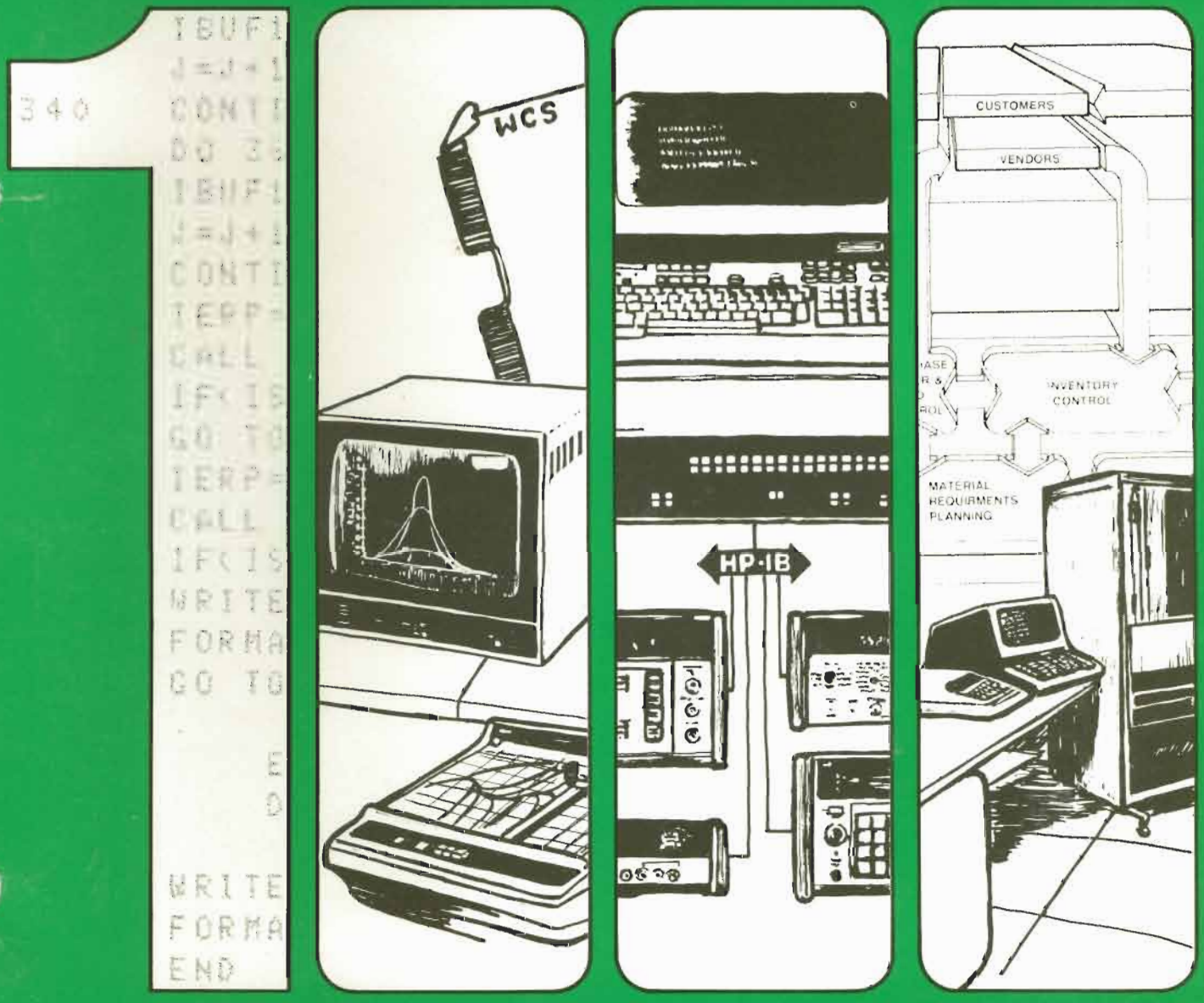
HEWLETT PACKARD

Hewlett-Packard Computer Systems

COMMUNICATOR



1981

Volume V Issue 2

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HEWLETT-PACKARD
COMPUTER SYSTEMS

Volume V
Issue 2

COMMUNICATOR/1000

Feature Articles

Departments

ABOUT THIS ISSUE

Before I let you know about the second issue of the Communicator/1000 for 1981, bear with a plea from the Editor. We are receiving an underwhelming number of Feature Articles. Articles for future issues have yet to be selected, so if you have any ambition to write the Great American novel, perhaps you would consider starting with an article for the Communicator. As a general guideline, this is predominately a textual magazine rather than a forum for lengthy program listings.

Our first article in this issue is in the area of Operating Systems. Marty Silver of HP's Data Systems Division has written an article on Multi-Station Traps with Basic/1000. This will be of particular interest to those readers in the automatic test field who have a need for on-going testing in conjunction with program development. Marty has obviously put a lot of effort into this Feature article. Thanks for a fine job! Unfortunately, Marty was not eligible for a calculator since he is a member of our Technical Marketing Department.

We have two articles in the area of Operations Management. One is from one of HP's Systems Engineers from the El Paso, Texas office, John Pezzano. John has written a concise, useful article on utilizing the unused memory at the end of a FORTRAN program. If his name seems familiar, that's because John has been one of our past contributors. We appreciate your strong support, John, but where do you keep all those calculators?

The second article in Operations Management is by another one of our own, Gary Ericson from Technical Marketing, Data Systems Division. (Boy, did we get off cheap this time!!) Gary's contibution will increase our readers abilities to work with FMP calls effectively in Database Management. As Gary points out, there are some pitfalls to watch out for and with his tips, we hopefully will avoid some wasted time and effort.

Our calculator awards are easy this time, since two out of our three articles were not eligible for selection. The remaining contributor will receive an HP32E calculator.

Best Feature article by an HP Field Employee:

John Pezzano
HP El Paso, Texas

"Using Memory behind your FORTRAN Program"

I hope you enjoy this issue. Keep those cards and letters coming!.

Sincerely,

The 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 not in the Data Systems Division Technical Marketing Dept.

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

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. Employees of Technical Marketing at HP's Data Systems Division factory in Cupertino are not eligible to win a calculator.

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

CORRECTIONS TO PREVIOUS ISSUES

Volume IV, Issue Six of the Communicator/1000 included an article by Charles F. Fugee entitled "Documentation Enhancement". Mr. Fugee's article discusses different methods of invoking the expanded and compressed print modes of the 263X line printer family. The article also states that the same escape sequences can be sent to the 2608 line printer. The 2608 however, has neither a compressed print mode nor an expanded mode and the RTE-IVB driver for the 2608 requires Exec control requests (not escape sequences) to enable alternate print modes. Therefore, the documentation enhancements that Mr. Fugee discusses apply only to the 263X family of printers, and not to the 2608.

Volume V, Issue One of the Communicator/1000 had an error in the article entitled "Designing a High-Performance Data-Capture System" by Carl Reynolds. On page 43, the last line of FORTRAN code omitted the second of seven parameters. The line should have appeared as follows:

```
CALL TMDFN(KEEP1,KEEP1,KEEP2,ITSNU,ITSNU,KEEP3,ICOMEN)
```

The editor of the Communicator apologizes for any inconvenience that may have been caused by these oversights.

LETTERS TO THE EDITOR

Dear Editor,

In the Communicator/1000 Volume IV, Issue 3 was an article entitled "Generating RTE for Pleasure and Performance". This article was excellent with many ideas that we have been forced to learn through experience.

The section with the title "Reduce the Number of Tracks Your System Requires" prompts me to write. Although the suggestion of loading permanent programs on-line after the System Generation is a good one (and at times necessary since RT4GN has finite table areas), a permanent program loaded later is assigned a whole track and as a result, this method actually uses more disc space.

A technique that I evolved about four years ago to solve the problem is as follows. I determined the NAM's of all members of the non- essential libraries. In the parameter setting portion of the RT4GN answer file, I now set all these NAM's to type 8 (subroutine only required at RT4GN time). This technique works well and as stated in the article greatly reduces LOADR time.

Yours truly,

Robert J. Meldrum
Telesat Canada

Dear Mr. Meldrum,

Thank you for the additional suggestion to Mr. Kurtz's article.

Sincerely,

The Editor

EDITOR'S DESK

Dear Editor,

Mr. Liu's program to print the contents of a 264X terminal screen (Volume IV, Issue 3, pp. 15-17) works great for my 2645 and 2647 terminals. However, whenever I tried it on my 2640B terminal, the message "1B" appeared on the screen, and the keyboard locked. The terminal was restored to normal operation after I OF'ed the program from another terminal. Any explanation?

Sincerely,

Ronald F. Lee
Gulf Oil Chemicals Company

Dear Mr. Lee,

Escape commands on the 264X line of terminals are upward-compatible to terminals introduced within the family. However, those introduced with a later member of the family are not backward-compatible. The command <ESC>&k1B is used by Mr. Liu to enable block mode on the terminal. This command unfortunately is not available on the 2640B. The message "1B" is caused by the terminal entering command mode upon receipt of the escape character and returning to display mode upon detection of an invalid command (after the lower-case k).

Sorry for any confusion caused by the phrase 264X terminal.

Sincerely,

The Editor

Dear Editor,

I have to modify some system programs in the RTE-IV operating system. I would like to know the purpose of two entry points, $PVCN and $BMON.

Sincerely,

Jaromir Vostry
Institute of Physics
Charles University
Prague, Czechoslovakia

Dear Mr. Vostry,

$PVCN is an entry point in Table Area 1. It is used as a level count for privileged and reentrant calls. If entering a "privileged" mode in $LIBR, $PVCN is incremented from zero to one. If $LIBR is entered again, $PVCN will indicate how deep reentrancy has gone. Exit via $LIBX will set $PVCN from non-zero to zero upon return to "non-privileged" mode.

$BMON is a flag in %BMPG1. The value of $BMON is equal to one in RTE-IVB systems and zero in RTE-II, RTE-III and RTE-IVA systems.

Sincerely,

The Editor

FORWARD FILE BY FILE NUMBER

by Keith J. Kunz/HP Salt Lake City

You want to get to the ninth file on that mag tape or mini- cartridge. You find the 'CN,lu,FF' command only to discover that you must type nine of them to get where you want to go! If you are lazy you put the command in a softkey but if you are clever you build the following transfer file called 'CN'.

```
:SE,1G,2G,3G,00
:CN,1G,2G
:CA,4,4G,+,1
:IF,3G,NE,4G,-3
::
```

Now you simply type :TR,CN,lu,FF,x where lu is the logical unit of the mag tape or minicartridge and x is the index of the file you want positioned. No more counting CN commands!

Idea Credit - Harry Geary - H. E. Cramer/Salt Lake City

GET MORE OUT OF YOUR DISC WITH THE SPARE CARTRIDGE POOL AND TAPE

by Jeff Deakin/Lever & Kitchen, Australia

The files in most systems can be classified into a number of sets containing files that are related in some way. For example, the files comprising FORTRAN 4X could form one group, and those comprising IMAGE another group. Very frequently, such groups are stored on the disc in a fairly disorganised manner.

One technique for maintaining an ordered system is to create a "large" number of pool cartridges which can be dynamically allocated and dismounted from the system. This can be done through the use of the :AC,crn and :DC,crn,RR commands. In conjunction with magnetic tape backup (using WRITT), this enables a highly organised system which can actually utilise far more space than exists on the disc. Prior to dismounting, each cartridge should be backed up onto tape. Later, when re-allocating, run READT to overlay the disc. Note that both READT and WRITT have the capability to move directory tracks if different sized cartridges are used for a given set of files.

A major disadvantage of this technique is that all information about UNMOUNTED cartridges in the spare cartridge pool is unavailable since such cartridges do not appear on the cartridge list. To make effective use of the spare cartridge pool, the user needs to know:

1. which cartridges are free, and
2. how many tracks each has.

With a view to providing this information, the program POOLS has been written, and is listed below. The program obtains the pool cartridge list from the accounts file, and compares it with the current cartridge list to see which pool LU's are not mounted. For each spare LU, it then fetches the subchannel number from the Device Reference Table as well as the associated number of tracks from $TB32. The output has the form:

```
1     Spare out of 8 Pool Disc LUs
LU    #Tracks
25    200
```

```
FTN4X,L,Q
      PROGRAM POOLS(3),LIST POOL CARTRIDGES
      INTEGER DCB(144),ABUF(128),NAM(3),ISTAT(125),SPARE(40)
      DATA NAM/25500B,41503B,52041B/,ISC/103066B/,ICR/-3/,IOPTN/1B/
*     OBTAIN POOL LIST
      CALL OPEN(DCB,IERR,NAM,IOPTN,ISC,ICR)
      IF(IERR.GE.0) GO TO 1
      WRITE(1,19) IERR
      GO TO 18
1     CALL READF(DCB,IERR,ABUF,128,LEN,1) !READ HEADER IN ACCOUNT FILE
      IF(IERR.GE.0) GO TO 2
      WRITE(1,20) IERR
      GO TO 18
2     IPOOL=ABUF(3) !DISC POOL ADDRESS
      CALL READF(DCB,IERR,ABUF,128,LEN,IPOOL) !READ POOL LU'S
      IF(IERR.GE.0) GO TO 3
      WRITE(1,20) IERR
      GO TO 18
3     CALL CLOSE(DCB)
      DO 4 I=1,128
4     IF(ABUF(I).EQ.0) GO TO 5
5     NPOOLS=I-1
      NP=0
      IF(NPOOLS.EQ.0) GO TO 11
      CALL FSTAT(ISTAT,125,0,1) !GET CARTRIDGE LIST
*     CHECK EACH POOL LU TO SEE IF ALLOCATED
      DO 10 I=1,NPOOLS
         LUPOOL=ABUF(I)
         DO 6 J=1,125,4
            IF(ISTAT(J).EQ.0) GO TO 6
            IF(ISTAT(J).EQ.LUPOOL) GO TO 10
6        CONTINUE
         NP=NP+1
         SPARE(NP)=LUPOOL
10    CONTINUE
11    WRITE(1,12) NP,NPOOLS
12    FORMAT(I4," SPARE OUT OF",I3," POOL DISC LUS")
      IF(NP.GT.0) THEN
         WRITE(1,14)
         DO 13 I=1,NP
            CALL SUBCH(SPARE(I),ISUBCH,IEQT) !GET SUBCHANNEL
            CALL TRAK(ISUBCH,ITRK) !GET LAST TRACK
            WRITE(1,15) SPARE(I),ITRK
13       CONTINUE
14       FORMAT("    LU  #TRACKS")
15       FORMAT(3X,I3,I7)
      END IF
18    CONTINUE
19    FORMAT("OPEN ERROR",I4)
20    FORMAT("READ ERROR",I4)
      END
```

```
*
      SUBROUTINE SUBCH(LU,ISUBCH,IEQT),GET SUBCH & EQT # OF LU
*     RETURNS SUBCHANNEL # & EQT # OF DISC LU
      IDRT=IGET(1652B)+LU-1 !DRT ADDRESS OF LU
      ID=IGET(IDRT) !DRT ENTRY
      IEQT=IAND(ID,77B) !EQT #
      ISUBCH=ISHFT(ID,-11) !GET SUBCHANNEL #
*     IS IT A DISC SUBCHANNEL?
      IQT=IGET(1650B)+15*(IEQT-1)+4 !ADDRESS OF ENTRY IN EQT WORD 5
      IQT=IGET(IQT)
      IQT=IAND(IQT,37400B)/256
      IF(IQT.NE.32B) THEN
         ISUBCH=0
         IEQT=0
      ENDIF
      RETURN
      END
      END$
```

```
ASMB,L
      NAM TRAK Get #tracks from $TB32
      ENT TRAK
      EXT $TB32,.ENTR
SUBCH BSS 1
TRAKS BSS 1
TRAK  NOP
      JSB .ENTR
      DEF SUBCH   Address of subchannel
      LDA SUBCH,I   Get subchannel
      MPY =D5     Form address
      ADA =D4       in $TB32
      XAX           store in index reg X
      LAX TB32,I    Get word in $TB32
*                   indexed by X
      STA TRAKS,I    Save in TRAKS
      JMP TRAK,I    Exit
TB32  DEF $TB32       Address of $TB32
      END
      END$
```

SYSTEM IDENTIFIER FOR RTE

by Dan Barnes/HP Data Systems Division

Do you need a way to identify which system RTE is running on? Do you have multiple systems that are configured similarly but require different adjustments for each one, such as initializing the DS node? Would it help if you could get a number indicating which hardware configuration RTE is running on? This article presents a program (SYSNO) that retrieves a system number and suggests a way to use that number.

I have four systems linked with DS/1000-IV (labeled nodes 1-4). RTE needs to know which system it is running on to initialize the DS subsystem with the correct node number. The program SYSNO will retrieve a system number from 0 to 7 (I use 1 to 4). Using the IF statement in FMGR I can transfer control to the statement that will initialize DS with the correct node number.

```
:RU,SYSNO              Get the system number
:CA,8,1P               Transfer it to the global 8G
:IF,8G,NE,1,2          Is this system 1?
:RU,DINIT,*DINI1       Yes, initialize DS as node 1
:IF,8G,EQ,8G,10        Skip the remaining tests
:IF,8G,NE,2,2          Is this system 2?
:RU,DINIT,*DINI2       Yes, initialize DS as node 2
:IF,8G,EQ,8G,7         Skip the remaining tests
:IF,8G,NE,3,2          Is this system 3?
:RU,DINIT,*DINI3       Yes, initialize DS as node 3
:IF,8G,EQ,8G,4         Skip the remaining tests
:IF,8G,NE,4,2          Is this system 4?
:RU,DINIT,*DINI4       Yes, initialize DS as node 4
:IF,8G,EQ,8G,1         Skip the error message
:DP,Unable to initialize DS/1000.  Call the system manager


 *DINI1     initializes the system as node 1
 *DINI2     initializes the system as node 2
 *DINI3     initializes the system as node 3
 *DINI4     initializes the system as node 4
```

The program SYSNO requires a 12979B Dual-port I/O extender. The I/O bus switch feature is used to hold the system number.

I lock my I/O extender to port A and configure the I/O bus switch to one of eight select codes (70B through 77B). I use select codes 71B through 74B to indicate systems 1 through 4. A SFC is performed on all eight select codes. The select code that causes a skip condition indicates which system is which (e.g. select code 74B is system 4). The value 0 through 7 is returned in the global 1P.

```
ASMB,R,L
      NAM SYSNO,3,1 GET SYSTEM NUMBER  REV 1.0 <810710.1644>
**!
**!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!**
**!                                                                          !**
**! NAME:         SYSNO                                                      !**
**!                                                                          !**
**! PURPOSE:      TO RETURN THE SYSTEM # (0-7) AND THE SYSTEM ID.            !**
**!                                                                          !**
**! ENTRY         SYSNO                                                      !**
**! POINTS:                                                                  !**
**!                                                                          !**
**! CALLING       RU,SYSNO                                                   !**
**! SEQUENCE:                                                                !**
**!                                                                          !**
**! PARAMETERS: NONE                                                         !**
**!                                                                          !**
**! RESULT:       SYSTEM # IN 1P.                                            !**
**!               SYSTEM ID IN 2P.                                           !**
**!               ZERO IN 3P, 4P AND 5P                                      !**
**!                                                                          !**
**! ERRORS:       RETURNED VALUE OF -1 (IN 1P) INDICATES IT COULD NOT        !**
**!               DETERMINE WHICH OF EIGHT SYSTEMS IT WAS RUN ON.  IT        !**
**!               MIGHT MEAN YOU DO NOT HAVE AN I/O EXTENDER HOOKED UP       !**
**!               OR YOUR I/O EXTENDER HAS A PROBLEM.                        !**
**!                                                                          !**
**! EXTERNAL      $LIBR,$LIBX,EXEC,PRTN                                      !**
**! REFERENCES:                                                              !**
**!                                                                          !**
**! METHOD:       THE EIGHT SELECT CODES (70B THROUGH 77B) USED BY THE       !**
**!               12797B I/O EXTENDER FOR SWITCHING THE I/O EXTENDER         !**
**!               BETWEEN TWO CPU'S IS USED AS A SYSTEM IDENTIFIER.          !**
**!                                                                          !**
**!               A SFC SC IS PERFORMED ON ALL 8 SELECT CODES. THE ONE       !**
**!               SELECT CODE THAT DOES CAUSE A SKIP IS ASSUMED TO           !**
**!               INDICATE THE SYSTEM # (I.E. SC 74B INDICATES SYSTEM        !**
**!               #4).  IF MORE THAN ONE SELECT CODE CAUSES A SKIP OR        !**
**!               NONE OF THE SELECT CODES CAUSES A SKIP, AN ERROR IS        !**
**!               ASSUMED AND A VALUE OF -1 IS RETURNED IN GLOBAL 1P.        !**
**!               THE SYSTEM ID IS RETURNED IN 2P.  IT WILL INDICATE         !**
**!               WHICH SELECT CODES CAUSED A SKIP CONDITION (WERE           !**
**!               CONFIGURED).                                               !**
**!                                                                          !**
**! NOTES:        YOU SHOULD NOT USE THIS PROGRAM IF YOU ARE SHARING A       !**
**!               I/O EXTENDER BETWEEN TWO CPU'S.  MAKE SURE YOUR            !**
**!               EXTENDER IS LOCKED TO THE PORT YOU ARE USING.  THIS        !**
**!               PROGRAM IS BEING RUN ON SYSTEMS LOCKED TO PORT A.          !**
**!                                                                          !**
**! ASSUMPTION: THE SELECT CODES NOT CONFIGURED WILL NOT CAUSE A SKIP.       !**
**!               THE SELECT CODE CONFIGURED WILL CAUSE A SKIP.              !**
**!                                                                          !**
**! PARAMETERS    SYSTEM # - A VALUE FROM 0 TO 7 (IN A 16 BIT WORD)          !**
**! RETURNED:     INDICATING WHAT SYSTEM SYSNO IS BEING RUN ON.              !**
**!               RETURNED IN GLOBAL 1P.                                     !**
**!                                                                          !**
**!               SYSTEM ID - A VALUE INDICATING WHICH SELECT CODES          !**
**!               CAUSED A SKIP TO OCCUR IN RESPONSE TO A SFC SC             !**
**!               INSTRUCTION ON EACH SELECT CODE.  STATUS OF SELECT         !**
**!               CODES 70B THROUGH 77B ARE RETURNED IN BITS 0 THROUGH 7     !::
**!               OF GLOBAL 2P.  BITS 8 THROUGH 15 WILL BE SET TO ZEROS.     !**
**!               GLOBALS 3P, 4P AND 5P ARE SET TO ZERO.                     !**
**!                                                                          !**
**!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!**
**!
```

```
**!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!**
**!                                                                            !**
**! B377  MASK USED TO ZERO OUT BITS 8 THROUGH 15 OF THE SYSTEM ID             !**
**!       BEFORE IT IS RETURNED IN 2P.                                         !**
**!                                                                            !**
**! BIT0  USED TO INDICATE (IN THE ID) SELECT CODE 70B WAS CONFIGURED.         !**
**!                                                                            !**
**! BIT1  USED TO INDICATE (IN THE ID) SELECT CODE 71B WAS CONFIGURED.         !**
**!                                                                            !**
**! BIT2  USED TO INDICATE (IN THE ID) SELECT CODE 72B WAS CONFIGURED.         !**
**!                                                                            !**
**! BIT3  USED TO INDICATE (IN THE ID) SELECT CODE 73B WAS CONFIGURED.         !**
**!                                                                            !**
**! BIT4  USED TO INDICATE (IN THE ID) SELECT CODE 74B WAS CONFIGURED.         !**
**!                                                                            !**
**! BIT5  USED TO INDICATE (IN THE ID) SELECT CODE 75B WAS CONFIGURED.         !**
**!                                                                            !**
**! BIT6  USED TO INDICATE (IN THE ID) SELECT CODE 76B WAS CONFIGURED.         !**
**!                                                                            !**
**! BIT7  USED TO INDICATE (IN THE ID) SELECT CODE 77B WAS CONFIGURED.         !**
**!                                                                            !**
**! COUNT VARIABLE USED TO HOLD THE NUMBER OF SC'S THE RESPONDED               !**
**!       (HOPEFULLY ONLY ONE) AND USED TO CONVERT THE SYSID TO THE            !**
**!       SYSTEM # (0-7).                                                      !**
**!                                                                            !**
**! DEFS  VARIABLE USED AS A POINTER TO INDEX THROUGH THE TABLE OF             !**
**!       SELECT CODE CONFIGURED INDICATERS.                                   !**
**!                                                                            !**
**! DEFTB POINTER USED TO INITIALIZE THE VARIABLE DEFS.                        !**
**!                                                                            !**
**! EXIT  USED TO INDICATE TO RETURN TO THE SYSTEM.                            !**
**!                                                                            !**
**! KT    VARIABLE USED AS A LOOP COUNTER.                                     !**
**!                                                                            !**
**! MD8   CONSTANT OF -8 USED TO INITIALIZE THE LOOP COUNTER KT.               !**
**!                                                                            !**
**! PARM3 DUMMY VARIABLE FOR PASSING INFORMATION TO GLOBAL 3P.                 !**
**!                                                                            !**
**! PARM4 DUMMY VARIABLE FOR PASSING INFORMATION TO GLOBAL 4P.                 !**
**!                                                                            !**
**! PARM5 DUMMY VARIABLE FOR PASSING INFORMATION TO GLOBAL 5P.                 !**
**!                                                                            !**
**! SC70  EQUATE SETTING UP THE SELECT CODE 70B.                               !**
**!                                                                            !**
**! SC71  EQUATE SETTING UP THE SELECT CODE 71B.                               !**
**!                                                                            !**
**! SC72  EQUATE SETTING UP THE SELECT CODE 72B.                               !**
**!                                                                            !**
**! SC73  EQUATE SETTING UP THE SELECT CODE 73B.                               !**
**!                                                                            !**
**! SC74  EQUATE SETTING UP THE SELECT CODE 74B.                               !**
**!                                                                            !**
**! SC75  EQUATE SETTING UP THE SELECT CODE 75B.                               !**
**!                                                                            !**
**! SC76  EQUATE SETTING UP THE SELECT CODE 76B.                               !**
**!                                                                            !**
**! SC77  EQUATE SETTING UP THE SELECT CODE 77B.                               !**
**!                                                                            !**
**! SYS#  VARIABLE HOLDING THE SYSTEM # TO BE PASSED BACK IN GLOBAL 1P.        !**
**!                                                                            !**
**! SYSID VARIABLE HOLDING THE SYSTEM ID TO BE PASSED BACK IN GLOBAL 2P.       !**
**!                                                                            !**
**!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!**
**!
```

```
      EXT $LIBR,$LIBX,EXEC,PRTN
A     EQU 0
B     EQU 1
SC70  EQU 70B       SC70-SC77 ARE THE SELECT CODES TO
SC71  EQU 71B       BE CHECKED
SC72  EQU 72B
SC73  EQU 73B
SC74  EQU 74B
SC75  EQU 75B
SC76  EQU 76B
SC77  EQU 77B
*
SYSNO JSB $LIBR     TURN OFF MEMORY PROTECT AND
      NOP           THE INTERRUPT SYSTEM
*
      CLA           BY CLEARING THE A REG AND THEN
      SFC SC70      TURNING ON A BITS REPRESENTING
      IOR BIT0      THE SELECT CODES THAT DO NOT
      SFC SC71      CAUSE A SKIP (BITS 0-7) AND THEN
      IOR BIT1      COMPLEMENTING THE RESULT.  THE
      SFC SC72      ONE BIT REMAINING SET INDICATES
      IOR BIT2      WHICH SELECT CODE IS CONFIGURED.
      SFC SC73      SELECT CODE 74B WOULD BE INDICATED
      IOR BIT3      BY BIT 4 BEING SET AND ALL OTHERS
      SFC SC74      BEING CLEAR
      IOR BIT4
      SFC SC75
      IOR BIT5
      SFC SC76
      IOR BIT6
      SFC SC77
      IOR BIT7
      CMA
      AND B377      ZERO BITS 8-15
      STA SYSID     SAVE THE ID FOR LATER PROCESSING
      JSB $LIBX     TURN MEMORY PROTECT AND THE
      DEF *+1       INTERRUPT SYSTEM BACK ON
      DEF *+1
*
*     CHECK TO SEE IF MORE THAN ONE BIT IS ON OR
*     IF NONE OF THE BITS ARE ON.  IF MORE THAN
*     ONE IS ON OR NONE ARE ON IT MEANS THAT
*     NONE OR MORE THAN ONE SELECT CODE RESPONDED
*     THIS IS AN ERROR.
*
      LDB SYSID     GET THE VALUE TO BE CHECKED
      LDA MD8       SET THE COUNTER TO -8
      STA KT
      LDA DEFTB     INITIALIZE THE TABLE POINTER
      STA DEFS
      CLA
      STA COUNT
      SZB,RSS       IF THE SYSTEM ID IS ZERO
      JMP ERROR     THIS IS AN ERROR
LOOP1 LDA DEFS,I    GET A TABLE ENTRY
      CPA B         DOES IS MATCH
      ISZ COUNT     YES, BUMP THE COUNTER BY ONE
      ISZ DEFS      BUMP THE TABLE POINTER
      ISZ KT        BUMP THE LOOP COUNTER
      JMP LOOP1     IF NOT ZERO GO LOOP
      LDA COUNT     SEE IF THE COUNT IS 1
      CMA,INA       IF IT IS NOT THEN THERE IS AN
      ISZ A         ERROR (MULTIPLE SELECT CODES
      JMP ERROR     RESPONDED INSTEAD OF ONLY ONE)
```

```
*
*     CONVERT THE SYSTEM ID TO A NUMBER FROM 0 TO 7
*     TO BE PASSED BACK IN GLOBAL 1P
*
      LDB SYSID     GET THE VALUE TO BE CHECKED
      LDA MD8       SET THE COUNTER TO -8
      STA KT
      LDA DEFTB     INITIALIZE THE TABLE POINTER
      STA DEFS
      CLA           INITIALIZE THE CONVERTER
      STA COUNT
LOOP2 LDA DEFS,I    GET A TABLE ENTRY
      CPA B         DOES IS MATCH
      JMP OVER2     YES GET OUT
      ISZ COUNT     BUMP THE CONVERTER COUNTER
      ISZ DEFS      BUMP THE TABLE POINTER
      ISZ KT        BUMP THE LOOP COUNTER
      JMP LOOP2     IF NOT ZERO GO LOOP
*
*   IF YOU GO THROUGH THE LOOP 8 TIMES IT MEANS THAT MORE
*   THAN ONE BIT WAS SET.  THIS SHOULD NEVER HAPPEN
*
      JMP ERROR     IF YOU GOT HERE IT IS AN ERROR
OVER2 LDA COUNT     PUT THE CONVERTED VALUE IN
      STA SYS#      SYS# TO BE PASSED BACK IN 1P
QUIT  JSB PRTN      PASS THE SYSTEM # AND SYSTEM ID
      DEF *+2       BACK TO FMGXX
      DEF SYS#
      JSB EXEC      RETURN TO THE SYSTEM
      DEF *+2
      DEF EXIT
      HLT 0         HALT IN CASE THE EXIT FAILS
ERROR CMA           SET THE ERROR VALUE OF -1 IN SYS#
      STA SYS#      TO BE RETURNED IN THE GLOBAL 1P
      JMP QUIT
*
EXIT  DEC 6         RETURN TO SYSTEM PARAMETER FOR EXEC
SYS#  DEC -1        PARAMETER TO BE RETURNED IN GLOBAL 1P
SYSID DEC -1        PARAMETER TO BE RETURNED IN GLOBAL 2P
PARM3 DEC 0         PARAMETER TO BE RETURNED IN GLOBAL 3P
PARM4 DEC 0         PARAMETER TO BE RETURNED IN GLOBAL 4P
PARM5 DEC 0         PARAMETER TO BE RETURNED IN GLOBAL 5P
BIT0  OCT 1         SELECT CODE 70B INDICATER (BIT 0)
BIT1  OCT 2         SELECT CODE 71B INDICATER (BIT 1)
BIT2  OCT 4         SELECT CODE 72B INDICATER (BIT 2)
BIT3  OCT 10        SELECT CODE 73B INDICATER (BIT 3)
BIT4  OCT 20        SELECT CODE 74B INDICATER (BIT 4)
BIT5  OCT 40        SELECT CODE 75B INDICATER (BIT 5)
BIT6  OCT 100       SELECT CODE 76B INDICATER (BIT 6)
BIT7  OCT 200       SELECT CODE 77B INDICATER (BIT 7)
MD8   DEC -8        LOOP COUNTER INITIALIZER
KT    BSS 1         LOOP COUNTER
DEFTB DEF BIT0      TABLE POINTER INITIALIZER
DEFS  DEF *         TABLE POINTER (SETUP FROM DEFTB)
COUNT BSS 1
B377  OCT 377       MASK OUT BITS 8-15
      END SYSNO
```

MULTI-STATION TRAPS WITH BASIC/1000D

by Marty Silver/HP Data Systems Division

INTRODUCTION

Multi-Station traps are used on systems where more than one copy of BASIC is running concurrently and each copy of BASIC needs its own trap table. Multi-Station traps are most commonly used on automatic test systems, where BASIC is used as a test programming language. A copy of the trap table is needed for each copy of BASIC that is executing on the system.

With Multi-Station traps a test operator can be conducting a test at the test station terminal, while test programmers can be doing test program development at program development terminals. Each user has his own copy of BASIC and his own trap table. The copies of BASIC can be executing concurrently thereby sharing system resources.

Multi-station test systems are test systems with one computer controlling more than one separate test station. The Multi-Terminal Interface Software (MTIS) provides the multi-station test system capability, as well as the multi-station trap capability. Once a multi-station test system has been setup with multi-station traps, device interrupts from each of the separate test stations can be serviced independent of the other test stations.

For example, suppose a test program executing on test station one of the test system has been setup in the following manner to handle interrupts for HP 9411A Switch Controller.

```
100TRAP 11 GOSUB 7000
110CALL SRQ(L9,16,"DVINT")
 .
 .
7000REM TRAP SERVICE ROUTINE FOR SWITCH CONTROLLER
 .
9999END
```

If the switch controller interrupts, the BASIC program needs to be notified on the interrupt. This occurs through a series of steps, which are listed below:

1. DVR37 takes the interrupt and begans to process it. It finds the name of the device interrupt program, DVINT, in the EQT extension area for the switch controller and schedules it.

2. DVINT finds the station number containing the interrupting switch controller and the trap number assigned to the switch controller. It then sets the assigned trap cell in the trap table for that test station.

3. BASIC checks the trap table before the execution of every statement. It finds the trap cell set, so it starts executing the trap service routine starting at statement 7000.

4. Once the trap service routine is finished executing, BASIC resumes executing the statements of the program at the point of the interrupt.

The following sections explain the software modules involved in communicating the hardware interrupt to the BASIC test program.

DESCRIPTION OF THE MULTI-STATION TRAP SOFTWARE MODULES

Traps are used to interrupt or change the course of BASIC program execution, e.g., to signal some asynchronous data, or to indicate an error condition, or an operator command. BASIC checks the trap table between execution of each statement to determine whether any trap has been set. Multi-Station traps are implemented using several separate software modules: $TRPL5, %IB4A, %DSCHD, %DVIN5, and &TRTB5. $TRPL5, %DVIN5, and &TRTB5 are part of the HP 92425C Multi-Terminal Interface Software. The remaining two modules are included with the HP 92068A RTE-IVB Operating System.

The trap library, $TRPL5, contains the trap setting subroutine (TRPNT), the multi-station trap table (TRTBL), and the trap program (TRAP). The trap library replaces the library normally used in the Multi-User Real-Time BASIC, %BAMLB. The HP-IB library, %IB4A, contains the subroutine (SRQ) used to set up the alarm program name into the EQT extension for HP-IB devices. Each of the multi-station trap software modules will be described in the following sections.

THE MULTI-STATION TRAP TABLE

The multi-station trap table, TRTBL, is a trap table that is configured to allow up to four BASIC programs to execute at different stations simultaneously. Each copy of BASIC will have a station trap table with sixteen trap cells. Included with the sixteen trap cells are four words used by the TRAP program for control purposes. The first word is used to connect a trap table to a copy of BASIC, the second word is not used, the third word is used to hold the priority of the current interrupt, and the last word is used as a flag to specify whether or not there is a need to search the sixteen trap cells. The general form of the trap table is shown following:

```
ASMB,Q,C
      NAM TRTB5,30 92425-1X069 REV.2001 791130<< 4 STN, 16 TRAPS >>
*
      ENT TABL$,STN$,TRTBL,TREND,SRQ.T
      SUP
*
*---------------------------------------------------------------
*    THE STRUCTURE OF THE TRAP TABLE FOR EACH STATION IS BELOW
*
*     WRD                  FUNCTION
*     ---                  --------
*      1           STATION # FOR THIS TRAP BLOCK
*      2           NOT USED CURRENTLY
*      3           PRIORITY OF CURRENT INTERRUPT (1 TO 16, 1=HI)
*      4           FLAG not equal to 0, SEARCH TABLE BEFORE EVERY LINE
*                       equal to 0, DON'T SEARCH TABLE BEFORE EVERY LINE
*
*      5           TRAP CELLS:
*   through           BIT 15  1=TRAP SET/ 0=CLEAR
*     20              BIT 14  1=TRAP IN PROCESS/ 0=CLEAR
*                     BIT 13-0 SEQUENCE NUMBER, 0 IMPLIES NOT ENABLED
*
*---------------------------------------------------------------
*
TABL$ DEC 16        16 ENTRIES/STATION
STN$  DEC 20        TOTAL SIZE OF A STATION'S TABLE
SRQ.T EQU *         DUMMY ENTRY POINT FOR HPIB SUB. DON'T USE!
TRTBL DEF *+1
      REP 80
      NOP
TREND DEF *
      END
```

The entries in the trap table listing are explained below.

TABL$ This entry determines the number of trap cells per station. All stations must have the same number of trap cells.

STN$ This entry is ALWAYS four more than the number of trap cells per station. Four words are used for control purposes.

SRQ.T This entry point is used by the HP-IB subroutine SRQSN(SESLU,TRAP) for indexing into the trap table. This routine is not used on systems with multi-station traps, so the entry point is not used on systems with multi-station traps.

TRTBL Beginning address of the trap table.

REP X This entry specifies the actual size of the table. The value of X is determined by multiplying the value in STN$ by the total number of copies of BASIC executing simultaneously.

TREND Ending address of the trap table.

The four control words for each station trap table are described below.

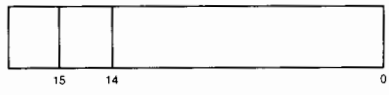
Word 1 This entry is used for connecting the station trap table to a copy of BASIC. The station number is the same as the system logical unit of the terminal the user is logged on to.

Word 2 This entry is not used at this time.

Word 3 This word is used to hold the priority of the current interrupt. When the trap program finds a station trap cell set, it enters that trap cell number into this word.

Word 4 The search table flag is stored in this word. When this word contains a zero the station trap table is not searched. This word will contain a non-zero value when it is necessary for the trap program to search the station trap table.

Trap Cells Each trap cell (see Figure 1) has three separate parts. Bit fifteen is used to signify the state of the trap cell, and bit fourteen is used to signify whether or not the trap service routine is in process or not. The rest of the fourteen bits are used to hold the first statement number of the trap service routine.



BIT 15	1 = TRAP SET	0 = TRAP CLEAR
BIT 14	1 = TRAP IN PROC	0 = TRAP NOT IN PROC
BIT 13-0	SEQ #	0 = NOT ENABLED

Figure 1: Trap Cell

The trap table (see Figure 2) shown is the default table included in the trap library. If the number of trap cells per station trap table or the number of station trap tables needs to be changed, the source for the trap table is provided with the HP 92425C product. The source file is &TRTB5. To change the number of traps per station, change the TABL$ entry, then add four to the new TABL$ value and use this value for the STN$ entry. Finally, update the value in the REP statement. To change the number of station trap tables or the number of copies of BASIC that can execute simultaneously, change the REP statement. After the trap table has been assembled, it must be relocated following relocation of the trap library during the system generation. Relocating the modified trap table after the trap library during the system generation is going to cause one GEN ERR 08 error because of the duplicate module and five GEN ERR 05 errors because of duplicate entry points. Twenty is the maximum number of trap cells per station.

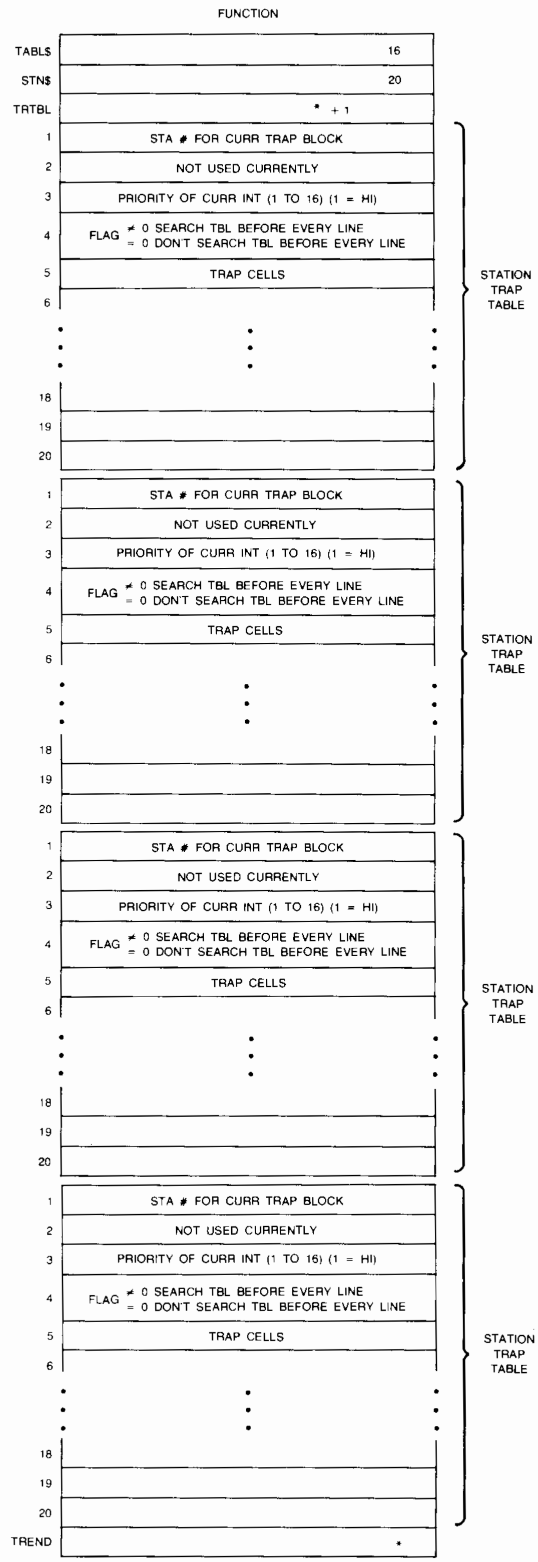


Figure 2: Default Multi-Station Trap Table

THE TRAP PROGRAM

The trap program, TRAP, is called from the BASIC interpreter for all trap table processing. TRAP performs five different operations on the trap table. The user can use the BASIC TRAP statement to specify a particular operation on the trap table. This operation is to set up a trap cell to handle real time interrupts. The TRAP statement is the only way a programmer can specify to BASIC to call the TRAP program. The other four operations of the trap program are inherent in the BASIC interpreter, and are transparent to the user.

The first operation of the trap program is to initialize a station trap table at the beginning of the program execution phase. When the BASIC RUN command is entered, the BASIC interpreter calls the trap program in the following manner.

```
LDA 1           DECIMAL ONE INDICATES AN INITIALIZE REQUEST
JSB TRAP
JMP ERROR       ERROR RETURN, NOT USED WITH THIS CALL
RETURN          IF B REGISTER CONTAINS MINUS ONE, IT MEANS NONE
                   OF THE STATION TRAP TABLES WERE AVAILABLE
```

If one of the station trap tables is available (the first control word of the station trap table containing a zero), the trap program will first claim the table by putting the station number (terminal system logical unit number) in the first word of the table (see Figure 3) and the number of trap cells plus one in the third word. The trap program will initialize the station trap table by putting zeros in the remainder of the words.

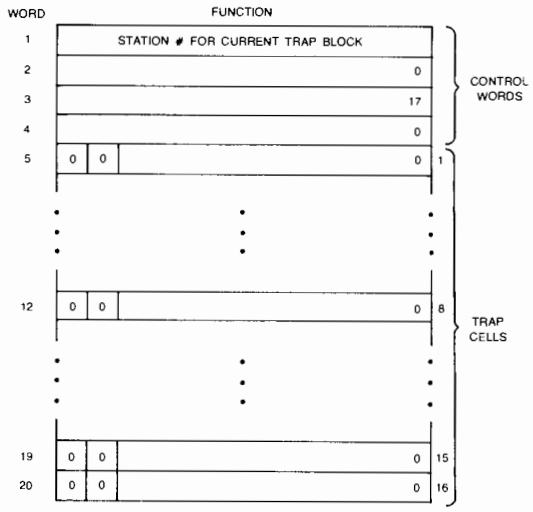


Figure 3: Trap Table Just Initialized

If the trap program returns a minus one in the B register from an initialization request, the BASIC interpreter outputs a trap table full message to the station terminal. This case will occur when more copies of BASIC are trying to execute simultaneously than there are station trap tables available. This means if you plan to have more than four copies of BASIC executing programs simultaneously, you need to modify the trap table to increase the number of station trap tables.

The BASIC TRAP statement, shown below, is used to associate a task with a trap cell.

```
TRAP 8 GOSUB 8000
```

The TRAP statement associates the task (trap service routine) starting at statement number 8000 with station trap cell number 8 (see Figure 4). When BASIC executes the TRAP statement it calls the trap program in the following manner.

```
LDA -TRP        MINUS THE TRAP NUMBER
LDB STMT        STATEMENT NUMBER SPECIFIED IN TRAP STATEMENT
JSB TRAP
JMP ERROR       ERROR RETURN
RETURN          NORMAL RETURN
```

The trap program makes sure a valid trap number and a valid statement number were specified in the TRAP statement. The statement number specified in the TRAP statement cannot be used in more than one station trap cell. Each station trap cell must have its own unique statement number. Specifying a statement number already used in another station trap cell would cause the trap program to exit through the error path. The statement number specified is put into the lower bits (bits 0-13) of the station trap cell specified in the TRAP statement. The search table flag is set to a non-zero value, so the station trap table is searched before the execution of the next program statement.

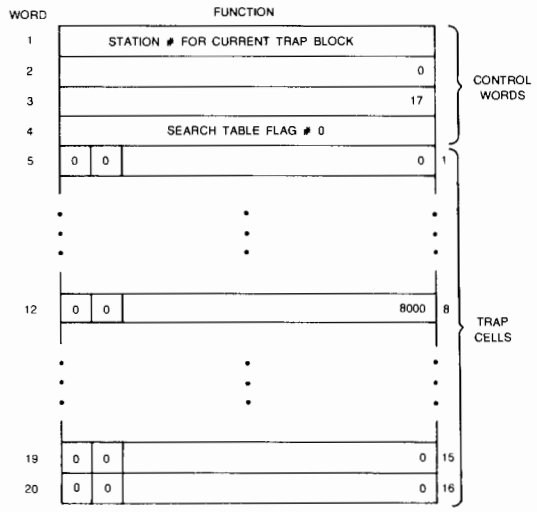


Figure 4: Trap Table After BASIC Trap Statement

The most often performed operation of the trap program is to check the station trap table before executing each statement in the BASIC program. The trap program checks to see if any of the trap cells have been set. A trap cell is set when bit fifteen of the trap cell is one. The trap program performs this operation when called in the following manner from BASIC.

```
LDA -1000       MEANS TO CHECK THE STATION TRAP TABLE
LDB PTR         POINTER (PTR) TO NEXT STATEMENT TO BE EXECUTED
JSB TRAP
JMP ERR         ERROR RETURN
RETURN          NORMAL RETURN
```

The trap program first finds the station trap table, which was initialized at the beginning of the execution phase. It checks the search table flag, which is contained in the fourth control word. If the search table flag is zero, the trap program does not check the trap cells. The trap program returns the following values to BASIC specifying the program is not to be interrupted.

- A register contains a minus one (−1)
- B register contains the address of the statement to be executed next

The trap program will return the same values shown above to BASIC if the search table flag is set, and none of the trap cells have been set. Before the trap program returns to BASIC, it clears the search table flag. This way time will not be wasted searching the table between every BASIC statement until a trap cell has been set (by the trap setting program TRPNT) or the BASIC TRAP statement is executed.

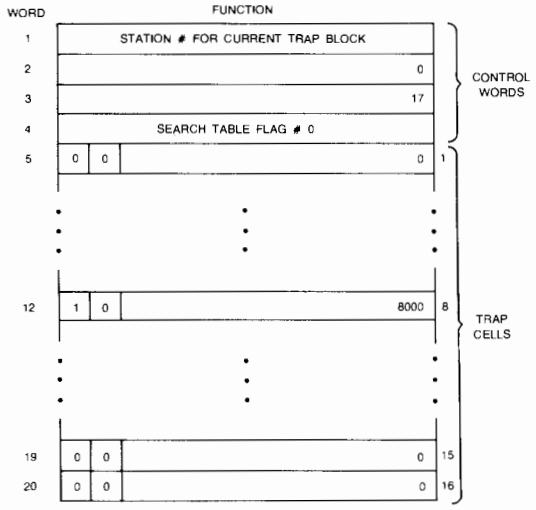


Figure 5: Trap Table After TRPNT Sets Trap Cell

The BASIC TRAP statement and the trap setting program TRPNT both set the search table flag to a non-zero value, so the trap table will be searched before the next BASIC statement (see Figure 5).

If the TRAP program finds the search table flag set, it will search the station trap cells. The current priority word is checked to see how much of the trap table is to be searched. The current priority word will contain the number of trap cells plus one if the entire table is to be searched. If the current interrupt being serviced is for station trap cell eight, the current priority word will contain an eight. This means only the trap cells of higher priority will be searched. The lower the number of the trap cell the higher its priority. At this point the trap table may be in one of many possible states. The trap program will search part or all of the trap table depending on the current interrupt priority looking for a station trap cell set (bit 15 equal one). If a trap cell is set, bit 15 is cleared and the in progress bit (bit 14) is set to one (see Figure 6). The trap program will then enter the priority of the set trap cell into the third control word of the station trap table. The priority is equal to the number of the set trap cell. The trap program then returns the following values to BASIC:

- A register contains statement number from the station trap table cell
- B register contains statement number that would have been executed if no interrupt

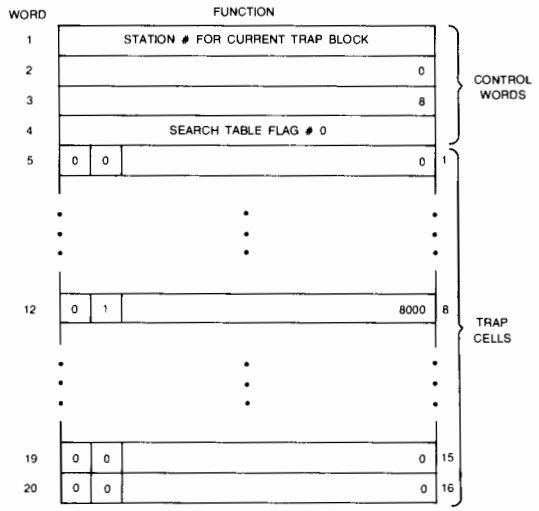


Figure 6: Trap Table after BASIC finds Trap Cell Set

BASIC will then begin to process the trap service routine starting at the statement number returned in the A register. If BASIC was currently executing a trap service routine, it is queued while the returned trap service routine is executed. BASIC knows the more recent trap service routine is of higher priority because the trap program only checks station trap cells of higher priority.

While executing a trap service routine BASIC is still checking the trap table before executing each statement. If the trap service routine is not interrupted by a trap of higher priority, the last statement of the trap servce routine will be executed. If the last statment of the trap service routine is a BASIC RETURN statement, another operation of the trap program will be executed. BASIC will call the trap program in the following manner.

```
LDA -256        MEANS END OF TRAP SERVICE ROUTINE
LDB -STMT       MINUS STATEMENT NUMBER TO BE EXECUTED NEXT
JSB TRAP
JMP ERROR       ERROR RETURN (NOT USED WITH CALL)
RETURN          NORMAL RETURN
```

The current interrupt priority word is used to find the trap cell associated with the trap service routine that just completed. The trap in progress bit is cleared, and the trap program begins to look for the next highest priority trap cell with its trap in progress bit set. If it finds a station trap cell with its trap in progress bit set, the current priority word is updated to contain that trap cell number. If no trap cell has its trap in progress bit set, the current priority word is set to the lowest priority. In either case, the search table flag is set to a non-zero value, so the station trap table will be searched before the execution of the next statement.

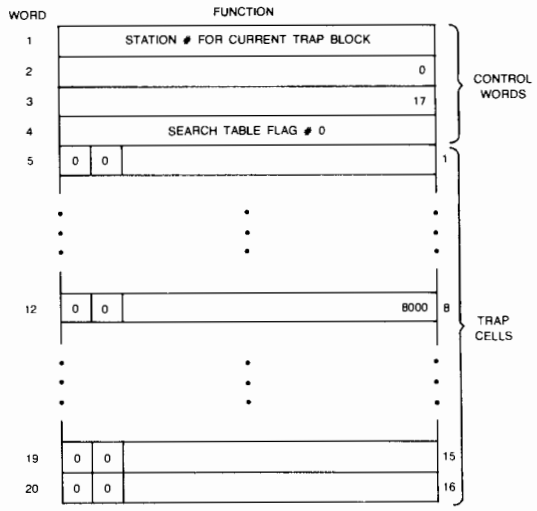


Figure 7: Trap Table After BASIC RETURN Statement in Service Routine

The last operation of the trap program is to free up a station trap table at the end of the execution phase. This occurs when a BASIC program completes execution, not to be confused with the processing of the BASIC BYE command. The TRAP routine is called in the following manner.

```
LDA 2           MEANS A TERMINATE REQUEST
JSB TRAP
JMP ERROR       NOT USED WITH THIS CALL
RETURN          NORMAL RETURN
```

The trap program will find the station trap table and zero out the first control word, which contains the station number. This makes the station trap table available for other users.

THE TRAP SETTING PROGRAM TRPNT

TRPNT is a trap setting routine used by interrupt handling programs to set bit 15 of a specified trap cell in the trap table. The trap number and the station number are passed to TRPNT in the following manner.

```
LDA TRP         TRAP NUMBER TO BE SET
LDB STN         SESSION LU OF TERMINAL
JSB TRPNT
JMP ERROR       ERROR RETURN
RETURN          NORMAL RETURN
```

TRPNT finds the station number from the passed session logical unit. Using the station number, TRPNT finds the proper station trap table. Once the validity of the trap number has been checked, TRPNT sets the search table flag to a non-zero value and sets bit 15 of the specified trap number (see Figure 5). TRPNT does not check to see if the trap cell contains a statement number. If the trap cell does not contain a statement number, the trap program will ignore the interrupt.

DEVICE INTERRUPT HANDLING PROGRAM DVINT

DVINT is a general purpose interrupt trap setting program. It is a part of the HP 92425C product, and the source filename is &DVIN5. It may be used with both general purpose and HP-IB devices. It contains a table of 255 entries, one per system logical unit. The table is used to map the system LU of a device to a particular trap cell number. A sample of the table is shown below.

```
LU1    DEC 0
LU2    DEC 0
 .       .
 .       .
LU255  DEC 0
```

The source of DVINT must be modified to set up the mapping between the trap cell and the system LU. For example, changing "LU80 DEC 0" to "LU80 DEC 10" in DVINT's table will cause trap cell 10 to be set if system LU 80 interrupts. The source for DVINT must be assembled and loaded as a permanent program, so its ID segment will exist when it is needed to service an interrupt.

Once DVINT has been scheduled, it finds the station number and obtains the trap cell number from its internal table. If a HP-IB device interrupts, its driver schedules DVINT and passes it the status word, subchannel of the device, and the EQT number of the HP-IB card. Using the EQT number and the subchannel of the device to obtain the system LU from the device reference table, DVINT is able to find the station number from the MTIS device reference table extension. If a DVM72 device interrupts, DVINT uses passed EQT word 4 and the EQLU routine to find the system LU of the interrupting device. DVINT is then able to find the station number from the MTIS device reference table extension.

It then calls TRPNT to set the proper trap cell. The next time the trap program searches the trap table it will find the trap cell set. TRPNT is called in the following manner.

```
LDA TRAP NUMBER
LDB STATION NUMBER
JSB TRPNT
RETURN1     (ERROR RETURN)
RETURN2     (SUCCESSFUL RETURN)
```

DRIVER INTERFACES FOR SELECTING DVINT AS THE INTERRUPT PROGRAM

On an interrupt the HP-IB driver DVR37, looks into the EQT extension area of the interrupting device for an alarm program. This alarm program is DVINT. The SRQ routine is used to put the name of the DVINT program into the EQT extension area for the interrupting device. SRQ is a part of the HP-IB library, %IB4A,

```
CALL SRQ( SESLU, IV, "DVINT" )
CALL SRQ( SESLU, IV, 0 )

     SESLU = Session LU of the interrupting device.
     IV    = Arbitrary value passed (not used by DVINT).
     DVINT = Alarm program to be scheduled on interrupt.
     0     = Remove alarm program name from EQT ext. area
```

Where as the SRQ routine is used for the HP-IB devices the DSCHD routine is used for the general purpose driver, DVM72. DSCHD is called from BASIC in the following manner.

```
CALL DSCHD( SESLU, 3, "DVINT" )
CALL DSCHD( SESLU, 3, 0 )
```

Instead of putting the program name into the EQT extension area, DSCHD places the ID segment address of the program into EQT13 for the device. On an interrupt from the device, specified by the session logical unit, the driver will schedule the alarm program whose ID segment address is in EQT13.

USING MULTI-STATION TRAPS IN A BASIC PROGRAM

All of the pieces for using multi-station traps have been talked about. To effectively use multi-station traps all of the pieces have to be brought together in an example. The following numbered steps in the example refer to the numbers in Figure 8.

1. The BASIC RUN command causes the trap program to obtain and initialize a trap table (see Figure 1). BASIC then begins to execute the program.

2. When BASIC executes the TRAP statement, the TRAP program puts the sequence number in the specified trap cell and sets the search trap table flag word to a non-zero value (see Figure 2).

3. BASIC automatically checks the trap table for set trap cells using the TRAP program. The TRAP program first checks the search trap table flag. At this point the search trap table flag is non-zero, but no interrupts have occurred. The TRAP program searches the trap table and finds no trap cells set (bit fifteen set), so it clears the search table flag (set to zero).

4. Again BASIC is automatically checking the trap table for set trap cells using the TRAP program. At this point no interrupts have occurred and the search table flag was cleared in step 3. This means, the TRAP program checks the search table flag and finds it cleared. The TRAP program returns to BASIC indicating no interrupts occurred because the search table flag was cleared.

5. The program makes a call to SRQ to set up the alarm program for the device that may interrupt. The HP-IB driver does the set up by modifying the equipment table for the card the device is attached to and the equipment extension area for the device.

   a. The HP-IB driver moves the the alarm program name into words two, three, and four of the equipment extension area for the device specified by the session logical unit parameter of the SRQ call.

   b. The HP-IB driver then sets the SRQ interrupt arming flag in the equipment table for the HP-IB card which the specified device is controlled by.

6. The device (session logical unit; "L" in SRQ call) sends a hardware interrupt to the HP-IB card via the SRQ line. At this point, the alarm program has been set up for the interrupting device.

   a. The HP-IB driver gets the status from the device pulling the SRQ line and schedules the alarm program (DVINT) passing it the following parameters.

      PARAMETER #1 — status of interrupting device
      PARAMETER #2 — device bus address
      PARAMETER #3 — device equipment address
      PARAMETER #4 — arbitrary value passed by SRQ call

   b. The HP-IB driver also sets the alarm program scheduling active bit.

7. The alarm program DVINT uses the passed device equipment address and bus address to determine the system logical unit number of the interrupting device.

   a. Once the system logical unit number has been determined, DVINT can find the trap cell number assigned to the interrupting system logical unit number by looking in its internal table.

   b. DVINT uses the trap setting routine TRPNT to set bit fifteen of the trap cell specified in its internal table. TRPNT also sets the search table flag to a non-zero value.

8. The TRAP routine, which is always checking trap table for BASIC, finds the search table flag set. It then searches the trap cells of higher priority than specified in the current priority word. At this point all trap cells get searched because no current interrupt is being serviced.

   a. The TRAP routine finds the trap cell eight set. It clears the trap set bit (bit fifteen) and sets the trap in process bit (bit fourteen).

   b. The TRAP routine updates the current priority word with the value of the current interrupt trap cell number (trap cell eight).

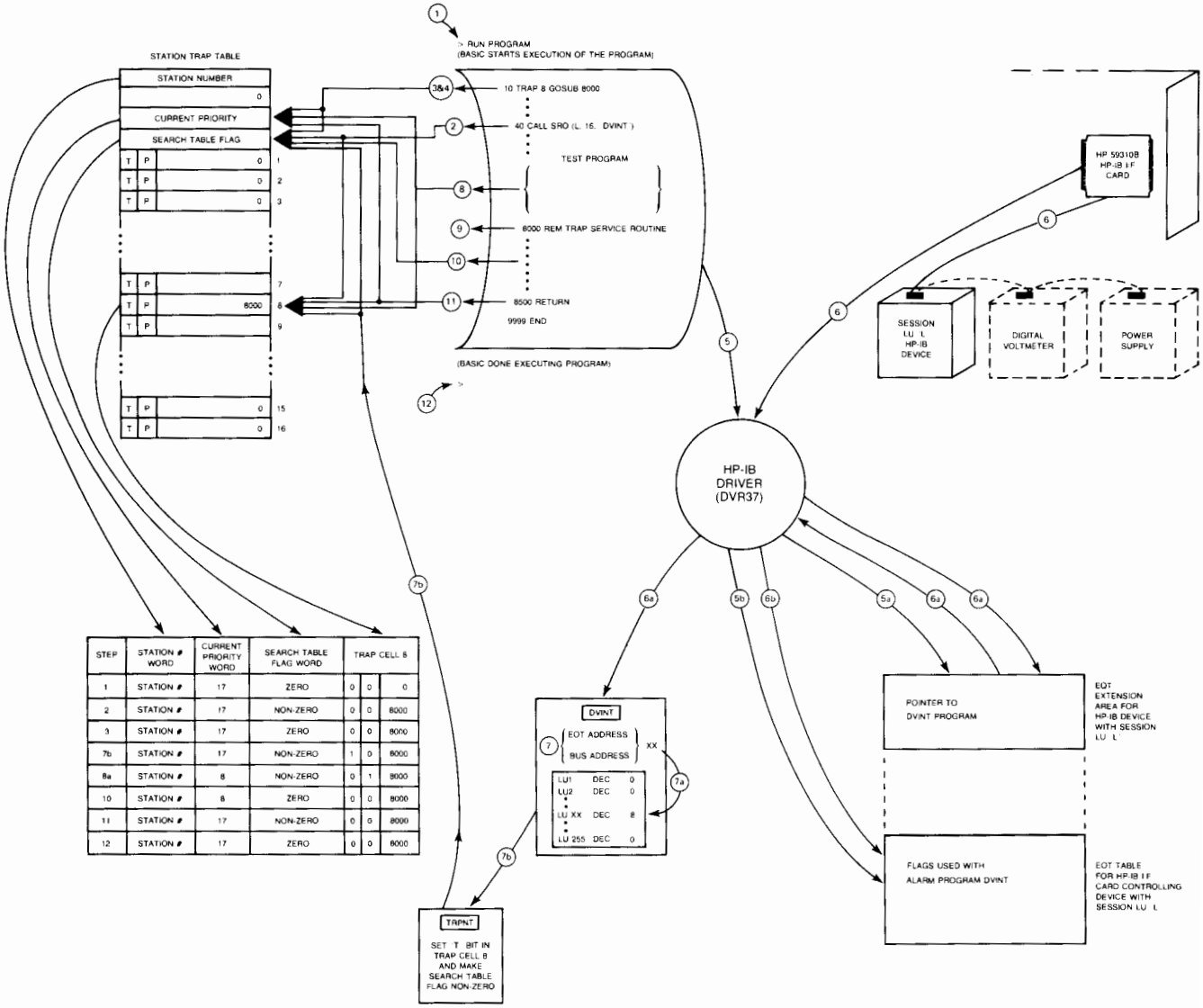
   c. The TRAP routine returns the first statement of the trap service routine it obtains from the lower fourteen bits of trap cell eight.

9. BASIC begins to execute the trap service routine for the device assigned to to trap cell eight.

10. Between each statement of the trap service routine, BASIC is checking the trap table. The search table flag is still set after the TRAP routine found trap cell eight set, so the trap table is searched. The TRAP routine searches only the lower seven trap cells because of the value in the current interrupt word. At this point no other interrupts have occurred, so TRAP clears the search table flag after finding no trap cells set.

11. BASIC encounters the RETURN statement in the trap service routine for trap cell eight. BASIC uses the TRAP routine to clear the in process bit of trap cell eight and searches for the next trap cell with its in process bit set. At this point no other trap cells have their in process bits set. The search table flag is set, in case trap cells of lower priority than trap cell eight had been set during the processing of the service routine, and the priority word is set to the lowest value. (If TRAP had found a in process bit set the current priority word would have been set to the value of that trap cell.)

12. BASIC returns to the point of the program that was executing at the time the interrupt was serviced. If no more interrupts occur, the program will finish executing. At this point, BASIC uses the TRAP routine to release the trap table by zeroing out the first control word.



STEP	STATION # WORD	CURRENT PRIORITY WORD	SEARCH TABLE FLAG WORD	TRAP CELL 8		
1	STATION #	17	ZERO	0	0	0
2	STATION #	17	NON-ZERO	0	0	8000
3	STATION #	17	ZERO	0	0	8000
7b	STATION #	17	NON-ZERO	1	0	8000
8a	STATION #	8	NON-ZERO	0	1	8000
10	STATION #	8	ZERO	0	0	8000
11	STATION #	17	NON-ZERO	0	0	8000
12	STATION #	17	ZERO	0	0	8000

Figure 8: Trap Processing Example

The previous example shows the details of the trap processing, but doesn't give an example of how multi-station traps work. The following program and explanation describes the use of traps on a system with two stations.

```
100  CALL SRQ( 21, 16, "DVINT" )
110  TRAP 11 GOSUB 7000
120  FLAG = 0
130  PRINT " WAITING FOR SRQ FROM INSTRUMENT "
140  IF FLAG = 1 GO TO 1000
150  GO TO 130
1000 PRINT " END OF MULTI-STATION TRAP PROGRAM "
1010 STOP
7000 PRINT " TRAP ROUTINE FOR TRAP CELL 11 ENTERED "
7010 FLAG = 1
7020 RETURN
9999 END
```

Statement 100 puts the name of the alarm program into the EQT extension area for the device with session LU 21. Statement 110 puts the number 7000 into the lower fourteen bits of trap cell 11. The device with session LU 21 must have its system LU mapped to trap cell 11 in the alarm program DVINT. To demonstrate the use of multi-station traps, the sample program must be used on a system with at least two stations. For the example, assume session LU 21 on station one is mapped to system LU 71. While session LU 21 on station two is mapped to system LU 81. Both system LUs 71 and 81 are mapped to trap cell 11 in the alarm program DVINT. A copy of the sample program can be executing on both stations simultaneously and receive interrupts from session LU 21. As each device interrupts, DVR37 finds and schedules the alarm program DVINT. DVINT finds the station number (station one for system LU 71 and station two for system LU 81), then finds the trap cell number from its table. DVINT uses TRPNT to set bit fifteen of trap cell 11 in the trap table for station one and in the trap table for station two. (Note: The station number is actually equal to the system logical unit number of the terminal, but for ease of explanation station one and station two were used in the description.)

Multi-station traps are necessary in automatic test systems where two or more test stations are controlled by one computer system. They allow test programs to handle interrupts from test station instruments in real time. In RTE-IVA, automatic test systems had multi-station traps with keyboard traps. In RTE-IVB it is up to the user to implement keyboard traps in the automatic test systems. This is the subject of the next section.

USING KEYBOARD TRAPS IN RTE-IVB AUTOMATIC TEST SYSTEMS

Keyboard traps are used by the test operator to alter the progress of a test program during its execution. The test operator strikes the terminal keyboard to get the break-mode prompt. In RTE-IVA the test operator would enter a number between one and seven depending on the keyboard trap desired. In RTE-IVB the user must write a program to set a trap upon an interrupt from a terminal keyboard. The program must obtain the station number, check the keyboard trap requested, and use TRPNT to set the trap cell in the trap table. Below is a copy of the keyboard trap setting program used for our RTE-IVB automatic test training system.


```
ASMB,R,Q
      HED   << "SET" THE KEYBOARD TRAP SETTING PROGRAM >>
      NAM SET,2,41
*
*-------------------------------------------------------
*
*     NAME: SET
*   SOURCE: &SET:RT
*    RELOC: %SET:RT
*    PRGMR: MARTY SILVER
*     DATE: 11/11/80
*
*-------------------------------------------------------
*
*     ENTRY POINT:
*
      ENT SET
*
*     EXTERNAL REFERENCES:
*
      EXT GETST, ERROR, EXEC, LOGLU, TRPNT, $PARS
*
*     ERROR 1 - INVALID TRAP CELL NUMBER PASSED TO PROGRAM
* ONLY TRAP CELLS OF 1 TO 7 ARE VALID
*
*     ERROR 2 - TERMINAL NOT IN SESSION.  PROGRAM WILL NOT
* SET ANY TRAPS UNLESS TERMINAL IS IN SESSION
*
*     ERROR 3 - TRAP SETTING SUBROUTINE "TRPNT" ENCOUNTERED
* PROBLEMS IN SETTING THE TRAP CELL
*
*     THE SOURCE FOR THIS PROGRAM CAN BE ASSEMBLED WITH THE
*     RTE-IVB ASSEMBLER, AS SHOWN BY THE FOLLOWING EXAMPLE:
*
* :RU,ASMB,&SET,6,%SET
*
*     LOAD THIS PROGRAM AS A PERMANENT PROGRAM WITH THE

*     THE RTE-IVB LOADR, AS SHOWN BY THE FOLLOWING EXAMPLE.
*
* RU,LOADR:IH
* /LOADR:   OP,SS
* /LOADR:   OP,PE
* /LOADR:   RE,%SET
* /LOADR:   EN
*
*-------------------------------------------------------
*
*     GET KEYBOARD TRAP CELL NUMBER TO BE SET
*
SET   NOP               ENTRY POINT
      JSB               GETST  GET THE RUNSTRING
      DEF *+4           RETURN POINT DEFINED
      DEF IRBUF         RETURNED STRING BUFFER
      DEF ILEN          NEGATIVE CHARACTER LENGTH OF STRING
      DEF ILOG          TRANSMISSION LOG
*
*     PARSE THE STRING RETURNED FROM GETST INTO PARAMETERS
```

```
*
      LDA IRBAD          ADDRESS OF STRING TO BE PARSED
      LDB ILOG           POSITIVE CHARACTER LENGTH OF STRING
      JSB $PARS          PARSE THE STRING
      DEF IBUF           RESULTS OF PARSE OPERATION
*
*     IS THE FIRST PARAMETER A NUMERIC VALUE
*
      LDA IBUF           GET FLAG WORD
      CPA D1             IS IT NUMERIC?
      JMP CHK            YES, NOW CHECK FOR A VALID NUMBER
      JMP BADPM          NO, OUTPUT ERROR CODE 1
*
*     CHECK TO SEE IF VALID KEYBOARD TRAP CELL   (1-7)
*
CHK   LDA IBUF+1         GET THE TRAP CELL NUMBER
      SSA                IS IT NEGATIVE?
      JMP BADPM          YES, OUTPUT ERROR CODE 1
*
      SZA,RSS            IS IT EQUAL TO ZERO?
      JMP BADPM          YES, OUTPUT ERROR CODE 1
*
      ADA M8             IS NUMBER GREATER THAN 7
      SSA,RSS            CHECK SIGN BIT
      JMP BADPM          YES, OUTPUT ERROR CODE 1
*
*     GET THE STATION NUMBER USING THE LOGLU
*     (STATION   IS EQUAL TO SYSTEM LU   OF TERMINAL)
*
      JSB LOGLU          GET STATION NUMBER
      DEF *+2            DEFINED RETURN POINT
      DEF LUSYS          RETURNED SYSTEM LU
      STA SESLU          SAVE RETURNED SESSION LU
*
*     CHECK TO SEE IF TERMINAL IS IN SESSION
*     (LUSYS IS NEGATIVE IF TERMINAL NOT IN SESSION)
*
      LDA LUSYS          GET THE SYSTEM LU NUMBER
      SSA                IS THE TERMINAL IN SESSION
      JMP NOSES          NO, OUTPUT ERROR CODE 2
*
*     SET THE SPECIFIED KEYBOARD TRAP CELL USING
*     MTIS SUBROUTINE "TRPNT"
*
      LDA IBUF+1         PASS TRAP CELL   TO BE SET
      LDB SESLU          PASS STATION NUMBER
      JSB TRPNT          SET THE SPECIFIED TRAP CELL
      JMP TRPER          ERROR RETURN FROM TRPNT
*
*     TERMINATE PROGRAM
*
EXIT  JSB EXEC           CALL TO SYSTEM TO TERMINATE
      DEF *+2
      DEF 06             EXEC CODE OF 6
*
*     ERROR ROUTINES
```

```
*
BADPM JSB ERROR          INVALID TRAP CELL   SPECIFIED
      DEF *+3            DEFINE RETURN POINT
      DEF IERR1          ERROR CODE  1
      DEF IERMS          PROGRAM NAME
      JMP EXIT           TERMINATE PROGRAM
*
NOSES JSB ERROR          TERMINAL NOT IN SESSION
      DEF *+3            DEFINE RETURN POINT
      DEF IERR2          ERROR CODE  2
      DEF IERMS          PROGRAM NAME
      JMP EXIT           TERMINATE PROGRAM
*
TRPER JSB ERROR          "TRPNT" ERROR
      DEF *+3            DEFINE RETURN POINT
      DEF IERR3          ERROR CODE  3
      DEF IERMS          PROGRAM NAME
      JMP EXIT           TERMINATE PROGRAM
*
*     CONSTANTS AND TEMPORARY STORAGE
*
IBUF  BSS 33             ARRAY RETURNED FROM $PARS
IRBAD DEF IRBUF          ADDRESS OF IRBUF ARRAY
IRBUF BSS 20             ARRAY USED BY ROUTINE GETST
ILOG  NOP                TRANSMISSION LOG
LUSYS NOP                SYSTEM LU RETURNED FROM LOGLU
SESLU NOP                SESSION LU RETURNED FROM LOGLU
M8    DEC -8             USED CHECKING FOR VALID TRAP CELL
06    OCT 6              EXEC CALL CODE OF 6
D1    DEC 1              USED TO CHECK FOR NUMERIC PARAMETER
IERR1 DEC 1              ERROR CODE  1
IERR2 DEC 2              ERROR CODE  2
IERR3 DEC 3              ERROR CODE  3
IERMS DEC 5              PROGRAM NAME
      ASC 3,SET           FOR USE IN OUTPUTTING ERROR CODES
ILEN  DEC -40            MINUS CHARACTER LENGTH OF
*                         STRING BUFFER
      END SET
```

This program is scheduled from the break-mode prompt after BASIC has been scheduled. It is necessary for BASIC to be running, so a trap table is initialized and attached to the station. The test operator can interrupt the executing BASIC test program by striking the keyboard and getting the break-mode prompt. The test operator will then run the keyboard trap setting program SET passing it a keyboard trap cell number to set. For example, the following entry would set keyboard trap cell number three.

```
S=20 COMMAND ?RU,SET,3
```

The preceding command is what the test operator would enter in response to the break-mode prompt. The program SET will check the trap number to see if it is a number between one and seven. An error one if an invalid trap number is requested using the HP 92425C MTIS error routine ERROR. If the trap number is valid, SET will check to see if the test operator is operating within session. An error two if the test operator is out of session. SET will use the HP 92425C MTIS trap setting program TRPNT to set the requested trap cell if the test operator is operating in session. If TRPNT encounters any problems in setting the trap cell, SET will output an error three. After the trap cell has been set, BASIC will find the trap set and begin to execute the interrupt service routine for that trap cell. The following BASIC program can be used to test the operation of the keyboard trap setting program.

```
10   TRAP 1 GOSUB 1000
20   TRAP 2 GOSUB 2000
30   TRAP 3 GOSUB 3000
40   TRAP 4 GOSUB 4000
50   TRAP 5 GOSUB 5000
60   TRAP 6 GOSUB 6000
70   TRAP 7 GOSUB 7000
80   LET F = 0
90   PRINT " TRY THE KEYBOARD TRAP SETTING PROGRAM !"
100  IF F = 1 GOTO 120
110  GOTO 80
120  PRINT
130  PRINT " KEYBOARD TRAP SETTING PROGRAM WORKED ! "
140  PRINT
150  PRINT "            END OF PROGRAM !     "
160  STOP
1000 PRINT "       KEYBOARD TRAP ONE WORKS ! "
1010 WAIT (3000)
1020 FLAG = 1
1030 RETURN
2000 PRINT "       KEYBOARD TRAP TWO WORKS ! "
2010 WAIT (3000)
2020 FLAG = 1
2030 RETURN
3000 PRINT "       KEYBOARD TRAP THREE WORKS !      "
3010 WAIT (3000)
3020 FLAG = 1
3030 RETURN
4000 PRINT "       KEYBOARD TRAP FOUR WORKS !"
4010 WAIT (3000)
4020 FLAG = 1
4030 RETURN
5000 PRINT "       KEYBOARD TRAP FIVE WORKS !"
5010 WAIT (3000)
5020 FLAG = 1
5030 RETURN
6000 PRINT "       KEYBOARD TRAP SIX WORKS ! "
6010 WAIT (3000)
6020 FLAG = 1
6030 RETURN
7000 PRINT "       KEYBOARD TRAP SEVEN WORKS !      "
7010 WAIT (3000)
7020 FLAG = 1
7030 RETURN
9999 END
```

This BASIC program will allow the user to test all of the keyboard traps. While the program is in the loop the user can get break-mode by striking the keyboard, and run the program SET to set a keyboard trap. BASIC will find the trap set and start executing the proper trap service routine.

This program gives the test operators the use of keyboard traps, if they have a capability level of 30. It requires a capability of at least 30 to be able to run a program from break-mode. Many users of automatic test systems set the capability of test operators at 10, which means the test operators can not run SET to set any keyboard traps. The user can change the capability level for the break-mode RUN command from 30 to 10 by altering the system capability table $CMND. The source for $CMND is part of the HP 92068A RTE-IVB product and is contained in the file &$CMND. Pages J-12 through J-15 of Appendix J in the RTE-IVB System Manager's Manual give the directions for altering the capability level of system and break-mode commands. The following changes were made to our automatic test system to allow test operators with capabilites of 10 to run programs from break-mode.

Original &$CMND

```
          .
          .
        ASC 1,PR
        OCT 0
*
L.30    ASC 1,RU
        OCT 0
        ASC 1,OF
        OCT 40000
          .
          .
        ASC 1,L2
        OCT 0
*
L.10    ASC 1,FL
        OCT 0
        ASC 1,RS
        OCT 0
          .
          .
```

Modified &$CMND

```
          .
          .
        ASC 1,PR
        OCT 0
*
L.30    ASC 1,OF
        OCT 40000
          .
          .
        ASC 1,L2
        OCT 0
*
L.10    ASC 1,RU
        OCT 0
        ASC 1,FL
        OCT 0
        ASC 1,RS
        OCT 0
          .
          .
```

The modified &$CMND file was assembled and relocated after the operating system modules during the generation. This solution allows the test operator to run the program set because it is loaded as a permanent program. Any programs the test operator is not allowed to run should be saved as Type 6 files and not RPed during boot up or log on.

Multi-station traps have been used in automatic test systems for a long time. They allow the user to separate two or more test stations and still have the capability to handle real time interrupts. Keyboard traps give the user some added flexibility in designing their BASIC test programs. Best of luck in the use of multi-station traps.

USING MEMORY BEHIND YOUR FORTRAN PROGRAM

by John Pezzano/HP El Paso

RTE systems (including RTE-IVB, RTE-L/XL, and RTE-M permit the user to use the space between the end of the program and the end of the partition. For the Assembly programmer, this is an easy job, but in the past this has been virtually impossible for the FORTRAN programmer unless he wrote an assembly interface between his main program and his subroutines. I have developed a general purpose routine written in Assembly which does not require the knowledge of Assembly writing to use but will interface between virtually any FORTRAN routines to give the FORTRAN programmer the dynamic memory use now afforded only to the Assembly programmer.

WHY USE THIS MEMORY?

When RTE runs a program, rarely does the program fill all the space to the end of a page boundary. Normally this space is wasted. In addition, the LOADR can be requested to oversize the program beyond what is required. In fact, many HP programs, including the LOADR itself, require this oversizing. This space can have many uses. Programs such as LOADR, FTN4, ASMB, etc. use this space for table areas. A user program might want this space to save temporary information, rather than saving it on the disc which would be much slower. However, the most common and beneficial use of this area in most circumstances would be for the file management Data Control Block (DCB).

In a program using files, a DCB of at least 144 words must be used for the temporary storage of buffers to/from the disc. Generally, the larger the DCB defined, the better will be the program performance. However, sometimes the user does not know if there will be a large enough partition to hold a larger DCB, so the minimum size is defined. To change the size would require modifying the program, recompiling, and reloading it which is usually not worth the effort. However, by using the space between the end of the program and the end of the partition, the size of the DCB is dependent only on the size specified at load time or changed with the RTE-IVB "SZ" command.

FINDING OUT HOW MUCH SPACE IS AVAILABLE

There are a number of different calls in different RTE systems that will tell the user how much space is available and how long it is. However, there is one call common to all current systems that will give this. It is the LIMEM call, used as follows:

```
CALL LIMEM (0,FWAM,LEN)
```

where:

0	no meaning in RTE-IVB. Required for compatability with RTE-M and RTE-L/XL.
FWAM	is the address of the first word after the main program or the largest segment in a segmented program.
LEN	is the number of words between the FWAM and the end of the partition.

It is obvious, therefore, that if we know where our array can be and how big it is, we should be able to use it. But there is a catch!

WHY CAN'T WE USE LIMEM IN FORTRAN?

Theoretically, we should be able to call LIMEM to get the array address and length and call a subroutine with these values. By dimensioning the array to 1 and using the length as the actual size, the subroutine should work. For example,

```
PROGRAM MAIN
CALL LIMEM (0,IBUF,LEN)
CALL SUB (IBUF,LEN)
     .
     .
     .
SUBROUTINE SUB (IBUF,LEN)
INTEGER IBUF(1)
CALL OPEN(IBUF,IERR,6HFILENM,0,ISC,ICR,LEN)
      .
      .
      .
```

Alas, this will not work! The problem is that instead of passing the address of IBUF to the subroutine as an address, the main passes the address of a local variable which contains as a value the address. There is no way in FORTRAN that I know of to get around this problem. For the Assembly programmer, the value can be passed as an address.

THE SOLUTION

The programmer who knows Assembly can call an Assembly subroutine which in turn calls another FORTRAN subroutine or does it in Assembly. However, if the user does not know Assembly, this method is not much use. In addition, the user would have to write an Assembly routine for each different requirement.

As an alternative, I have written an Assembly routine which will call any FORTRAN subroutine whose address is passed to it, pass up to 15 parameters to it (only the first two, array address and length, are restricted) all without any modification. It therefore can be used as a general purpose routine for passing the extra memory array to any FORTRAN subroutine.

Subroutine PASST, as it is called, requires only the following:

1. The caller must define the callee as a BLOCK COMMON name.
2. The caller must pass this name as the first parameter to PASST which will be replaced by the array address.
3. The caller must provide a second (dummy) parameter which will be replaced by the length.
4. The callee must have the first two parameters defined as the array and length respectively. The other parameters are optional.
5. The callee must define the array as an array.
6. The callee must not write past ARRAY(LEN).

There are no other restrictions. Other parameters if used, can be of any type or size. Since restrictions 5 and 6 apply to any passed array, there really are only four hard requirements.

An simple demonstration program will illustrate this:

```
FTN,L
      PROGRAM MAIN
      COMMON/SUB/SUB
      CALL PASST(SUB,SUB,1,2,3,4,5,6,7,8,9)
      END
      SUBROUTINE SUB(I,L,I1,I2,I3,I4,I5,I6,I7,I8,I9)
      INTEGER I(1)
      WRITE (1,10) I1,I2,I3,I4,I5,I6,I7,I8,I9,L
      DO 34 J=1,L
34    I(J)=J
10    FORMAT (9I6)
      END
```

HOW DOES IT WORK?

For those sophisticated enough to understand Assembly and those that wish to use it, a listing of PASST follows this article, but here is a brief explanation.

By defining SUB as a BLOCK COMMON name, FORTRAN considers it to be an external variable. We pass this to PASST, thus passing the address of the subroutine we want to call. This is required since PASST has no idea what the name of the routine is. (When a program is loaded, addresses, not names are used.) The second parameter is just a placeholder for the length. PASST picks up the parameters. The routine .ENTR will do this for us and prevent overwriting if we are passed too many parameters. PASST then picks up the address of the subroutine and sets it up for a call to it. Next, a call to LIMEM is made, getting the starting address and length of the array. We put the starting address into the call to the subroutine (which is the one thing FORTRAN cannot do) and call it. Upon return, we do our own return back to the caller. VOILA!

We can now call any subroutine through PASST.

```
1
 PAGE 0001 #01                          1:03 PM  MON., 13  JULY, 1981


0001                ASMB,L,Q,R
**  NO ERRORS PASS#1 **RTE ASMB 92067-16011**
1
 PAGE 0002 #01                          1:03 PM  MON., 13  JULY, 1981


0001                ASMB,L,Q,R
0002  00000               NAM PASST,7 PASSES ARRAY
0003                      ENT PASST
0004                      EXT LIMEM,.ENTR
0005*
0006*     THIS SUBROUTINE IS USED AS A GENERAL PURPOSE INTERFACE BETWEEN A
0007*     FORTRAN MAIN OR SUBROUTINE AND ANOTHER FORTRAN SUBROUTINE IN
0008*     ORDER TO PERMIT THE CALLED ROUTINE TO USE THE AREA BETWEEN THE
0009*     END OF PROGRAM (OR LARGEST SEGMENT) AND THE END OF MEMORY. THIS
0010*     ROUTINE ALLOWS THE PASSING OF UP TO 13 PARAMETERS TO THE SUBROUTINE
0011*     EXCLUDING THE ARRAY AND THE ARRAY LENGTH. THE CALLED ROUTINE MUST
0012*     HAVE AT LEAST TWO PARAMETERS, THE FIRST OF WHICH IS THE ARRAY AND
0013*     THE SECOND OF WHICH IS THE ARRAY LENGTH. SINCE THESE ARE NOT DIRECTLY
0014*     PASSED BY THE SOURCE ROUTINE, VALUES IN THE ARRAY CANNOT BE RETURNED
0015*     TO THE SOURCE UNLESS THEY ARE MOVED. THE CALLED ROUTINE MUST BE OF
0016*     THE FORM:
```

```
0017*
0018*      SUBROUTINE XXX (IARRY,L,P1,P2,P3...,P13)
0019*      INTEGER IARRY(1)
0020*
0021*      WHERE THE P'S ARE OPTIONAL PASSED PARAMETERS FROM THE SOURCE AND L
0022*      IS THE LENGTH OF IARRY. XXX IS ANY LEGAL FORTRAN SUBROUTINE NAME.
0023*      IF THE ARRAY IS DIMENSIONED AS REAL, 3 WORD OR 4 WORD DOUBLE
0024*      PRECISION, THEN THE SIZE WILL BE L/2, L/3, OR L/4 RESPECTIVELY.
0025*
0026*      THE SOURCE ROUTINE MUST BE OF THE FORM:
0027*
0028*      PROGRAM YYY  (OR SUBROUTINE YYY(...) )  YYY IS ANY LEGAL FORTRAN NAME
0029*      COMMON /XXX/XXX  WHERE XXX IS THE NAME OF THE SUBROUTINE TO BE CALLED
0030*                       (ANY LEGAL FORTRAN SUBROUTINE NAME)
0031*      CALL PASST (XXX,XXX,P1,P2,P3...P13)
1
 PAGE 0003 #01                          1:03 PM  MON., 13  JULY, 1981


0033  00000 000041R FSUB  JSB ADDR,I    CALL DEFINED FORTRAN ROUTINE
0034  00001 000021R       DEF JSUB      RETURN ADDRESS

0036  00002 000000  PARMS BSS 15        15 PARAMETERS ALLOWED
0037  00021 000022R JSUB  JMP PASST,I   EXIT TO CALLING ROUTINE



0039  00022 000000  PASST NOP           ENTRY TO THIS ROUTINE
0040  00023 000002X       JSB .ENTR     GET CALLING PARAMETERS
0041  00024 000002R       DEF PARMS     AND PUT HERE

0043  00025 000002R       LDA PARMS     GET ADDRESS OF FORTRAN SUBROUTINE
0044  00026 000041R       STA ADDR      AND STORE IN ANOTHER LOCATION

0046  00027 000001X       JSB LIMEM     GET STARTING ADDRESS AND SIZE
0047  00030 000034R       DEF *+4       OF ARRAY TO END OF MEMORY
0048  00031 000045R       DEF D0        (REQUIRED BY CALL)
0049  00032 000002R       DEF PARMS     PUT ARRAY ADDRESS HERE
0050  00033 000042R       DEF L         PUT LENGTH HERE

0052  00034 000044R       LDA JSUBS     .ENTR MAY OVERWRITE OUR CODE
0053  00035 000021R       STA JSUB      SO WE REWRITE IT JUST IN CASE

0055  00036 000043R       LDA AP1       CONTAINS THE ADDRESS OF LENGTH
0056  00037 000003R       STA PARMS+1

0058  00040 000000R       JMP FSUB      CALL FORTRAN SUBROUTINE
0059  00041 000000  ADDR  NOP           STORAGE FOR FORTRAN SUBROUTINE ADDRESS
0060  00042 000000  L     NOP           LENGHT PLACED HERE
0061  00043 000042R AP1   DEF L         ADDRESS OF ABOVE
0062  00044 000022R JSUBS JMP PASST,I
0063  00045 000000  D0    DEC 0
0064                      END
**  NO ERRORS *TOTAL **RTE ASMB 92067-16011**
```

FMP CONSIDERATION IN IMAGE SHARED DATABASE ACCESS

by Gary Ericson/HP Data Systems Division

It has been pointed out to me recently from a number of different sources that there is an idiosyncracy (if you want to call it that) in the File Management Package which affects IMAGE programs, but which is not readily apparent to the user. This affects programs like QUERY, DATACAP, and any user-written programs that access a database in shared read/write mode.

HOW DOES FMP WORK?

FMP was written with the concept of using a Data Control Block (DCB) buffer to speed up applications by reducing the number of disc accesses required to retrieve data from the disc. Basically, it works like this: when the first READF call is made, one or more 128 word blocks of data are read from the disc into the DCB buffer (the number of blocks depends on the size of the DCB). Then the actual record requested is transferred from the DCB into the user's buffer. For each READF call after that, the DCB is examined to see if the record desired is currently there. If so, the record is just transferred from the DCB to the user's buffer (just a memory to memory transfer). If the record is not currently in the DCB, a disc access is made to fill up the DCB with the blocks that contain the record, and then the record is transferred from the DCB to the user's buffer.

This works similarly for WRITF calls. If the record to be modified is already in the DCB, the WRITF call just transfers the user's buffer to the DCB (not to the disc). If the record is not in the DCB, the DCB is written out to the disc (if the DCB has modifications in it that haven't been written out to disc yet), the disc is read to fill up the DCB with the blocks containing the record, and the user's buffer is transferred to the DCB. This modified DCB is not written out to disc until the file is closed or until the DCB is written out to make room for other records.

You might want to read the RTE-IVB Programmer's Reference Manual for a more complete description of how all this works.

The net result is this: if the record to be read or written currently resides in the DCB buffer, the disc will not be accessed, but the record will just be read from or written to the DCB. If the record is not currently in the DCB (the DCB is empty or it has as different set of records in it), then the disc is read to get a copy of the blocks containing the record into the DCB. The DCB will then be accessed for the record.

The advantage to all this is that disc accesses are reduced. This is accomplished by only going to the disc when it's really necessary, and then transferring large amounts of data so that it will be necessary less often.

The disadvantage comes when a file is being shared in read/write mode by two or more programs. If one program does a WRITF call, his own DCB will be modified, but the disc will not be updated until one of the conditions mentioned above are met. This means that other programs reading the disc may not immediately see the changes the first program intended to make. Similiarly, if one program does actually modify the disc, another program doing a READF call will only read from the disc if one of the conditions mentioned above are met. This means that the second program may not see the changes that are physically on the disc.

The POST call was written to help this situation. If a DCB has been modified, the POST call will write the DCB to disc. If there have been no modifications to the DCB, the POST call just clears out the DCB. In either case, the result is that the DCB will now be empty, and the next record access will have to go to disc, both for READF and WRITF calls.

HOW DOES THIS AFFECT IMAGE?

Since all IMAGE calls eventually break down into standard FMP calls, the above discussion affects IMAGE directly. The impact with respect to modifying a database is reduced because of the fact that in 92069 IMAGE, all update calls (DBPUT, DBUPD, DBDEL) are concluded internally with a POST call which will force any updates to be written to the disc. This means that when you add, update, or delete a record, the change will show up on the disc immediately, and any other programs having the database open will have the chance to get the information you just put there.

The impact on reading the database is a little greater. When a database read is done (DBGET), a READF call is eventually generated. If the record being requested is already in the DCB, the data will not be retrieved from the disc. The problem, then, shows up in this kind of a sequence:

	QUERY #1	QUERY #2
1.	open database DB in mode 1	
2.		open database DB in mode 1
3.	FIND item IS "x" END; (this reads, say, record #7,putting record #7 into this local DCB)	
4.		FIND item IS "x" END; (this also reads record #7, putting that record into this local DCB)
5.	UPDATE R; (make some change to the record)	
6.		REPORT ALL; (list the contents of record #7)

In step 6, QUERY #2's REPORT will not reflect the changes made to record #7 by QUERY #1 because the unmodified record is still in QUERY #2's local DCB. The READF call that is generated to do the report will get the record out of the DCB, not off the disc.

It might be important to note that this occurs with any IMAGE programs, including DATACAP. Following the same kind of scenario as above, if DATACAP modifies record #7, QUERY may not see the changes made to that record. It's also important to note that while it appears from QUERY #2's perspective that no change has been made to record #7, the fact is that the real disc copy of that record has been modified. The disc will always be good because IMAGE posts its DCB's everytime it makes a change (whether from DBPUT, DBUPD, or DBDEL calls).

HOW DO WE GET OUT OF THIS?

When sharing a file like we've been discussing, it's apparent that at times it's desirable to force the READF call to go to the disc to pick up any changes that may have been made to the record. Basically what you need to do is change the contents of the DCB so that the record you are looking for is not in there. The READF routine will then have to go to the disc to get a copy of that record. The two possible ways to accomplish this are:

1. Change the contents of the DCB so that it contains some records other than the one you want.

2. Clear out the DCB altogether so that nothing is in there.

To change the contents of the DCB, you have to do a READF (or WRITF) call on a record that is not currently in the DCB. For example, if you want to get record #7 from disc, you could do a READF on record #99 (assuming that record #99 is not in your DCB). READF will look for record #99 in the DCB, and, not finding it, will go to the disc for it. This removes record #7 from the DCB. Now if READF is called for record #7, it won't find that record in the DCB, and it will have to go to disc for it. This whole process results in reading the real disc copy of record #7. The same thing applies to IMAGE: you make a DBGET call that retrieves a record that is not in the DCB, and then make the DBGET call for the record you want. This works fine, but it requires at least one extra disc access, and it's not always easy to know which records are not currently in your DCB. This is especially true in QUERY where you don't have any information about the physical records you're looking at.

The most efficient way to clear out the DCB is to do a POST call on the DCB as was mentioned above. In IMAGE, though, the DCB's for datasets are kept internally and the user doesn't have access to them. The only other way to clear out a DCB is to close the file and re-open it. This method is rather costly, but it's easy and it works. In IMAGE, a user can use DBCLS to close the whole database (very time-consuming) or he can use DBCLS (mode=2) to just close the individual dataset he is reading from. Unfortunately, in QUERY, the user has only the option of closing the whole database and re-opening it.

THE BOTTOM LINE

The bottom line of this whole discussion is this: when two or more IMAGE application programs (user-written, QUERY, DATACAP, etc.) have the same database open in shared read/write mode (mode=1), modifications made to the database by one program may not be seen immediately by the other programs. The disc will be up-to-date, but the local DCB in a program may not have the updated copy. The solution under IMAGE is to either do a DBGET on a record that's far enough away from the record you want that you're sure it's not currently in the DCB, or close the specific dataset and re-open it, or close the whole database and re-open it. In QUERY, this translates into doing a FIND on something you know is far enough away, or closing and re-opening the database with the DATA-BASE= command.

JOIN AN HP 1000 USER GROUP!

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

NORTH AMERICAN HP 1000 USER GROUPS

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

NORTH AMERICAN HP 1000 USER GROUPS (CONTINUED)

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

OVERSEAS HP 1000 USER GROUPS

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

OVERSEAS HP 1000 USERS GROUPS (CONTINUED)

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

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 the U.S., contact your local sales and service office for prices in your country.

Printed in U.S.A. 7/81

Part No. 5951-6111