SPC 2
A      EQU 0           A REGISTER
B      EQU 1           B REGISTER
B1     OCT 1
B2     OCT 2
B77    OCT 77
B377   OCT 377
DN15   DEC -15
DN2    DEC -2
DN1    DEC -1
D4     DEC 4
D5     DEC 5
D8     DEC 8
D25    DEC 25
D56    DEC 56
CRLF   OCT 6412        <CR><LF>
CRLFA  DEF CRLF        ADDRESS OF CRLF
CRWD   OCT 65400       CRLF DMA CONTROL WORD
DMAAD  NOP             DMA QUAD POINTER
ERR    NOP             ERROR CODE
GFLG   NOP             GLOBAL FLAG
GLOBL  NOP             GLOBAL HOLD AREA
L      NOP             LENGTH HOLD AREA
QUADA  DEF QUADS       ADDRESS OF QUADS
QUADS  BSS 8           QUAD STORAGE AREA FOR RTE-L
RASIC  OCT 46000       READ ASIC CONTROL WORD
RCTWD  OCT 175200      READ DMA CONTROL WORD
RDUE   NOP             RESIDUE ADDRESS
RELRG  NOP             RELOCATION REGISTER #
SC     NOP             SELECT CODE
SCHA   DEF QUADS       ADDRESS OF SYSCOM HOLD AREA
WASIC  OCT 1000        WRITE ASIC CONTROL WORD
WCTWD  OCT 175400      WRITE DMA CONTROL WORD
       SPC 2
       END DRIVR
```

DUMMY LIBRARY FOR RTE-L

```
ASMB,L,Q,R
      HED Dummy Library to Satisfy RTE-XL externals on an RTE-L System
      SPC 1
**************************************************************************
**                        RTE-L Dummy Library                           **
**                                                                      **
**     This is a dummy library to satisfy the externals used in the     **
**   subroutine DRIVR unique to RTE-XL.  This library must be           **
**   searched when loading DRIVR for execution on an RTE-L system       **
**   (i.e. SEA,%..XLB) unless the external references are removed       **
**   from the DRIVR source code.                                        **
**************************************************************************
      SPC 1
      NAM ..XLB,7  Dummy library to satisfy XL externals
      SPC 1
      ENT $BASE,$MASZ,$MATA,$SC0
      SPC 1
$BASE NOP           POINTER TO THE RELOCATION REGISTERS
$MASZ NOP           SIZE OF ENTRY IN THE PARTITION TABLE
$MATA NOP           ADDRESS OF THE PARTITION TABLE
$SC0  NOP           PAGE # PRECEDING SYSTEM COMMON
      SPC 1
      END
```

TRICKS WITH EDIT/1000

by Michael Wiesenberg/HP Data Systems Division

EDIT/1000 is as powerful as almost any word processor on the market (particularly when regular expressions are used), but many people do not use all its capabilities because the manual does not present as many examples as it might. Perhaps this is not a fault in the documentation, because there are so many things you can do, that no one manual could cover them all. Here are some tricks you can do with EDIT/1000.

COLUMN EXCHANGES

Here's how to exchange any number of columns in any order, with a one-line command.

For example, you want to exchange columns 1 and 3 in the following (a "rule line" is also displayed, so that you can readily see where the columns begin and end; use the HL command to get a rule line). Here's what you start with:

```
1111  11  111   1111        222222222222222222 333333
1                           2      2           333 333333
1  1    1        1  1       22222222 2 2 2     3
1  1    1        1  1       22222222 2 2 2     3
11111111111111              222222222222
11111                       2222   22222  22   333333333
111111111                   222                33  33
111111                      2  2   2  2  2 2   33  3333
1111111111                  22222222           3333 333

'''',''''1'''',''''2'''',''''3'''',''''4'''',''''5'''',''''6'''',''''7'
```

Here's what you want to end with:

```
333333                      222222222222222222 1111  11  111    1111
333 333333                  2      2           1
3                           22222222 2 2 2     1  1    1        1  1
3                           22222222 2 2 2     1  1    1        1  1
                            222222222222       11111111111  111
333333333                   2222   22222  22   11111
33  33                      222                111111111
33  3333                    2  2   2  2  2 2   111111
3333 333                    22222222           1111111111
```

First, pad the last column with spaces such that the spacing extends beyond the longest item in that column. Put in as much spacing as you want that column to have when it is moved. If you want the spacing to line up the same as before making the switch, extend the column as far as the spacing of the first.

The first field begins at 1 and ends at 23; the second at 24 and 42; and the third begins at 43, with its widest element ending at 52. To switch fields 1 and 3 such that 3 occupies the same space as 1, first make 3 as wide as 1 with the U command:

```
sewc 65
1$u// //
sewc
```

(If you use the SE WC command without specifying parameters, the action is to revert to the default condition. That is, SEWC is the same as SEWC 1 150.) The first field is 23 "ticks" wide. To make the third the same, it had to be extended to tick 65. What the U command does is make an unconditional exchange. Anything at tick 65 is exchanged for a space. If the end of the line is shorter than that, the intervening area is padded with spaces. (To verify that EDIT does this, first list, in line mode, the file, with DISPLAY FUNCTIONS on. Notice that the carriage returns all come at the end of a line, or one space after. (EDIT fills an odd-numbered byte at the end of a line with a space.) Turn DISPLAY FUNCTIONS off, issue the three commands, and again list the lines with DISPLAY FUNCTIONS on. Notice that the carriage returns are all neatly lined up.)

Now use the "." to represent any character, and "<n>" to repeat the specification. That is, ".<23>" means any 23 consecutive characters. Tag the three fields to be recalled. "{.<23>}" means a field consisting of any 23 characters. The braces ("{}") mean that we will recall this field later. And this command, "1$x/{.<23>}{.<19>}{.<23>}/&3&2&1//", means, in each line let's label the first 23 characters Field 1, the next 19 Field 2, and the following 23 Field 3. Let's bring them back again in this order: first Field 3, then Field 2 then Field 1.

Here's the entire move in a one-line command:

```
sere|sewc65|1$u// //|sewc|1$x/{.<23>}{.<19>}{.<23>}/&3&2&1//
```

Let's say you want the third field to be only as wide as it is in the original, with, perhaps, two extra spaces so the columns don't run together. The command looks like this:

```
sereon|sewc54|1$u// //|sewc|1$x/{.<23>}{.<19>}{.<12>}/&3&2&1//
```

You could add a space between the second and third fields, if you wish, this way:

```
sereon|sewc54|1$u// //|sewc|1$x/{.<23>}{.<19>}{.<12>}/&3&2&1//
```

Don't repeat the "SEREON" command each time. Just "turn on" regular expressions once, and leave them on for that editing session. (If you don't know the state of regular expressions, type "SHRE.") The "SERE" command has no default; it toggles back and forth between on and off. If you don't know the "state" of regular expressions (whether turned on or off), and you want to turn them on without the bother of issuing the "SHRE" command (and you want to make sure that you don't accidentally turn them off), use "SEREON." If you **know** they are not turned on, you can use the short form, "SHRE."

This sort of procedure can be extended to more columns (or done with two). Just remember to pad the last column with spaces. If the last column is not moved, you can omit this padding

MIXING COMMANDS

You could make exchanges like the preceding on noncontiguous lines. If the lines come at regular intervals, you do it one way; if the lines to be exchanged share a certain common characteristic, they don't even have to be regularly spaced.

For example, you wish to make an exchange on every third line. The original could look like this:

```
11111111111111111111  2222222222222222  33333333333333333333
11  11111111111111  11  222222          222  33 3333333333333333 33
22222 2222222222222  33 3333333333333333 33  11 1 111111111111 1 11
11 11 1111111111 11 11  2222 22222222222222  33 3333333333333333 33
11 111 11111111 111 11  222 222222222222222  33 3333333  3333333 33
22 2222222222222222  33 333333 33 333333 33  11 1111 111111 1111 11
11 11111 1111 11111 11  22 2222222222222222  33 33333 3333 33333 33
11 111111 11 111111 11  22 2222222222222222  33 3333 333333 3333 33
22 22222222      22  33 333 33333333 333 33  11 1111111  1111111 11
11 1111111111111111 11  22 222222222222 222  33 33 3333333333 33 33
11 1111111111111111 11  222 2222222222 2222  33 3 333333333333 3 33
2222          22222  33  33333333333333  33  11 1111111111111111 11
11111111111111111111  2222222222222222  33333333333333333333
'''''/''''1''''/''''2''''/''''3''''/''''4''''/''''5''''/''''6''''/''''7'
```

You want to move all the 1s into the first column, the 2s into the second, and the 3s into the third. Notice that only every third line is out of place. First turn on regular expressions (SEREON). Mark the first and last lines (control-K and type a letter after the colon that appears near the right margin of the screen). Next, add two spaces to the end of every line. Then, use the repeat command (an underscore followed by a number as the last element of the command).

```
sereon
:a:bg/{@}$/&1 //
+3|g/{.<21>}{.<24>}{.<24>}/&3&1&2//|_3
```

Voila!

```
11111111111111111111  2222222222222222  33333333333333333333
11  11111111111111  11  222222          222  33 3333333333333333 33
11 1 111111111111 1 11  22222 2222222222222  33 3333333333333333 33
11 11 1111111111 11 11  2222 22222222222222  33 3333333333333333 33
11 111 11111111 111 11  222 222222222222222  33 3333333  3333333 33
11 1111 111111 1111 11  22 2222222222222222  33 333333 33 333333 33
11 11111 1111 11111 11  22 2222222222222222  33 33333 3333 33333 33
11 111111 11 111111 11  22 2222222222222222  33 3333 333333 3333 33
11 1111111  1111111 11  22 22222222      22  33 333 33333333 333 33
11 1111111111111111 11  22 222222222222 222  33 33 3333333333 33 33
11 1111111111111111 11  222 2222222222 2222  33 3 333333333333 3 33
11 1111111111111111 11  2222          22222  33  33333333333333  33
11111111111111111111  2222222222222222  33333333333333333333
```

If the affected lines are not evenly spaced, you use a different approach. This is what you have:

```
2222222222222222  33333333333333333333  11111111111111111111
11  11111111111111  11  222222          222  33 3333333333333333 33
22222 2222222222222  33 3333333333333333 33  11 1 111111111111 1 11
11 11 1111111111 11 11  2222 22222222222222  33 3333333333333333 33
11 111 11111111 111 11  222 222222222222222  33 3333333  3333333 33
22 2222222222222222  33 333333 33 333333 33  11 1111 111111 1111 11
22 2222222222222222  33 33333 3333 33333 33  11 11111 1111 11111 11
11 111111 11 111111 11  22 2222222222222222  33 3333 333333 3333 33
11 1111111  1111111 11  22 22222222      22  33 333 33333333 333 33
22 222222222222 222  33 33 3333333333 33 33  11 1111111111111111 11
11 1111111111111111 11  222 2222222222 2222  33 3 333333333333 3 33
11 1111111111111111 11  2222          22222  33  33333333333333  33
2222222222222222  33333333333333333333  11111111111111111111
```

Issue this set of commands:

```
:a:bg/{@}$/&1  //
:a-1
f/^2/!x/{.<21>}{.<24>}{.<24>}/&3&1&2//!_5
```

CREATING A COMMAND FILE

A command file provides a convenient way of executing several edit commands, such as executing one or more commands upon several files. You can let EDIT open each file, execute the commands, and then write the file back to disk, without having to oversee the operation. You can even use EDIT's capabilities **to generate the contents of the command file**.

For example, to produce the list of files, use the RU command. Suppose you were writing a language manual. The names of all the files of the manual would probably be related in some way. Say you were writing a FORTRAN manual. You might have named all the files FTN.x, where x is a number for a chapter or a letter for an appendix. Suppose you had used the term "FORTRAN 77" everywhere, and the marketing department decided to issue the language under the name FORTRAN/9000. Within each file in the manual set, you wish to execute this command:

```
1$g/FORTRAN 77/FORTRAN\/9000//
```

(Notice that the slash that is part of the name "FORTRAN/9000" must be preceded by the escape character ("\") so that EDIT does not interpret it as the terminator for the field to be exchanged. That is, if you used this command:

```
1$g/FORTRAN 77/FORTRAN/9000//
```

You would get this response:

```
1$g/FORTRAN 77/FORTRAN/9000//
/1$g/FORTRAN 77/FORTRAN/9000//
?                          ^
/
```

EDIT questions the slash because it thinks the slash is part of the command. You must use the escape character — that back slash — to tell EDIT that you mean to use the slash literally.)

You could call up each file individually, issue the command, write the file back to the disc, and call up the next file. But why not let the computer do all the work? Get a list of appropriate files by running FMGR, and then using the DL command:

```
/RU,FMGR
:DL,FTN.--::AA
```

(Specify the cartridge or you may get a longer list than you want.)

That produces something like this on your screen:

```
CR= AA
 ILAB=FTNMNL NXTR= 00169 NXSEC=118 #SEC/TR=128 LAST TR=00227 #DR TR=0

NAME   TYPE   SIZE/LU    OPEN TO

FTN.1  00004 00024 BLKS
FTN.1  00004 00024 BLKS +001
FTN.1  00004 00024 BLKS +002
FTN.2  00004 00024 BLKS
FTN.2  00004 00024 BLKS +001
FTN.2  00004 00024 BLKS +002
FTN.2  00004 00024 BLKS +003
FTN.2  00004 00024 BLKS +004
FTN.2  00004 00024 BLKS +005
FTN.2  00004 00024 BLKS +006
FTN.2  00004 00024 BLKS +007
FTN.2  00004 00024 BLKS +008
FTN.2  00004 00024 BLKS +009
FTN.2  00004 00024 BLKS +010
FTN.2  00004 00024 BLKS +011
FTN.2  00004 00024 BLKS +012
FTN.3  00004 00024 BLKS
FTN.3  00004 00024 BLKS +001
FTN.3  00004 00024 BLKS +002
FTN.3  00004 00024 BLKS +003
FTN.3  00004 00024 BLKS +004
FTN.3  00004 00024 BLKS +005
FTN.3  00004 00024 BLKS +006
FTN.3  00004 00024 BLKS +007
.
.
.
:EX
Resume EDI.B on CMDFIL:Q:Q:4
```

Use the DELETE LINE key to remove all lines from your screen that are not part of the list of files (including the heading lines). Use the SC (copy screen) command to copy the contents of the screen into your file. Eliminate all the extents, as shown in the **EDIT/1000 User's Guide** ("Create Procedure Files from Disc Directory Listing"), using the command this way:

```
D/\+[0-9]* *$/AVQ/
```

You should now have a list of lines, each of which begins with the characters "FTN." followed by one or more characters, and then a number of spaces and numbers in which you will have no interest. In fact, the only part of each line with which you are concerned is that "FTN." plus one more character. Now issue this command:

```
1$g/^ *{[^ ]+}@/FI,&1::AA!1$g\/FORTRAN 77\/FORTRAN\\\/9000\/\/!wr//
```

Notice the extra escape ("\") characters. Notice also that you need not precede the vertical bars by escape characters, even though the vertical bar is a special character for EDIT/1000 (command separator). EDIT is "smart" enough to "know" that a vertical bar in the replace field is meant in its literal sense rather than in its special sense as the command separator. (The same applies to the dollar sign, which would otherwise be interpreted as the special character signifying "linked to the end of the line.") The final two slashes actually terminate this command; the two slashes preceded by backslashes ("\/\/") will become the terminating slashes in the exchange to be made on each file. Execute the command, and the previous contents of the file now become:


```
FI,FTN.1::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.2::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.3::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.4::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.5::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.6::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.7::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.8::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.9::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.A::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.B::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.C::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.D::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.E::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.F::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.G::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
FI,FTN.H::AA!1$g/FORTRAN 77/FORTRAN\/9000//!wr
```

Now use this file as a command file. First write the command file back to the disc, then transfer to it:

```
wr!tr,cmdfil,q/
```

Use the "q" option to speed things up. The "q" means "don't tell me what you're doing," so that commands and actions are not echoed to your display. Without the "q", you would see this on your display:

```
EOF
wr!tr,cmdfil/
posted file CMDFIL::Q:4
EOF
/tr,cmdfil/
opened file CMDFIL::Q:4



/FI,FTN.1::AA!1$g/FORTRAN\/9000/FORTRAN 77//!wr
closed file CMDFIL::Q:4
opened file FTN.1::AA:4
490 lines read.
  .chapter Introduction to FORTRAN 77
/1$g/FORTRAN\/9000/FORTRAN 77//
  the FORTRAN 77 compiler.
/wr
posted file FTN.1::AA:4
  the FORTRAN 77 compiler.
/FI,FTN.2::AA!1$g/FORTRAN\/9000/FORTRAN 77//!wr
closed file FTN.1::AA:4
opened file FTN.2::AA:4
1888 lines read.
1888 lines read.
  .chapter Language Elements
/1$g/FORTRAN\/9000/FORTRAN 77//
  a FORTRAN 77 extension to the ANSI 77 standard.
/wr
posted file FTN.2::AA:4
  a FORTRAN 77 extension to the ANSI 77 standard.
/File already closed cmdfil::Q:4
  a FORTRAN 77 extension to the ANSI 77 standard.
```

The "q" suppresses this display. On the other hand, if you want to monitor the commands as they are executed, do not specify the "q" option.

The final slash suppresses "asking." That is, without it, EDIT asks "OK?"

Also, make sure that everything will work as you want. If you make any mistakes, such as forgetting to specify a security code, or issuing a command that confuses EDIT, EDIT returns to interactive mode.

CHANGING LOWER CASE TO INITIAL CAPITALS

You are a technical writer. An engineer has given you input in the form of a file he created with EDIT/1000. All of the figures in his text look like this:

```
Figure 1-1 The timing modifications generator
Figure 1-2 Line noise signal monitor program block diagram etc.
```

You want them to look like this:

```
Figure 1-1.  The Timing Modifications Generator.
Figure 1-2.  Line Noise Signal Monitor Program Block Diagram.
```

First set regular expressions on and case folding on:

```
    sereon|secfon
```

Then, do it in essentially two steps. First find an instance of the line you want to change, like this:

```
    f/Figure 1-/
```

Then capitalize the first letter of every word in that line:

```
    g/{[a-z]}{[a-z]+}/>1&2//
```

Specify doing this for some number greater than the number of occurrences in which you want to make the change; the first time EDIT/1000 doesn't find the string "Figure 1-", it returns to interactive mode. The whole thing then looks this:

```
    f/Figure 1-/1g/{[a-z]}{[a-z]+}/>1&2//1_100
```

That produces the following:

```
Figure 1-1 The Timing Modifications Generator
Figure 1-2 Line Noise Signal Monitor Program Block Diagram
```

Now you just need to add two periods and one extra space. First issue the command to remove trailing spaces. (EDIT/1000 pads lines with odd numbers of bytes with an extra space.) The whole command looks like this:

```
bk!1$x/{Figure 1-[0-9]+}{/@}/&1.  &2.//
```

Voila! Well almost. Notice that you specified "x" in the last exchange instead of "g." That was so you could actually see what all of those lines look like. Since you capitalized, earlier, **every** word in them, you probably ended up with some instances of "of," "and," "to," etc., capitalized. If you don't see any such in the list you just generated (with that "x"; remember that the "g" command suppresses the listing of exchanges), then it really was "voila!" Otherwise, set case folding off ("SECFOFF"), and issue this command:

```
1$x/{Figure 1-\@}:{O}{f}:/&1<2&3//
```

The two colons ensure that the "o" of "of" comes at the beginning of a word and the "f" at the end, so that words like "Often" don't get affected. If there are any lines that have more than one "of" in them, you have to issue the command again, because the \@ covers all characters up to the **last** occurrence in the line of the following specified string. If you saw that at most there were two "of"s in any affected line, you could issue the command this way:

```
1$x/{Figure 1-\@}:{O}{f}:/&1<2&3//1_1
```

Yes, "1" and not "2". EDIT/1000 was written by a programmer; to a programmer, 0 is the first number, and 1 the second. Now you have to take care of the occurrences of "to". You needn't retype the command. Just enter two slashes ("//") to display the last command on the EDIT/1000 command stack. This is what you see:

```
---Commands---
1$x/{Figure 1-\@}:{O}{f}:/&1{2&3//
```

Move the cursor to the "O" and type in "T", move to the "f" and type in "o", and this is what you have:

```
1$x/{Figure 1-\@}:{T}{o}:/&1<2&3//
```

Press RETURN, and that command is executed.

Enter two slashes again, change the "T" to "A", the "o" to "n", use the INSERT CHAR key to add a "d", and this is what you have:

```
1$x/{Figure 1-\@}:{[A]}{nd}:/&<2&3//
```

Press RETURN, and the new command is executed.

Keep it up until you've fixed all the words you didn't want capitalized. Yes, it takes a little while and a few commands, but it sure beats going in and hand modifying all those lines.

You can fix all occurrences of "Of," "On," and "To" in one operation, instead of three, by issuing this command:

```
1$x/{Figure 1-\@}:{[TO]}{[ofn]}:/&1<2&3//
```

That would also change "Tn," "Tf," and "Oo," but you're not likely to have any of those.

COUNTING THE WORDS IN A FILE

You can't just search for all of an inclusive group; the "f" command finds only the first occurrence on each line. You can however make a replacement throughout the file that doesn't change the file; such a replacement counts all exchanges:

```
sere|1$x/{[a-z0-9]+}/&1/q/
```

(The "q" means "make the exchanges but don't show them to me.") The group to be replaced must be inclusive enough that it doesn't count certain single words as two or more each. For example, because we are considering only groups of one or more letters or numbers or both (but no other characters), 18 replacements would be counted in the following:

```
My father-in-law, Mr. Smith's, company, A-1 Charters, was rated
"No. 1," wasn't it?
```

There are not 18 words in the sentence. You could improve with this command, which, in fact, finds 13:

```
1$x/{[a-z0-9-'.,?;"]+}/&1/q/
```

But what if some of the words include colons, angle brackets, braces, asterisks, etc.? Better would be, first, to insert at least one space at the end of every line, and then replace everything except spaces:

```
1$g/{@}$/&1 //
1$x/{[^ ]+}/&1/q/
```

(If you don't first insert spaces at the ends of lines, you won't "find" words at the ends of lines, because not every line ends in a space.) That results in 13 exchanges, which is fine if you want to consider "father-in-law" as one word. If you're getting paid by the word for an article, however, you want it to be three words, so use the following:

```
1$x/{[^- ]+}/&1/q/
```

That gives 16, so be prepared to argue with the magazine about whether "A-1" is one or two words!

MANUFACTURING PRODUCTIVITY AUTOMATION AT BENDIX

by Michael J. Miller/Manager, Data Processing
The Bendix Corporation, Hydraulics Division
St. Joseph Michigan

INTRODUCTION

This article describes the use of a Hewlett-Packard HP 1000 system to monitor production at the Hydraulics Division of the Bendix Corporation located in St. Joseph, Michigan. This division is a major supplier of wheel cylinders, master cylinders, disc brake caliper and housing assemblies and power brake booster units for the passenger car and light truck industries. Foundry and manufacturing operations are performed at this facility. The application described herein, however, is relevant only to the machining and assembly operations performed in the manufacturing segment of the business.

HISTORY

Use of a computer-based system to monitor production began here in 1973 when a system was installed to monitor a new disc brake machining line. Data collection hardware and software were designed and developed by the supplier of the machining equipment — BUHR Machine Tool Company, then a subsidiary of the Bendix Corporation. From the original four machines, the system was quickly expanded to include the entire disc brake machining department — approximately 30 machines.

By today's standards, the hardware was archaic — a 4-microsecond processor, disc drives with a 500 millisecond average access time, a typewriter-like console, and a card read/punch for loading programs and data. Recompiling and reloading programs was clumsy; reconfiguration of the operating system usually took an entire weekend. There was no power fail recovery system, resulting in frequent calls to data processing personnel at all hours of the day and night for assistance in performing restarts. The serviceability of the custom built data collection and video display subsystems was of constant concern.

In spite of its shortcomings, the system quickly gained favor with the manufacturing managers in the machining area. Video monitors displayed the status of all machines in the area; it was no longer necessary to have the office find out which machines were down or how long they had been down. A quick glance at the monitor would tell a foreman whether or not his attention was needed somewhere. The tool management system was quickly adopted by machine setup men as an easy way to keep tools changed without any manual record keeping. At the same time, the foreman and superintendent in the department were getting reports, monitoring the status of tools. The system became a source of information that manufacturing people depended upon to make their jobs easier, and to keep their machines producing parts.

In 1976, a decision was made to replace the computer, data collection system and video display system with more up-to-date hardware, one which could be more easily expanded, more easily programmed and more readily serviced. Several systems were evaluated, and the HP 1000 system with an RTE operating system was eventually selected for the following reasons:

1. The RTE operating system was very similar to the existing system, making software conversion fairly simple.

2. HP offered a data collection system which could easily be interfaced with the computer.

3. HP has a sound reputation for product quality and support. Based on the existing lease and maintenance costs, the cost of purchasing the equipment would be recovered in less than two years.

The conversion to the HP system was an easy one. The basic design of the application software was not changed. The major task involved rewriting the data collection module from Assembler to FORTRAN IV. The entire conversion required less than two months, and actual out-of-service time was less than one week.

In 1978, the system was expanded from 30 monitored machines to 200. This expansion included all of the machining and assembly lines for the four major product lines. Operator entry stations were installed at each machine and features were provided to assign operators to machines, report production by part number, process dispatch service requests, etc.

Today, the system is in a relatively static state. Occasionally a change is made to accommodate a new machine or to disconnect an obsolete one. New reports are added from time to time as needed, but the system is often just casually monitoring machines, updating monitors, responding to requests for reports, printing end-of-shift production summary reports and sending service requests to the dispatch pager.

FEATURES

Four major functions are provided by the system:

1. Production status reporting
2. Production history reporting
3. Tool management
4. Service paging

The "production status reporting" function provides status information for machines on CRT monitors located at supervisors' work stations in each department. Each monitor displays only the status of stations in the immediate department. Information is updated at 90-second intervals. The information being displayed for each machine includes "idle time", current "downtime reason", "current-shift production" from a cycle relay and from a part-present sensor (if installed), and "current-shift runtime". This function is supported by continuous monitoring of cycle relay contacts, part-present sensors, and operator stations located at each machine.

The "production history reporting" function provides printed production and downtime summary reports for previously completed shifts. Reports are printed on keyboard/printer terminals located at the supervisors' work stations at the end of each shift and as requested from these terminals at other times. Production statistics are retained by the system for monthly transmission to a large host system.

The "tool management" function provides manufacturing personnel with a mechanism for replacing cutting tools at regular predetermined usage intervals. The system provides tool status reports as requested on keyboard/printer terminals and automatically generates "overdue tool change" reports at regular intervals. The keyboard/printer terminals are also used to enter tool change transactions as tools are replaced and to change the "standard tool change interval" as necessary.

The "service paging" function permits a machine operator to request assistance from a foreman or stock handler without interrupting his work cycle. A request for service is initiated by use of a "service request" switch on the operator station located at each machine. The monitoring system senses the request and forwards an appropriate message to a CRT monitor in the dispatch center. The message is then broadcast on the PA system.

BENEFITS

Broadly stated, the objective of the system is to reduce controllable manufacturing costs. Specific benefits provided are as follows:

Reduction in Avoidable Machine Downtime Based on user experience, the production status reporting system can be expected to reduce operator-related machine downtime by 20 to 30 percent. This is accomplished by providing supervisory personnel with information needed to quickly react to a downtime-producing situation.

Retooling downtime is a significant contributor to total downtime on multi-station, multi-spindle machines. On machines of this type, the tool management system can be expected to reduce total retooling time by 30 percent. This is accomplished by reducing the number of occurrences in which a machine is down to replace broken or worn tools and by permitting setup personnel to consolidate tool changes.

Stockup downtime results when a machine operator must wait for a stock trucker to deliver material to his machine. The service paging system reduces downtime due to stockup by permitting the operator to request service without interrupting his work, before his stock is depleted.

Reduction in Cutting Tool Costs In an uncontrolled environment, cutting tools will be replaced when they break or become worn to the extent that the machined part is inspected and determined to be out of spec. The tools used in this type of environment often have little, if any salvage value. The tool management system provides the means to monitor the number of pieces machined by each tool (or group of tools) on a machine, and to change them before they are permanently damaged. The system can potentially increase average tool life by 10 percent, providing a 10 percent reduction in tooling costs.

Reduction in Rejected Material Costs Costs of rejected material in a machining environment are affected by a number of factors including inspection procedures, the type of metal-working operation, the design of the machine, the type of tools used, and the manner in which tools are replaced. Costs can be especially significant on a high volume line because many rejects may be produced in a given inspection interval before being detected. In a situation such as this, a tool management system can conservatively be expected to reduce scrap and rework costs by five percent.

Reduction in Inventory Costs Many companies are today using automated requirements planning systems to order raw material from their vendors. These systems use production data from manufacturing departments to determine the current balances of raw material in stock, look at projected manufacturing schedules, and then release orders to vendors for more material. Systems of this type are nearly a necessity in a sizeable manufacturing operation today due to the importance of controlling inventory levels. One potential problem with this type of system, however, is that it is very sensitive to errors in raw data. Overstated or understated production figures will quickly be transformed into over-buys and under-buys of specific raw material items. Those errors often result in premium freight billings, obsolete inventory costs, late shipments, and in lost labor costs. The production status and history reporting features of this system provide information which has been proven to be more accurate than when collected by more conventional means.

HARDWARE

The equipment components supporting the monitoring system are as follows:

Cycle Relays The cycle relay is located in the machine's electrical control panel. An electrical contact on the relay changes from "Open" to "Closed" status whenever the machine cycles.

Part-Present Sensors Part-present sensors are devices which detect the presence of parts. The sensors contain a relay contact which changes from Open to Closed status when a part passes in front of the sensing device.

Operator Entry Stations These entry stations, located at machines, are used by operators to enter downtime causes and to request service from foremen and material truckers.

Keyboard/Printer Terminals Keyboard/printer terminals are used by manufacturing personnel to enter data, and request reports. The system controller, usually connected by a standard RS-232 connector, can automatically print messages and reports onto the terminal.

Video Driver Subsystem The video driver subsystem converts serial digital data from the system controller into a video signal needed to drive a standard video monitor. This device is normally connected to the system controller with a standard RS-232 connector.

Video Monitors Television-like CRT units are placed at strategic locations throughout the shop, normally displaying current machine status information.

Data Collection Subsystem Operator entry stations, cycle relay and part-present relay contacts are connected to the data collection subsystem, which, in turn, is connected to the system controller.

System Controller The system controller is an HP 1000 process control computer equipped with 192k bytes of main memory and a 15M byte disc drive.

SOFTWARE

Computer functions run under control of an HP RTE real-time operating system. The system was designed to be unattended. It operates continuously and is shut down only when maintenance is necessary. A computer operator is needed only to periodically transmit accumulated production data to a host batch processing mainframe. About 30 application programs, written in FORTRAN by the division's software development group, provide the following functions:

- Collecting machine status data from cycle relays, part sensors, and operator entry stations
- Updating machine status information on video monitors at regular intervals
- Reporting and initializing production statistics at the end of each shift
- Maintaining production data for historical production reports
- Servicing requests for production and tool status reports
- Reporting overdue tool changes on a regular schedule
- Changing the part number that a machine is producing
- Assigning machine operators to specific machines
- Routing foreman and material trucker requests to a dispatch area for paging

2250 BUFFER MANAGEMENT FROM DOWNLOADED SUBROUTINES

by Diana Bare/HP Roseville Division

ABSTRACT

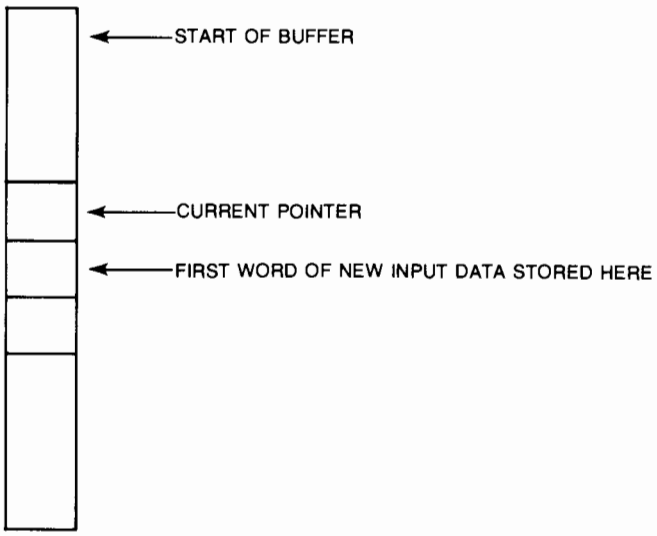
The HP 2250 Measurement and Control Processor gives the user control over a system of buffers and variables for local data storage and manipulation. Buffer data is stored and fetched relative to a buffer pointer for each buffer. The 2250's on-board language, MCL/50, provides several commands for manipulating buffer pointers, but these commands are not accessible to subroutines downloaded from an HP 1000 host. Also, MCL/50 does not provide commands for determining the current position of a buffer's pointer, or for other types of buffer status information.

This writing describes how buffers are managed internally by the 2250 firmware, and how HP 1000 Assembly language subroutines can be written to examine and change 2250 buffer status without using MCL/50. It is assumed that the reader is familiar with the 2250 and with programming in MCL/50.

INTRODUCTION

The 2250 uses an HP 1000 L-Series computer, which has a 32k word address space. The 2250 firmware resides in ROM in 16k of those words. The remaining 16k address space is RAM: 1k of base page, and 15k of system table space and user available memory. Since the 2250 uses an HP 1000 computer, subroutines can be written in HP 1000 Assembler or FORTRAN and 'downloaded' to the 2250 to be executed there. Half of the 2250's base page is reserved for use by downloaded subroutines; the other half, from 0B to 777B, is used for internal registers (A and B), interrupt trap cells, stack management, and firmware base page variables. (If no subroutines are downloaded, the 1000B words of base page reserved for downloaded subroutines become part of the user available memory.)

Several of the firmware base page variables are used to manage the 2250's buffers and buffer pointers. Buffers are allocated by executing the MCL/50 DIMENSION command. Buffers can be used to store and operate on input and output data from the 2250 function cards. Each buffer has one pointer, which is used to indicate a 'current position' in the buffer, and all I/O is done relative to the buffer pointer. The buffer pointer is a pre-increment pointer, which means that it always points to the last used word in the buffer, and must be incremented to point to the next word to be used. For example, data read into the current input buffer is stored, starting at the word immediately after the word pointed to by the buffer pointer.

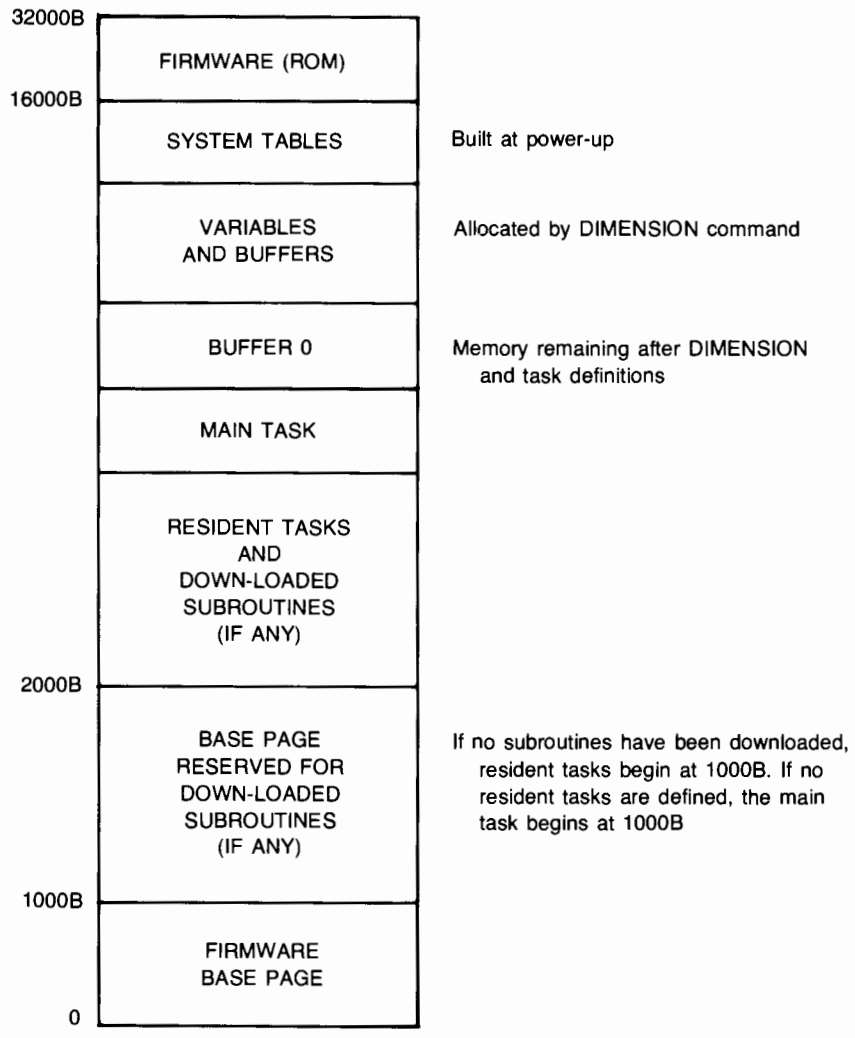


The rest of this writing discusses tables used by the firmware to manage 2250 buffers, and presents several subroutines that demonstrate how to use the firmware base page locations to examine and alter those tables.

NUMBER OF BUFFERS

2250 buffers are allocated when the MCL DIMENSION command is executed. The DIMENSION command specifies the number and size of the buffers. The buffers have fixed names: for example, if the DIMENSION command allocated 10 buffers, they are named B1, B2,...,B10. In addition to any buffers allocated by the DIMENSION command, the main MCL task executing in the 2250 always has access to a dynamically allocated result buffer, named B0. Buffer 0 contains all the unallocated, unused memory available to the user. After a main task is compiled; thus, its size changes for each new main task, and for any execution of the DIMENSION command.

The DIMENSION command can be executed at any time, from any task. Memory space for buffers is allocated from high memory, and does not interfere with memory used for storing resident tasks and down-loaded subroutines, which are stored in low memory. The DIMENSION command will not execute if doing so would not leave enough unused memory (buffer 0) to compile and execute a new main task.



In the word at address 756B on the 2250's base page, the firmware keeps the number of buffers currently defined in the 2250. This count always includes buffer 0, even if there is currently no main task. Thus, this word will always contain 1 plus the number of buffers allocated by the last DIMENSION command executed.

The subroutine CHECK, listed below, can be called by other subroutines to verify that a buffer number passed to the calling subroutine exists in the current 2250 buffer configuration. It examines location 756B, described above, to check the buffer number passed to it. If the buffer does not exist, CHECK returns a negative number as the result, flagging an error. If there is no error, the buffer number is returned as the result. This subroutine might be used by other subroutines to verify the existence of a 2250 buffer; some of the other subroutines given later in this paper use the CHECK routine.

```
ASMB,R,L
*
      NAM CHECK   Check 2250 buffer number <821207.1620>
*
      ENT CHECK
*
      EXT .ENTR
*
*  This subroutine checks a 2250 buffer number for validity.
*  It has 2 parameters: the address of the buffer number,
*  and the address of the location where the result of the
*  check is to be posted.  If the buffer number is OK, the
*  result will be the buffer number; if the buffer number
*  is not OK, the result will be -1.  The calling routine
*  should test the result, and if it is negative, report an
*  error.
*
*  Parameter storage
*
BNUM  BSS 1         ; ADDRESS of buffer number
OK    BSS 1         ; ADDRESS for decision
*
CHECK NOP
      JSB .ENTR     ; fetch parameters
      DEF BNUM
      LDA BNUM,I    ; get buffer number
      STA B         ; copy into B reg : use A reg for result
      SSA           ; if it's negative,
      CCA           ;   error : make A reg negative
      CMB           ; otherwise, add 1 and negate
      ADB BUFNM     ; add number of buffers (includes B0)
      SSB           ; if still negative,
      CCA           ;   error : make A reg negative
      STA OK,I      ; store decision : either buf num or -1
      JMP CHECK,I   ; exit
*
*
B     EQU 1
BUFNM EQU 756B      ; number of 2250 buffers defined
*
      END
```

BUFFER SIZE

The 2250 firmware maintains a table in RAM of the sizes of the buffers allocated, and the size of B0. This table has one word for each buffer, including buffer 0, and its address is stored in the base page word 760B. The buffer number is used to index into the table; that is, adding the number of the buffer to the contents of location 760B gives the address of the word containing the size of that buffer. For example, the first word of the table is the size of buffer 0, the second word is the size of buffer 1 (if defined), and so on.

The subroutine BSIZE, given below, can be called to determine the size of a particular 2250 buffer. BSIZE has two parameters, both passed by reference: the address of the buffer number, and the address where the buffer size is to be placed. If the buffer does not exist, the error will be flagged by reporting the buffer size as −1. This subroutine is written to be called from MCL: minor changes would make it more suitable for calling from another down-loaded subroutine. A sample MCL call would be: 'CALL BSIZE(V1,V2)', which would place into variable V2 the size of the buffer whose number is in V1. One example of the use of BSIZE would be for a resident task which is written to fill a variable buffer with some data. The task could call BSIZE to determine the size of the buffer it is to fill, and so it could be used for variable size buffers.

```
ASMB,R,L
*
      NAM BSIZE    Find 2250 buffer size  <821207.1620>
*
      ENT BSIZE
*
      EXT .ENTR,CHECK
*
*  Parameter storage
*
BNUM  BSS 1        ; ADDRESS of the buffer number
RESLT BSS 1        ; ADDRESS for the result
*
*  BSIZE returns the size of a specified 2250 buffer.  It
*  has two parameters: the number of the buffer, and the
*  location for the result.  Both parameters are passed
*  by reference, allowing BSIZE to be called directly
*  from MCL.  If the buffer does not exist, BSIZE will
*  return a buffer size of -1, flagging an error.
*
BSIZE NOP
      JSB .ENTR    ; fetch parameters
      DEF BNUM
      JSB CHECK    ; go check the buffer number param
      DEF *+3
      DEF BNUM,I
      DEF RESLT,I
      LDA RESLT,I  ; get CHECK result
      SSA          ; check for error
      JMP BSIZE,I  ; if error, return
      LDA BNUM,I   ; otherwise, get buffer number
      ADA BUFSZ    ; index into buffer size table
      LDA A,I      ; get buffer size
      STA RESLT,I  ; store as result
      JMP BSIZE,I  ; return
*
*
A     EQU 0
BUFSZ EQU 760B     ; pointer to buffer size table
*
      END
```

BUFFER ADDRESS

The address of the table containing the starting address for 2250 buffers is kept in location 753B in base page. This table also has only one word per buffer, including buffer 0. The number of the buffer in question is used as an index into this table to get the starting address of the buffer; thus, this table is accessed in the same way as the buffer size table, described above.

Examining the contents of the buffer starting address table and the buffer size table reveals that the buffers are allocated sequentially in high memory, and that the space set aside for each buffer has one word more than the buffer's size. Thus, if buffer 1 was dimensioned to have 100 words, then the starting address for buffer 2 would be 101 plus the starting address of buffer 1. An example:

Buffer Number	Starting Address	Buffer Size
1	34216B = 14478	100
2	34363B = 14579	200
3	34674B = 14780	300
4	35351B = 15081	400

The numbers given in this table will vary from system to system, depending on the configuration of the 2250. The extra word per buffer is allocated for use by a 2250 firmware feature that is unsupported at this time.

As an example of the use of the buffer starting address and size tables, the subroutine CLEAR given below will initialize all words in a 2250 buffer to 0. None of MCL's buffer management commands cause a buffer to be cleared, and the combination of MCL commands to put a 0 in each word would execute much slower than this subroutine.

```
ASMB,R,L
*
      NAM CLEAR    Clear 2250 buffer  <821207.1620>
*
      ENT CLEAR
*
      EXT .ENTR,BSIZE
*
*  Parameter storage
*
BNUM  BSS 1        ; ADDRESS of the buffer number
*
*  CLEAR initializes the contents of a 2250 buffer to
*  0's.  It has only one parameter, the number of the buffer
*  to be cleared.  If the buffer does not exist, no action
*  is taken.
```

```
*
CLEAR NOP
      JSB .ENTR     ; fetch parameter
      DEF BNUM
      JSB BSIZE     ; check buffer number and get its size
      DEF *+3
      DEF BNUM,I
      DEF RESLT
      LDA RESLT     ; get BSIZE result
      SSA           ; if result is negative,
      JMP CLEAR,I   ;   buffer doesn't exist - exit
      CMA,INA       : negate buf size to use as loop counter
      STA RESLT
      LDA BNUM,I    ; otherwise, get buffer number
      ADA BUFAD     ; index into buffer address table
      LDA A,I       ; get buffer address
      CLB           ; make B reg 0
LOOP  STB A,I       ; write 0 to current buffer word
      INA           ; point to next buffer word
      ISZ RESLT     ; increment loop counter
      JMP LOOP      ; not done yet, go do the next word
      JMP CLEAR,I   ; all done : return
*
*
A     EQU 0
BUFAD EQU 753B      ; pointer to buffer address table
*
RESLT BSS 1         ; BSIZE result and loop counter location
*
      END
```

BUFFER POINTER LOCATION

The current positions of the pointers for the 2250 buffers are kept in a table whose starting address is at location 753B on the firmware's base page. This table also has only one word per buffer, including buffer 0. Each word in the buffer pointer table contains the address of a word in the corresponding buffer. This word is the 'current' word in the buffer: the next I/O instruction using this buffer (via the IN or OUT commands) would begin with the word immediately after the word pointed to.

Example: if buffer 1 has just been rewound, and the starting address for buffer 1 is 14478, then the pointer for buffer 1 contains the address 14477, which is the word just before the first word of buffer 1 to be used for I/O. If 10 words of data are then read into buffer 1, the pointer will be changed to contain the address of the last word read in, 14487, which is also the address of the word just before the next word to be used.

It is important to understand which MCL commands change the contents of a buffer pointer, and which commands just examine the pointer contents, leaving them unchanged. The descriptions below are summaries of all MCL commands which affect the contents of a 2250 buffer pointer. More information on these commands is available from the 2250 Programmer's Manual (25580-90001).

The DIMENSION command creates the buffer pointer table and leaves each buffer pointer word containing the address of the word just before the start of the buffer (in the rewound position).

The REWIND command changes a buffer's pointer word to contain the address of the word just before the start of the buffer, as explained in the example above.

The SKIP command adds the given number to the contents of the buffer pointer location, first checking that the new pointer value is within the buffer bounds.

The RELEASE command looks at the current buffer pointer, using it to set up a port release of the data from the start of the buffer up to and including the word pointed to by the buffer pointer. Once the port release is set up, RELEASE then rewinds the buffer by changing the pointer word to contain the address of the word just before the start of the buffer.

The IN and OUT commands do not themselves change the contents of a buffer pointer. Execution of these commands causes the firmware to copy the input or output buffer number into a base page location, to be used by any following I/O commands. It is only when an I/O command is executed that the input or output buffer's pointer is changed.

The CBUFFER command can be used to move a buffer's pointer to the end of a block of converted I/O data. More details on this command are given in the 2250 Programmer's Manual.

No other MCL commands change a buffer pointer's contents. All arithmetic references to a buffer's contents cause the buffer pointer to be examined, but do not change its contents. For example, the MCL command 'B1(0) = 100' will change the contents of the word being pointed to by the pointer for buffer 1, which is not necessarily the first word of the buffer. Similarly, 'B1(10) = 100' will change the contents of the word whose address is 10 plus the current buffer pointer.

As an example of the use of the buffer pointer table, the subroutine BPTR given below will return the current position of a buffer's pointer relative to the start of the buffer. BPTR has two parameters, both passed by reference: the number of the buffer whose pointer is to be looked at, and the location where the result is to be stored. BPTR can be called directly from MCL. Example: the MCL task 'DIMENSION(10,1,100) ; SKIP(B1,20) ; CALL BPTR(1,V1) !' will cause variable V1 to be set to 20, which is the number of words that the buffer pointer has moved past the start of the buffer. Another example: 'REWIND(B1) ; CALL BPTR(1,V1) !' will cause V1 to be set to 0, indicating that buffer 1 was in the rewound position when subroutine BPTR was called.

```
ASMB,R,L
*
      NAM BPTR    Find current buffer pointer  <821207.1620>
*
      ENT BPTR
*
      EXT .ENTR,CHECK
*
*  Parameter storage
*
BNUM  BSS 1         ; ADDRESS of the buffer number
RESLT BSS 1         ; ADDRESS for the result
*
*  BPTR returns the current position of a 2250 buffer's
*  pointer.  The pointer position is given as an offset
*  from the start of the buffer: that is, the number of
*  words past the start of the buffer.  BPTR has two
*  parameters: the number of the buffer, and the location
*  for the result.  Both parameters are passed by reference,
*  allowing BPTR to be called directly from MCL.  If the
*  buffer does not exist, BPTR will return a pointer
*  value of -1, flagging an error.
```

```
*
BPTR  NOP
      JSB .ENTR     ; fetch parameters
      DEF BNUM
      JSB CHECK     ; check buffer number param
      DEF *+3
      DEF BNUM,I
      DEF RESLT,I
      LDA RESLT,I   ; get result parameter
      SSA           ; check for error
      JMP BPTR,I
      ADA BUFAD     ; index into buffer address table
      LDA A,I       ; get buffer starting address
      CMA,INA       ; negate
      LDB BNUM,I    ; get buffer number parameter
      ADB BUFPT     ; index into buffer pointer address table
      ADA B,I       ; add pntr addr to -(buffer start addr)
      INA           ;   and add 1,
      STA RESLT,I   ;     giving the current pointer position
      JMP BPTR,I
*
*
A     EQU 0
B     EQU 1
BUFAD EQU 753B      ; pointer to buffer address table
BUFPT EQU 755B      ; pointer to buffer pointer table
*
      END
```

SUMMARY

2250 buffers are managed by the firmware using 4 base page words and three tables. The three tables are the buffer size table, the buffer starting address table, and the buffer pointer table. Three of the base page words contain the addresses of these tables, and the fourth word contains the number of buffers defined in the 2250.

Base Page Address	Contents
756B	Number of defined buffers
760B	Address of buffer size table
753B	Address of buffer start address table
755B	Address of buffer pointer table

Each of the three tables contains one word for each buffer, including buffer 0. By examining and changing the contents of these tables, subroutines written to be downloaded to the 2250 can use the 2250 buffer system to better advantage.

Some other examples of what subroutines can do using these tables are:

1. Change the position of a buffer pointer.

2. Redefine buffers to be in the 1000B words of base page reserved for downloaded subroutines, thus making better use of available memory.

3. Cause a port release to send data from a buffer starting at some word other than the first word in the buffer, by temporarily changing the 'start of buffer' word before executing the RELEASE command.

CUSTOMER COURSES FOR ATS/1000 USERS

Two customer courses are available at Data Systems Division, Cupertino, California, for users of ATS/1000 Automatic Test Systems.

The ATS/1000 User Test Programming Course (22972B) covers the development of test programs, including sections on implementing a "system" of test programs and an overview of test system components and configurations. Prerequisites are a knowledge of BASIC, and completion of the RTE-IVB Session Monitor User's Course (22994A) and the FORTRAN IV Programming Course (22974B), or equivalent. The User Course will be held February 28, 1983, May 16, 1983, and August 8, 1983.

The ATS/1000 Advanced Course (22973B) provides in-depth study of system installation and generation, software internals, and system maintenance information for system managers and system level programmers. It is not a test programming course. Prerequisites for the Advanced Course are completion of the ATS/1000 User Test Programming Course (22972B), RTE-IVB System Manager's Course (22995A), and HP-IB User's Course (22963B), or equivalent. Dates for the Advanced Course are March 7, May 23, and August 15, 1983.

Each course is one week in length and is held at the Cupertino site only. The Advanced Course follows the User's Course, but each course requires separate registration. Customers must register through their HP field representative, who will forward by TWX or phone, your name, company name and address, and an HP sales order number to the ATS registrar at Data Systems Division.

The cost of each course is $2,000. Approximately three weeks before the course starts, a pre-study packet is sent to the registrant.

NEW PRODUCT ANNOUNCEMENTS

FAST NEW HP 1000 REAL-TIME COMPUTER RUNS AT 3 MIPS

The fastest computer ever introduced by HP, a new "flagship" for the HP 1000 real-time computer family providing 3 million-instructions-per-second speed for less then $24,000, was recently released by HP.

The new A900 technical computer is the highest-performance member of the HP 1000 family, and is believed to be the fastest real-time computer on the market today. It features a very high-speed cache memory, two-level pipelined architecture, 3.7 megabytes-per-second peak I/O bandwidth, and a standard floating-point processor capable of performing a typical mix of floating-point instructions at 560,000 instructions per second. The new A-Series computer can support up to 6 megabytes of 64k RAM main memory, and provides three times the performance of previous HP 1000 processors.

The new top-of-the-line HP 1000 A900 is targeted primarily for the OEM market as a very high-performance, real-time engine for industrial automation and process applications. It is particularly well-suited to process monitoring and control, high-speed data acquisition and image- and signal-processing applications, where the A900's raw computational speed, floating-point performance and sophisticated I/O capabilities are typical customer requirements.

New Price/Performance Standard "We believe the A900 has just redefined the price/performance standard for the minicomputer industry," said the HP Data Systems Division marketing manager. "It's not only the fastest real-time minicomputer we're aware of — it's one of the most capable. While we dramatically improved our existing HP 1000 architecture to give the A900 its exceptional performance, its software is not only compatible with, but identical to, that of other A-Series computers — the A600 and A700 machines we introduced in February. With only 25 percent of the number of parts in our previous HP 1000 flagship, the F-Series, the A-Series is expected to be the most reliable family of minicomputers ever built by HP."

The A900 processor is implemented in Schottky TTL discrete logic and comes standard with a hardware floating-point processor and HP's Scientific Instruction Set (SIS) and Vector Instruction Set (VIS) firmware. The A900 design makes liberal use of state-of-the-art programmed components and the most advanced Schottky technology available.

The floating-point capability is implemented through three LSI chips developed in HP's CMOS/SOS technology. Unlike most processors requiring a separate board for hardware floating point, the A900 floating-point chips are designed as an integral part of the CPU for maximum performance and efficiency. When executing the single-precision Whetstone benchmark (B1), the A900 is capable of nearly 1,200,000 instructions per second.

"On a dollar-for-dollar comparison, we know of no other computer that even comes close to this level of performance," the manager added. "Comparably priced computers provide only one-half to one-third, or much less, of the computational horsepower of the A900.

Available Software As with all other HP A-Series computers, the A900 supports the HP 1000 line's major software packages. These include Graphics/1000-II 2D and 3D graphics software and Image/1000 database-management software. A-Series computers also can use DS/1000-IV networking software to connect to other HP 1000 or HP 3000 systems, as well as X.25 packet-switching data-communications software.

All A-Series computers execute under the new RTE-A.1 real-time operating system which supports programming in FORTRAN 77, Pascal, BASIC, and Macro/1000 Assembly languages.

To complement its processing power, the A900 supports up to 6 megabytes of main memory available in the latest generation 64k RAM technology. A single memory array board is offered with 768k bytes of storage and up to eight boards can be configured in the A900 computer. Add-on memory packages of 768k bytes, 1.5M bytes and 3M bytes are available for $6,000, $10,000, and $16,000 respectively.

The memory controller card includes built-in error-correction logic, making this feature standard for any memory configuration used in the A900. The memory controller utilizes a 32-bit data bus to memory, which improves the efficiency of memory access. All HP A-Series computers support up to 250 megabytes of mass storage.

Available in box or system-level products, the HP A900 computer is designed for OEMs, in-house systems designers and software suppliers who require ultra-high computational performance in low-cost packaged products.

Customers can equip the A900 with a choice of nine models of graphic input/output devices (including two recently introduced color graphic terminals), six models of HP CRT terminals, a variety of printers and several disc drives, including HP's newest 3 1/2-inch, 270k byte micro-floppy flexible disc drive.

Key features of the A900 include:

- Base set speed — 3.0 MIPS
- Floating-point speed — 560 KFLOPS
- Supports 6M bytes physical memory (64k RAMs)
- Addressability to 32M bytes
- 3.7M bytes I/O bandwidth
- DMA per channel
- 4K bytes of cache memory
- Wide 48-bit micro-instruction word
- Runs all A-Series software without modification
- Error-correcting, high-performance memory standard
- Power fail recovery for 3M bytes of physical memory
- High reliability (box MTBF = 8,000 hours)
- Very low mean-time-to-repair (MTTR box = 90 minutes)
- Support of large-system peripherals
- Support of full networking capabilities

NEW APPLICATIONS SOFTWARE FOR INDUSTRIAL MONITORING AND CONTROL

Process Monitoring and Control/1000 (PMC/1000), a high-level, menu-driven applications software package for direct or supervisory monitoring and control of continuous industrial processes, was recently introduce by Hewlett-Packard.

PMC/1000 is designed for small- to medium-scale continuous processes found in both process and discrete manufacturing environments. It is also appropriate for pilot plants and hybrid manufacturing applications.

Tested for more than two years in a variety of process applications, PMC/1000 runs on HP 1000 real-time computers. A typical PMC/1000 system also would use an HP 2250 measurement and control processor as a high-level interface between the computer and sensors, actuators and loop controllers. Color displays and a large number of HP instruments and peripherals can be part of a PMC/1000 system.

"PMC/1000 bridges the gap between expensive and difficult-to-customize turnkey systems and the costly and time-consuming approach of a user's writing process control software from scratch," said James D. Olson, manager of HP's Manufacturing Applications Program.

"Its low cost — relative to existing alternatives — and the fact that no user programming is needed to configure or operate a PMC/1000 system should go a long way toward justifying the many benefits of bringing computer control to process applications," Olson said. "While you don't have to be a computer expert to use it, PMC/1000 is extremely flexible, and can be applied to virtually any continuous process. We believe it offers far more computational capabilities than larger process control systems, and at a much lower cost."

The new HP software uses a fill-in-the-blanks menu format to specify control loops, conversions, tolerance limits, alarm conditions, historical logs and color displays. A "HELP" key accesses a built-in guidance system that gives users "how to" information as they go along.

The software's flexible, block architecture enables more experienced users to configure complex control structures and access the entire process data base. All program development, networking and system capabilities of the host HP 1000 computer remain fully accessible.

Customizable capabilities of PMC/1000 include flexible scanning and updating of all measured and controlled process parameters. Control actions include both proportional-integral-derivative (PID) loop control and Boolean logic control. Standard functions include engineering unit conversions and averaging, accumulation, and ratio computations. Dead-time, filtering and non-linear functions are provided, and user-defined algorithms also can be accommodated.

A large number of process strategies are within the software package's capabilities, such as feed-forward, cascade, cross-coupled and non-linear control. PMC/1000 thus is a good fit for such sophisticated applications as material and energy balancing.

Color display choices include showing the current status, or trends, of one, eight, or 32 process parameters at a time. Area, group, detail and multi-variable trends can be displayed.

In the event of an alarm condition, PMC/1000 ensures that appropriate action is taken automatically — from simple operator notification to shutdown of the process. A logging feature for historical data provides long-term graphical trending, as well as information needed for statistical analysis and management reports.

PMC/1000 is the latest addition to HP's Manufacturers Productivity Network (HP-MPN), which is the strategy for providing linked computer solutions throughout a manufacturing company. HP focuses existing products, and product development efforts, on providing practical computer tools and solutions for manufacturers.

The HP-MPN concept emphasizes providing both hardware and software products in four major areas: factory and plant automation, computer-aided engineering, operational planning and control, and administrative and office services.

JOIN AN HP 1000 USER GROUP!

Our thanks to Sandra Hawker, Interface 1000 editor, for providing this list.

U.S. LOCAL USERS GROUPS

California
Bay Area
Paul Vallis
Amdahl Corp.
1250 E. Arques Aves. M/S 140
Sunnyvale, California 94086
408/746-8285

LAB 1000 Users Group
Eva Kuiper
Puritan-Bennett Corporation
12655 Beatrice Street
Los Angeles, California 90066
213/827-9000

San Diego
Dave Petrie
Code 6111
Naval Ocean Systems Center
San Diego, California 92152
Ph. 714/225-2556

Georgia
Bob Albers
AT&T
P.O. Box 7800
Atlanta, Georgia 30357
404/873-7784

Eastern Idaho
George E. Santee, Jr.
P.O. Box 1604
Idaho Falls, Idaho 83401
208/523-7255

Illinois
David Olson
1846 W. Eddy St.
Chicago, Illinois 60657
Ph. 312/542-7036

Missouri
Greater Kansas City Users Group
William D. Jackson
P.O. Box 7535
N. Kansas City, Missouri 64116
Ph. 816/997-4763

Michigan
Detroit/1000
P.O. Box 332
Southfield, Michigan 48037

New Jersey and New York
Bennett Meyer
Singer-Kearfott
1150 McBride Avenue
Little Falls, New Jersey 07424

New Mexico
Greater White Sands Users Group
Guy Gallaway
Dynalectron Corp.
Radar Backscatter Division
P.O. Drawer 0
Holloman AFB, New Mexico 88330
Ph, 505/679-2472, ext 2770

Pennsylvania
Harry Spain
Westinghouse Electric Corp.
Bettis Automatic Power Lab.
P.O. Box 79
West Misslin, Pennsylvania 15122
Ph. 412/462-5000

DVR 1000 Delaware Valley Region Users Group
Jock McFarlane
RCA Laboratories
Princeton, New Jersey 08540
609/734-2206

Pacific Northwest Local Users Group
(Washington, Oregon, N. Idaho)
Phil Hardin
LYNX Corporation
15122 SE 46th Way
Bellevue, Washington 98006
Ph. 206/643-7472

U.S. LOCAL USERS GROUPS (CONTINUED)

Rhode Island
HP Oceanographic
227 Watkins
Graduate School of Oceanography
University of Phode Island
Kingston, Rhode Island 02881
Ph. 401/792-6116

Texas
DFW HP 1000 Users Group
Ken Penrod
P.O. Box 2674
Richardson, Texas 75080
Ph. 214/690-4449

Utah
North Central Utah HP 1000 Users Group
Malcolm Crawford
Brigham Young University
459 CB
Provo, Utah 84602
Ph. 801/378-4344

Salt Lake City HP 1000 Users Group
David Pope
BPC
391 South Chipeta Way, Suite F
Salt Lake City, Utah 84108
Ph. 801/581-5850

Washington, D.C. — Baltimore
Dan Steiger
Naval Research Lab
Code 5003
Washington, D. C. 20375
202/767-2384

EUROPEAN LOCAL USERS GROUPS

Belgium
Belgian HP 1000 Users Group
J. Tiberghien
Vrije Universiteit Brussel
Afd. Informatie
B-1050 Brussell
Belgium

France
French HP 1000 Users Group
Jean-Louis Rigot
Technicatime TA-DE-SET Cadarach
B.P. 1
F-13115 Saint Paul les Durance
France
Ph. 42/253952

Germany
German HP 1000 Users Group
Hermann Keil
Vorwerk & Co Elektrowerke KG
Rauental 38-40
D-5600 Wuppertal 2
Germany
Ph. 202/6093-3105

Netherlands
Dutch HP 1000 Users Group
Johannes Mondria, Chairman
Delft Hydraulics Laboratory
P.O. Box 152
8300 AD Emmeloord
The Netherlands
Ph. 05274/2922

Mathijs Beekman, Secretary
Cyt-U-Universitair
Radboutkwartier 261
NL 3511 CK UTRECHT
The Netherlands
Ph. 030/315624

EUROPEAN LOCAL USERS GROUPS (CONTINUED)

Norway
Norwegian HP 1000 Users Group
Per Otterson
National Inst. of Technology
Akersveien 24C
Oslo 1
Norway
Ph. 02/204550

Sweden
Swedish HP 1000 Users Group
Fred Moller
SAAB-SCANIA
C-Lab
S-15787 Sodertalje
Sweden
Ph. 0755/51000

USERS GROUPS OUTSIDE U.S.

Australia
Sydney
HP 1000 Users Group
Jeffrey Deakin
GPO Box 3060
Sydney NSW 2001
Australia

Melbourne
HP 1000 Users Group
Owen Marsh
Department of Transport, (RDSE)
G.P.O. Box 1839Q
Melbourne, 3001, Australia

Canada
Western Canadian HP 1000 Users Group
John Blommers
Defense Research Establishment
Pacific
FMO Victoria VOS 1BO
Canada

Toronto HP 1000 Users Group
Brenda Hogg
Grant Hallman Associates Ltd.
155 Gorden Baker Road, Suite 207
Willowdale, Ontario M2H 3N5
Canada
Ph. 416/498-8500

South Africa
South African HP 1000 Users Group
Andrew Penny
Hewlett-Packard
South Africa Pty.
Private Bag Wendywood
Sandton, 2144 South Africa
Ph. 802-1040

Switzerland
Swiss HP 1000 Users Group
Graham K. Lang
RCA Laboratories
Badenerstrasse 569
8048 Zurich, Switzerland
Ph. 01/526350

United Kingdom
United Kingdom HP 1000 Users Group
Kevin O'Meara
Thurston Software
Bldg. 73B Stansted
Airport London
Stansted, Essex
England CM24 8QW
United Kingdom

Although every effort is made to ensure the accuracy of the data presented in the **Communicator,** Hewlett-Packard cannot assume liability for the information contained herein.

Prices quoted apply only in U.S.A. If outside of U.S., contact your local sales and service office for prices in your country.

Printed in U.S.A. 2/83

Part No. 5955-6111
Vol. V #6