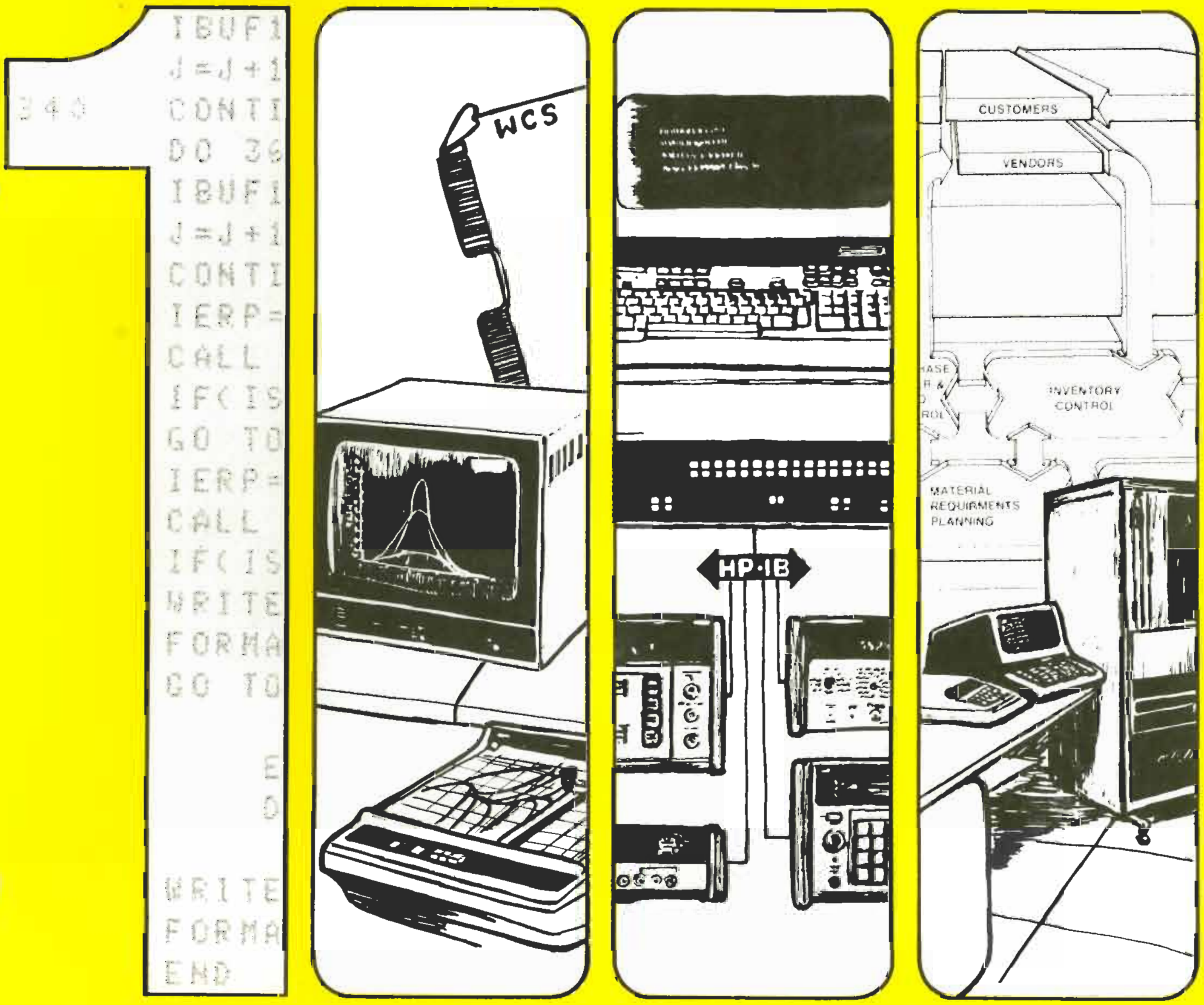
HEWLETT PACKARD

Hewlett-Packard Computer Systems COMMUNICATOR



1980

Volume IV Issue 5

HEWLETT-PACKARD
COMPUTER SYSTEMS

Volume IV
Issue 5

COMMUNICATOR/1000

Feature Articles

ABOUT THIS ISSUE

Looking at the articles in this issue I must say I am continually amazed at the variety of subjects HP/1000 users find to write about. The feature articles in this issue span the breadth from performance to data base internals. In addition, all four Bit Bucket articles are extremely useful.

In the OPERATING SYSTEMS category I have published the sixth article in the series on HP Subroutine Linkage Conventions written by Bob Niland of HP's Lexington, Massachusetts office. When the entire series has been published subscribers will have a valuable reference manual at their disposal.

In the COMPUTATION section Charles Fugee of Holloman Air Force Base in New Mexico submitted his results on benchmarking the computers in the HP/1000 product line. This article was published in the COMPUTATION section because Mr. Fugee's benchmarks are oriented towards number crunching performance. Note that if you wish to get a copy of his benchmarking program he has moved to HP's Atlanta, Georgia sales and service office.

In the OPERATIONS MANAGEMENT section there are two fine articles written by DSD employees. Jim Bridges has written an article on an up and coming topic, "Increasing System Availability through Redundant Components", and Jim Burkett describes his use of IMAGE/1000 to create a multi-level network database.

This time there was no competition for a calculator in the customer category. However, there were two competing entries from HP's Data Systems Division, the winner of which was decided by our panel of judges. There was no winner from HP field offices because Bob Niland is ineligible. The winners are:

Best Feature Article by a Customer	BENCHMARKING HP/1000 COMPUTERS Charles G. Fugee
Best Feature Article by an HP Division Employee not in Data Systems Technical Marketing	INCREASING SYSTEM AVAILABILITY THROUGH REDUNDANT COMPONENTS Jim Bridges

I hope you enjoy reading the articles in this issue, and continue to provide the Communicator with interesting and useful material.

The Editor

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

LETTER TO THE EDITOR

Dear Editor,

Here is an update to an article I wrote which appeared in Volume II, Issue 6 of the Communicator/1000 (pages 36-41).

Shared EMA capability was first developed and implemented on an RTE-IVA system (Rev. 1840) in October of 1980. It will also work on an RTE-IVB system. I coined the name "SHEMA/1000" for a program I contributed to the PLUS/1000 library. SHEMA/1000 gives the FORTRAN, PASCAL, or Assembly language programmer the ability to share a large area of physical memory by using the standard EMA programming conventions. The maximum amount of memory that can actually be shared is limited only by the number and size of user partitions and available memory. For more details on SHEMA/1000, refer to the Communicator article.

Jeff Mason, Systems Engineer in our San Antonio office, has discovered something that is missing from the subroutine IDMAP which will cause a program to abort with dynamic mapping violations when it attempts to access the shared EMA area. The problem lies in that the wrong version of IDMAP was submitted by the author to the contributed library (LOCUS), now PLUS/1000. The version of IDMAP that was submitted to the library was a specialized version for the particular application in which SHEMA/1000 was originally implemented. A general purpose version of IDMAP that was developed will be submitted to PLUS/1000 in the very near future. In the meantime, for those of you that are using SHEMA/1000, add the following code beginning after line 62 of IDMAP:

```
EXT $IDEX
STB 0         PUT ID SEG ADDR IN A REG
LDA 0,I
LSR 10        ISOLATE ID EXT NUMBER.
ADA =D-1      ADJUST RELATIVE TO 0.
MPY =D3       INDEX INTO ID EXT TABLE.
ADA $IDEX+0
STA 1745B     PUT ON BASE PAGE.
```

This will always ensure that the proper address of the executing programs ID extension entry is placed on base page prior to firmware access. This can also be done by forcing the program to be rescheduled and thus letting the system set-up base page.

I must apologize for any inconvenience that this may have caused users of SHEMA/1000. This is the only problem with it that has been brought to my attention.

Sincerely,

Larry W. Smith
Systems Engineer
HP Fullerton, Ca.

Dear Larry,

Thanks for the update, and for your continued support of the Communicator/1000.

With best regards,

The Editor

RESTORING GRANDFATHER RELOCATABLES FROM MAG-TAPE

By Eddie Yep/HP Data Systems Division

Have you ever wanted to generate a new system without shutting down your current system? Here is a program called GFREL that will enable a user to restore cartridge 32767 from mag-tape to a disc LU. (The disc LU cannot be LU 2.)

GFREL will copy the entire Grandfather from mag-tape to the specified disc LU. For RTE-II/III/IVA systems, the cartridge directory will be removed from the last track of the cartridge, and the disc directory moved to the sector 0 of the last track. GFREL then updates the disc directory to set the starting FMP area to track 0 and sector 0.

The user should then pack this cartridge to over-lay the operating system portion of the Grandfather and update the cartridge directory.

Below is a listing of GFREL within which its run string and limitations are described. GFREL will also be available in the PLUS/1000 contributed library produced by the HP/1000 Users Group.

```
FTN4,L,C
      PROGRAM GFREL(3,50), EXTRACT REL FROM GF 08-25-80
C
C THIS PROGRAM COPIES AN IMAGE OF A GRANDFATHER DISC FROM MAG TAPE TO
C DISC.  IT ASSUMES THAT:
C
C    1.  RTE-IVB GRANDFATHERS (EXCLUDING 7900) ARE IN LSAVE FORMAT.
C
C    2.  RTE-IVB GRANDFATHERS IN 7925(H) FORMAT CAN ONLY BE RESTORED
C        TO A 7925(H) DISC.
C
C    3.  RTE-IVB GRANDFATHER ON 7900 DISC IS IN 7900 DISC BACKUP
C        FORMAT.
C
C    4.  RTE-II/III/IVA GRANDFATHERS ARE IN DISC BACKUP FORMAT.
C        (SAME FORMAT AS RTE-IVB 7900)
C
C    5.  RTE-II/III GRANDFATHERS HAVE 203 TRACKS (7905/06/20 DISC).
C        (NOTE:  OFF-LINE SAVE MAY HAVE 256 TRACKS, BUT ONLY 203
C         ARE VALID. DIRECTORY IS ON THE 203RD TRACK)
C
C    6.  THE NUMBER OF TRACKS SPECIFIED FOR THE DISC LU IS LARGE
C        ENOUGH FOR THE GRANDFATHER (203 TRACKS FOR 7900 DISCS AND
C        256 TRACKS FOR ALL OTHERS), ...
C
C    *** OTHERWISE THE CARTRIDGE DIRECTORY WILL GET CORRUPTED!!! ***
C
C  THE CALLING SEQUENCE IS AS FOLLOWS:
C
C     RU,GFREL,MTLU,DSKLU,NTRKS,OPSYS,DTYPE
C
C     MTLU  =  MAG TAPE LU (SOURCE)
C     DSKLU =  DISC LU (DESTINATION)
C     NTRKS =  NUMBER OF TRACKS ON THE DESTINATION DISC LU
C     OPSYS =  OPERATING SYSTEM
C                1 - RTE-IVB
C                2 - RTE-IVA
C                3 - RTE-II/III
```

```
C     DTYPE =  TYPE OF DISC
C                1 - 7900
C                2 - 7905/06/20 (MAC/ICD)
C                3 - 7925       (MAC/ICD)
C
C     DISC BACKUP FORMAT
C     ------------------
C
C     IF MAG TAPE RECORDS ARE 2048 + 2, THEN TWO RECORDS (MINUS THE
C     TWO TAG WORDS AT THE FRONT) ARE WRITTEN INTO EACH TRACK. THE
C     SECOND RECORD BEGINS AT SECTOR 32.
C
C NOTE: SEE RTE-IVB UTILITY PROGRAMS REFERENCE MANUAL FOR TAPE
C       FORMATS.  RTE-II/III/IVA AND RTE-IVB 7900 DISC USE THE
C       7900 DISC BACKUP FORMAT.  RTE-IVB (EXCLUDING THE 7900
C       DISC) USE THE LSAVE FORMAT.
C
C
      DIMENSION IBUF(8193),IREG(2),IP(5),IHEDR(247),IDCB(144)
      DIMENSION INBUF(40),IPBUF(10)
C
      INTEGER DSKLU,RECSZ,HDSIZ,HDPTR,HDLEN,BFPTR,RECFG
      INTEGER OPSYS,DTYPE
C
      EQUIVALENCE (IREG,REG),(IREG,IA),(IREG(2),IB)
      EQUIVALENCE (IHEDR(239),NSEC),(IHEDR(42),RECFG)
C
      DATA INLEN/-80/,IPTR/1/,BFPTR/3/,LREC/0/,LTRK/0/,ISEC/0/
      DATA IPTR/1/,ICNT/0/,IFLG/0/,RECSZ/6144/,HDSIZ/140/
      DATA HDPTR/1/,HDLEN/36/,NOPAR/0/,NFLG/0/,JSEC/96/
      DATA IZERO/0/,I1/1/,I2/2/,I3/3/,I5/5/,I8/8/,I14/14/,I16/16/
      DATA I10/10/,I32/32/,I60/60/,I64/64/,I128/128/,I247/247/
      DATA I203/203/,I256/256/,I2048/2048/,LSAVE/0/
C
C     GET TERMINAL LU THEN GET RUN STRING AND PARSE IT
C
      ILU=LOGLU(DUMMY)
      CALL GETST(INBUF,INLEN,ILOG)
      DO 600 I=1,5
      IF(NAMR(IPBUF,INBUF,ILOG,IPTR))30,10
C
C     PARAMETERS SPECIFIED, DETERMINE IF CORRECT TYPE (INTEGER).
C     IF NOT, PROMPT FOR PARAMETERS.
C
10    CONTINUE
      ITYPE=IAND(IPBUF(4),I3)
      IF(ITYPE .EQ. I1) GO TO 20
      GO TO (100,200,300,400,500) I
C
C     TYPE IS CORRECT, GO CHECK IF VALUES ARE WITHIN LIMITS SPECIFIED
C
20    CONTINUE
      GO TO (120,220,320,420,520) I
```

```
C
C     NO PARAMETERS SPECIFIED, GO PROMPT FOR PARAMETERS
C
30    CONTINUE
      IF(NFLG .EQ. I1) GO TO 40
      WRITE(ILU,35)
35    FORMAT(/5X,"TO TERMINATE, ENTER 0")
      NFLG=1
40    CONTINUE
      GO TO (110,210,310,410,510) I
C
C     PROCESS MAG TAPE LU PARAMETER
C     ENSURE MAG-TAPE DRIVER IS DVR23/DVR24
C
100   CONTINUE
      WRITE(ILU,105)
105   FORMAT(/5X,"INVALID MAG TAPE LU!!!")
110   CONTINUE
      WRITE(ILU,115)
115   FORMAT(/5X,"PLEASE ENTER MAG TAPE LU..._")
      READ(ILU,*) IPBUF
      IF(IPBUF .EQ. IZERO) GO TO 9999
120   CALL EXEC(13,IPBUF,ISTAT)
      IF(IAND(ISTAT,37400B)*2 .EQ. 23000B) GO TO 125
      IF(IAND(ISTAT,37400B)*2 .EQ. 24000B) GO TO 125
      GO TO 100
125   CONTINUE
      MTLU=IPBUF
      IF(NOPAR .EQ. I1) GO TO 210
      GO TO 600
C
C     PROCESS DISC LU PARAMETER (DISC LU CAN BE 1-255)
C
200   CONTINUE
      WRITE(ILU,205)
205   FORMAT(/5X,"INVALID DISC LU!!!")
210   CONTINUE
      WRITE(ILU,215)
215   FORMAT(/5X,"PLEASE ENTER DISC LU..._")
      READ(ILU,*) IPBUF
      IF(IPBUF .EQ. IZERO) GO TO 9999
      IF(IPBUF .EQ. 2) GO TO 200
220   IF((IPBUF .LT. IZERO) .OR. (IPBUF .GT. 255))GO TO 200
      DSKLU=IPBUF
      IF(NOPAR .EQ. I1) GO TO 310
      GO TO 600
C
C     PROCESS NUMBER OF DISC TRACKS (VALUE CAN BE 1-8192)
C
300   CONTINUE
      WRITE(ILU,305)
305   FORMAT(/5X,"INVALID NUMBER OF TRACKS SPECIFIED!!!")
310   CONTINUE
      WRITE(ILU,315)
315   FORMAT(/5X,"PLEASE ENTER NUMBER OF TRACKS ON DISC LU..._")
      READ(ILU,*) IPBUF
      IF(IPBUF .EQ. IZERO) GO TO 9999
320   IF((IPBUF .LT. IZERO) .OR. (IPBUF .GT. 8192)) GO TO 300
      NTRKS=IPBUF
      IF(NOPAR .EQ. I1) GO TO 410
      GO TO 600
C
C     PROCESS TYPE OF OP-SYS
C
400   CONTINUE
      WRITE(ILU,405)
405   FORMAT(/5X,"INVALID OP-SYS SPECIFIED!!!")
410   CONTINUE
      WRITE(ILU,415)
415   FORMAT(/5X,"PLEASE ENTER TYPE OF OP-SYS:",
     2   /15X,"1 - RTE-IVB",
     3   /15X,"2 - RTE-IVA",
     4   /15X,"3 - RTE-II/III",16X,"..._")
      READ(ILU,*) IPBUF
      IF(IPBUF .EQ. IZERO) GO TO 9999
420   CONTINUE
      IF((IPBUF .LT. I1 ) .OR. (IPBUF .GT. I3)) GO TO 400
      OPSYS=IPBUF
      IF(NOPAR .EQ. I1) GO TO 510
      GO TO 600
C
C     PROCESS DISC TYPE
C
500   CONTINUE
      WRITE(ILU,505)
505   FORMAT(/5X,"INVALID DISC TYPE SPECIFIED!!!")
510   CONTINUE
      WRITE(ILU,515)
515   FORMAT(/5X,"PLEASE ENTER DISC TYPE:",
     2   /15X,"1 - 7900",
     3   /15X,"2 - 7905/06/20   (MAC/ICD)",
     4   /15X,"3 - 7925         (MAC/ICD)",4X,"..._")
      READ(ILU,*) IPBUF
      IF(IPBUF .EQ. IZERO) GO TO 9999
520   CONTINUE
      IF((IPBUF .LT. I1) .OR. (IPBUF .GT. I3)) GO TO 500
      DTYPE=IPBUF
      IF(NOPAR .EQ. I1) GO TO 650
600   CONTINUE
C
C     DETERMINE IF THE # OF DISC TRACKS SPECIFIED IS ACCEPTABLE.
C     IF NOT ENOUGH TRACKS ARE SPECIFIED THE CARTRIDGE DIRECTORY
C     WILL GET CLOBBERED.
C
C         7900 DISC:   AT LEAST 203 TRACKS
C         ALL OTHERS:  AT LEAST 256 TRACKS
C
650   CONTINUE
      IF(((DTYPE .EQ. I1) .AND. (NTRKS .GE. I203)) .OR.
     2 ((DTYPE .NE. I1) .AND. (NTRKS .GE. I256))) GO TO 675
      WRITE(ILU,660)
660   FORMAT(/5X,"*** NUMBER OF TRACKS SPECIFIED IS NOT LARGE ",
     2 "ENOUGH TO RESTORE GF !!! ***")
      GO TO 9999
```

```
C
C        SET UP PARAMETERS TO READ THE ENTIRE HEADER IN
C        LSAVE FORMAT.
C
675     CONTINUE
        IF((OPSYS .EQ. I1) .AND. (DTYPE .NE. I1)) LSAVE=I1
        IF(LSAVE .NE. I1) GO TO 700
        HDSIZ=I247
        HDPTR=I16
        HDLEN=I60
C
C       READ HEADER AND OUTPUT HEADER ID ONLY.
C
700     CONTINUE
        WRITE(ILU,710)
710     FORMAT(/5X,"BEGIN TRANSFER!    MAG-TAPE HEADER IS:"/)
        CALL EXEC(I1,MTLU,IHEDR,HDSIZ)
        CALL EXEC(I2,ILU,IHEDR(HDPTR),HDLEN)
C
C       SET UP PARAMETERS TO READ RECORDS IN LSAVE FORMAT.
C       ENSURE THAT THE NUMBER OF SECTORS/TRACK IS CORRECT.
C
        IF(LSAVE .NE. I1) GO TO 800
        BFPTR=I2
        RECSZ=NSEC * I64
        IF(NSEC .NE. JSEC) JSEC=NSEC
        MTREC=RECSZ + I1
        GO TO 900
C
C       SET UP PARAMETERS TO READ RECORDS OF 2048 WORDS
C
800     CONTINUE
        IF(RECFG .EQ. I1) GO TO 810
        IFLG=I1
        RECSZ=I2048
810     CONTINUE
        MTREC=RECSZ+I2
C
C       READ MT RECORDS AND WRITE CONTENTS TO THE DISC.
C       TAG WORDS ON MAG TAPE ARE NOT WRITTEN TO THE DISC.
C
900     CONTINUE
        IF(IFBRK(IDUM)) 910,920
910     CONTINUE
        STOP 55
920     CONTINUE
        REG=EXEC(I1,MTLU,IBUF,MTREC)
        IF(IAND(IA,200B) .NE. IZERO) GO TO 5000
        LREC=LREC+1
        CALL EXEC(I2,DSKLU,IBUF(BFPTR),RECSZ,LTRK,ISEC)
        IF(IFLG .NE. I1) GO TO 1000
```

```
C
C     IF THERE ARE 2048 WORDS/RECORD, TWO RECORDS ARE WRITTEN
C     PER TRACK.  THE SECOND RECORD STARTS ON SECTOR 32.
C
      IFLG=IZERO
      ISEC=I32
      GO TO 900
C
C     SET UPDATE TRACK AND SECTOR.  IF DESTINATION LU IS NOT
C     LARGE ENOUGH, ISSUE MESSAGE AND TERMINATE.
C
1000  CONTINUE
      ISEC=IZERO
      LTRK=LTRK+I1
      IF(LTRK .LE. NTRKS) GO TO 900
C
C     IF YOU GET TO HERE, YOU HAVE CLOBBERED YOUR DIRECTORY!!!
C
      WRITE(ILU,1005)
1005  FORMAT(/5X,"DESTINATION DISC LU IS TOO SMALL!!!",
     2   //5X,"*** DIRECTORY HAS BEEN CLOBBERED!!! ***")
      STOP 44
C
C     OUTPUT THE NUMBER OF RECORDS WRITTEN.
C
5000  CONTINUE
      WRITE(ILU,5010) LREC
5010  FORMAT(/5X,"NUMBER OF RECORDS WRITTEN:  ",I4)
C
C     ENSURE THAT RTE-II/III G.F. ON 7905/06/20 HAVE 203 TRKS.
C     INSURE THAT THE DIRECTORY IS ON THE LAST TRACK OF CRN.
C
C           ***  THE DIRECTORY IS LAST TRACK ***
C                  RTE-IVB    - SECTOR 0.
C                  ALL OTHERS - SECTOR 14.
C
C     UPDATE THE DIRECTORY LIST TO HAVE THE FMP AREA POINT
C     TO TRACK 0 AND SECTOR 0.
C
      IF((OPSYS .EQ. 3) .AND. (DTYPE .EQ. 2)) LTRK=203
      LTRK=LTRK-I1
      IF(OPSYS .NE. I1) ISEC=I14
      CALL EXEC(I1,DSKLU,IBUF,I128,LTRK,ISEC)
5020  CONTINUE
      IBUF(I5)=IZERO
      IF((LTRK+1) .LT. NTRKS) IBUF(I8)=NTRKS+IBUF(9)
C
C     SET THE SECTOR TO 0.  OVERLAY THE CARTRIDGE LIST
C     ON RTE-II/III/IVA GF'S (CARTRIDGE LIST EXIST ON THE
C     FIRST 2 SECTORS ON THE LAST TRACK) WITH THE
C     UPDATED DIRECTORY LIST.
C
      ISEC=IZERO
      CALL EXEC(I2,DSKLU,IBUF,I128,LTRK,ISEC)
      IF(OPSYS .EQ. I1) GO TO 5500
```

```
C
C     FOR RTE-II/III/IVA GF'S, PURGE THE DIRECTORY
C     ENTRIES ON SECTOR 14 BY SETTING WORD 1 OF EACH
C     16 WORD ENTRY TO -1.
C
      ISEC=14
      DO 5100 I=I1,I128,I16
      IBUF(I)=-1
5100  CONTINUE
      CALL EXEC(I2,DSKLU,IBUF,I128,LTRK,ISEC)
5500  CONTINUE
C
C     ENSURE THAT THE DIRECTORY IS ON THE LAST TRACK OF THE
C     DESTINATION DISC.  PURGE OLD DIRECTORY.
C
      IF((LTRK+I1) .GE. NTRKS) GO TO 6000
      NTRKS=NTRKS-I1
      DO 5600 J=1,JSEC-1,2
      CALL EXEC(I1,DSKLU,IBUF,I128,LTRK,J-I1)
      CALL EXEC(I2,DSKLU,IBUF,I128,NTRKS,J-I1)
      DO 5550 KK=I1,I128,I16
      IBUF(KK)=-1
5550  CONTINUE
      CALL EXEC(I2,DSKLU,IBUF,I128,LTRK,J-I1)
5600  CONTINUE
C
C     ISSUE SUCESSFUL TERMINATION MESSAGE
C
6000  CONTINUE
C
C     PACK DISC CRN TO REMOVE OP-SYSTEM AND UPDATE
C     CARTRIDGE LIST.
C
      WRITE(ILU,6005)
6005  FORMAT(/5X,"1.  PACK DSKLU TO GET RID OF OP-SYSTEM",
     2       /5X,"          PK,-LU (CRN)",
     3      //5X,"2.  UPDATE CARTRIDGE LIST",
     4       /5X,"          DC,-LU,RR",
     5       /5X,"          SL,LU,LU (RTE-IVB ONLY)",
     6       /5X,"          MC,LU",
     7      //5X,"    ***   TASK COMPLETED  ***   ")
      STOP 77
9999  CONTINUE
      WRITE(ILU,10000)
10000 FORMAT(/5X,"ZERO/CR/INVALID PARAM ENTERED:  ",
     2    "PROGRAM TERMINATED!!!!")
      STOP 66
      END
```

IN AND OUT OF SESSION

By Jack Sadubin/Environment Canada

The RTE4B system programmer invariably runs into session related problems when implementing a general purpose program which will run for users of different capability levels. This is especially true if the program being written is a powerful one which is intended to accomplish tasks which are not permitted to a user on his own (through FMP or break mode) due to session capability restrictions.

System commands issued via the 'MESSS' utility by such a program will only work for high capability users. Access to logical units such as the DS-1000 nodes may not work if these LU's are not in a user's SST. Other programs (sons) scheduled without wait will abort when the father completes first and the user logs off. These are only some of the problems we may run into under session.

Detaching one's program from the session environment is desirable under some of the above circumstances. The undocumented HP routine, DTACH, may be used to detach the program from session control but, once called, the program is permanently detached from session. If the program needs access to private or group disc LU's, or if the programmer wishes to return the program to session control for protection or time accounting, the program must be re-attached to the session.

Word 33 of the program id-segment is the key by which RTE4B controls session and non-session programs. This word contains either the session word for programs running under session control, or the negative of the LU from which the program was scheduled for programs running outside session. Modification of this word programmatically is the way by which a programmer can get around the problems, and detach or attach his program at will from session.

The small subroutine, DEATS, listed below permits the programmer to accomplish this useful feat anywhere in the code. The calling program can detach and attach itself to session at any time, and accomplish great flexibility. We can now reduce the need to write multiple versions of the same software for different capability users, and simplify error checking in the software.

If some of the software in your system runs into these problems, the usage of such a routine makes life a lot easier.

```
FTN4
      SUBROUTINE DEATS(OPTION),DETACH & ATTACH TO SESSION
C
C     ****************************************************************
C     * TO DETACH FROM SESSION CALL WITH OPTION= NON ZERO            *
C     * TO ATTACH TO   SESSION CALL WITH OPTION= ZERO                *
C     ****************************************************************
C
      INTEGER   MYNAME(3),ATTACH,DETACH,OPTION
      LOGICAL   IRAN
      DATA      IRAN /.FALSE./
C
COMMENT: FIRST TIME CALLED ?
C
      IF(IRAN) GO TO 100
      IRAN = .TRUE.
C
COMMENT: SET-UP
C
COMMENT: WHAT IS MY NAME ?
C
      CALL PNAME(MYNAME)
```

```
C
COMMENT: GET MY ID-SEGMENT ADDRESS WORD 33 (SESSION WORD) & SAVE IT.
C
      ID    = IDGET(MYNAME)+32
C
COMMENT: GET SESSION WORD & SAVE IT
C
      ATTACH= IGET(ID)
C
COMMENT: WHAT LU WERE WE SCHED FROM ?
C
      DETACH =LOGLU(ISYS)
      DETACH = -IABS(ISYS)
C
COMMENT: LETS DO IT
C
100   IF(OPTION .EQ.0) CALL IXPUT(ID,ATTACH)
      IF(OPTION .NE.0) CALL IXPUT(ID,DETACH)
      RETURN
      END
```

CLEARING SOFTKEY LABELS FROM THE SCREEN

by John Pezzano/HP El Paso

The HP 2645, 2647 and 2648 Terminals have programmable softkeys that can be set up using utilities "KEYS" and "KYDMP" (RTE-IVB Terminal Users Manual). Using these utilities, not only are the softkeys programmed, but labels are placed at the top of the screen and they are memory locked to prevent accidental erasure.

One problem with this technique is that the labels remain even after the user has logged off. The user may not want other users to know what softkeys are programmed in the terminal or what information is left on the screen at logoff. In addition, the constant, high brightness labels can eventually burn the screen (leaving a permanent imprint) after a long period of time if the terminal is left on for hours or days and the screen is never cleared after logging off.

Ideally, each user should unlock screen memory and clear the screen at logoff time, but this is not always done. There is an alternative. If one of the keys has been programmed to log the user off, ("`:EX`", "`:EX,SP`", or "`:EX,RP`") a little addition can clear the screen at the same time.

The secret to clearing the labels is to understand that escape sequences can be used to clear the screen and unlock memory, and that these sequences can be embedded in the "EX" command. When the "EX" command is interpreted by FMGR, the first parameter is checked. It must be either "SP" (save private cartridges) or "RP" (release them). Any other value results in a FMGR-056 error (bad parameter). However, parameters 2 and 3, which are used to optionally release group (RG) cartridges and kill active programs (KI), can have any value which will be ignored if it is not "RG" or "KI". Therefore, any escape sequence can be programmed in these parameters or after them. For example:

```
:EX,SP,XX
:EX,RP,RG,XX
:EX,RP,RG,KI,XX
```

where XX is anything, all are acceptable. In our case we want to clear lock, home up and clear screen. If we programmed (with KEYS) a softkey labeled "EXIT" which had in it one of the above where

```
XX=ESCM ESCH ESCJ,
```

as we logged off by hitting the "EXIT" key, our screen including labels would be cleared. The only remaining printout would be the system logoff messages which would follow the "EX" command.

As it turns out, almost any system command or FMGR command can have following the last parameter (provided it is separated by commas from the valid command). The extra information will be ignored, but will be seen by the terminal, permitting a single command to execute some action in the CPU while locally controlling the terminal.

USING DYNAMIC MEMORY SPACE WITH FORTRAN IV

By Robert P. Byard/HP Data Systems Division

Dynamic memory space is the name of the area between the last word of a program and the last word of the partition in which it executes. This space is usually wasted but available for programs to use. This article describes how dynamic memory space may be used by FORTRAN programs, and gives some example applications.

WHERE AND HOW MUCH?

The EXEC 26 call returns the address of the first available word after a program, the number of words between that address and the end of the partition, and the number of pages (including base page) in the partition.

```
      CALL EXEC( 26, FAVWD, NWRDS, NPAGS )
```

Exec call for dynamic memory space parameters.

Unfortunately this information is not useful at the level of FORTRAN because of the isolation from machine addresses forced by the nature of high level languages like FORTRAN. There is no clean way by FORTRAN alone to use a buffer space, given its logical address (FAVWD). Therefore, an assembly language interface is needed to make the dynamic memory space look like an ordinary FORTRAN array.

Imagine we have a FORTRAN program, HOHUM, with an integer array BUFF of length BUFFL. The first few lines of code would look as follows:

```
      PROGRAM HOHUM
      IMPLICIT INTEGER (A-Z)
      INTEGER BUFF(1234), BUFFL
      DATA BUFFL/1234/
         . . .
      END
```

In order to make BUFF into a dynamic memory array we must create an assembly program (HOHUM), and rewrite the FORTRAN routine as a subroutine (HOHUS). The routines are shown below.

```
      NAM HOHUM,3
      ENT HOHUM
      EXT EXEC, HOHUS
*
HOHUM NOP
*
      JSB EXEC       GET MEMORY SIZE
      DEF *+5
      DEF D26
      DEF BUFF
      DEF BUFFL
      DEF NPAGS


 *
      JSB HOHUS      PASS BUFF, BUFFL TO HOHUS
      DEF *+3
BUFF  BSS 1          PASSING THE DYNAMIC SPACE AS AN ARRAY
      DEF BUFFL
*
      JSB EXEC       IF HOHUS RETURNS, DO EXIT
      DEF *+1
      DEF D6
*
BUFFL BSS 1
NPAGS BSS 1
D26   DEC 26
D6    DEC 6
*
      END HOHUM

   SUBROUTINE HOHUS( BUFF, BUFFL )
   IMPLICIT INTEGER (A-Z)
   INTEGER BUFF(BUFFL)
   ...
   END
```

Note the unusual means used for passing BUFF to HOHUS. FORTRAN parameters are passed by means of a "call by reference" convention. The parameters are made available to a subroutine through indirect addressing. In this particular case we passed the address of the dynamic memory space directly in the call and achieved the effect of passing the dynamic memory space as an array.

DIVIDING DYNAMIC MEMORY SPACE INTO MULTIPLE ARRAYS

It may be that you want more than one array associated with the dynamic memory space. An extension of the preceding technique can accomplish this. Replace the end of the assembly module code (HOHUM) with the following.

```
       LDA NWRDS
       CLB
       DIV D3          A = NWRDS/3
       STA NWRDS
       ADA BUFF        DETERMINE BUFF2'S STARTING ADDRESS
       ADA NWRDS
       STA BUFF2
       ADA NWRDS       DETERMINE BUFF3'S STARTING ADDRESS
       STA BUFF3
*
       JSB HOHUS       PASS THE 3 ARRAYS BY THEIR ADDRESSES
       DEF *+5
BUFF   BSS 1
BUFF2  BSS 1
BUFF3  BSS 1
       DEF BUFFL
*
       ...

   SUBROUTINE HOHUS( BUFF, BUFF2, BUFF3, BUFFL )
```

The end result is three singly dimensioned buffers of equal length.

MULTI-DIMENSIONAL BUFFERS

The trick with multi-dimensional buffers is to calculate the number of words required per array entry and divide the length of the dynamic space by this number. For example, an integer array of (3,5,7) would require 15 words per entry or 105 words in total. Here is the code which calculates BUFFL before the call to HOHUS.

```
       LDA NWRDS
       CLB
       DIV ENTLN       DIVIDE BY ENTRY LENGTH (15)
       STA BUFFL       NUMBER OF ENTRIES AVAILABLE
       ...
ENTLN  EQU D15
D15    DEC 15

       SUBROUTINE HOHUS( BUFF, BUFFL )
       IMPLICIT INTEGER (A-Z)
       INTEGER BUFF(3,5,BUFFL), BUFFL
       ...
```

DYNAMIC BUFFERS, NOT STATIC BUFFERS!

Using these techniques will decrease the overall size of your programs. This results from moving otherwise static buffers into dynamic memory space. When your program is scheduled to run it will be assigned to the smallest possible partition. It may be that the resulting dynamic memory space will be too small for your program. In that case use the SZ command to increase the minimum number of pages in which your program may be executed. This will increase the dynamic memory space and consequently enlarge any buffers associated with that area.

```
:SYSZ,program                    tell me how big it is now...
 35147     15

:SYSZ,program,27                 increase from 15 to 27 pages...
```

The user should be on guard against an error common to these algorithms, namely zero or near zero dynamic memory space. In these cases BUFFL, or the number of entries in the "dynamic buffers", will be zero. Check BUFFL upon entry to the main routine to be safe. If it is too low, then terminate after informing the user to size up the program.

HP SUBROUTINE LINKAGE CONVENTIONS

by Robert Niland/HP Lexington, Mass.

[**Editor's Note:** This is the sixth part in a series of articles taken from Bob Niland's manual on HP Subroutine Linkage Conventions. The series started in Volume III, Issue 6.]

6-5. PRIVILEGED & RE-ENTRANT MODE WITH .ENTC

The preceding sections (6-3, 6-4) made little reference to parameter passing. In order for a privileged or re-entrant subroutine to be a useful external module, we clearly need to address this issue. We will begin the discussion with register-passing.

Both $LIBR and $LIBX calls preserve registers. This is not documented in any RTE manuals, but is apparent from inspection of the Library manual. Routines such as .LBT are privileged (type 6), and emulate machine instructions which use the contents of A and B as operands. If we desired to write emulator routines for all the HP1000 bit, byte and word instructions, and these routines might be called by memory-residents in an RTE-II/2100S environment, we could take advantage of this.

For example, the Set BitS (SBS) instruction would (in part) have the following calling sequence:

```
    ...
    JSB .SBS          Call the emulator
    DEF MASK          Address of mask.
    DEF A,I           Address of target word.
    ...               return point.
```

The entry/exit sequence in .SBS might look like this.

```
      NAM .SBS,6  SBS for 2100S and earlier cpu.
      ENT .SBS
      EXT $LIBR,$LIBX
.SBS  MPX             Entry point/return address.
      JSB $LIBR       Turn off interrupts,
      NOP             and go privileged.
      STA [A]         Save A and B since the SBS does not use
      STB [B]         them unless they are operands.
      LDA .SBS,I      Get "DEF <mask>"
      CPA @A          Is address the A register?
      JMP IS.A        Handle it.
      CPA @A.I        Is it A,I ?
      JMP IS.AI       Handle it.
      LDA A,I         Otherwise just get mask.
      ...             More code not shown.
      ...
      LDA [A]         Restore A
      LDB [B]         Restore B
      JSB $LIBX       Restore interrupts,
      DEF .SBS        and return external.
      ...
```

Emulating the SBS is actually a very tricky process, because, as we can see in the sample calling sequence, the operand addresses following the JSB .SBS may point to or through the A and B registers. However, in most routines the calling sequence does not allow the operand addresses to be A or B. In this event we can use a Library utility routine, .ENTC, to track down the direct addresses of the parameters.

For an imaginary routine (.NFER) which transfers 10 words under the following calling sequence...

```
      ...
      JSB .NFER        call for Namr transFER,
      DEF NAMR1        From array NAMR1.
      DEF @NMR2,I      To NAMR2 (through indirect link).
      ...              Return point.
```

the entry sequence in .NFER might look like this:

```
      NAM .NFER,6  fmp Namr transFER.
      ENT .NFER
      EXT $LIBR,$LIBX
      ...
@FROM MPX              Link to source.
@TO   MPX              Link to destination.
.NFER MPX              Entry point/return address.
      JSB $LIBR        Turn off interrupts,
      NOP              and go privileged.
      JSB .ENTC        Resolve link, adjust return point.
      DEF @FROM        Starting with @FROM.
      ...
      ...              Copy ten words from @FROM,I to @TO,I
      ...
      JSB $LIBX        Restore interrupts,
      DEF .NFER        and return external.
      ...
```

The entry sequencing is identical to that for .ENTR, except for three items:

1. A JSB $LIBR and its DEF <links> must appear in between the entry point and the JSB .ENTC.
2. Subroutine exit is not via JSB <entry>,I but is through $LIBX instead.
3. Since the caller places no DEF * +n+1 or DEF <return point> in his calling sequence, both caller and subroutine must agree on the number of parameters. And since .ENTC computes that number by comparing the DEF <links> to its own return address, no locations may be consumed between the last link and the subroutine's entry point.

.ENTC can also be used for re-entrant subroutines. The calling sequence in this case is similar:

```
        NAM .NFER,6  fmp Namr transFER.
        ENT .NFER
        EXT $LIBR,$LIBX
        ...
TDB     NOP               Re-entrant list header.
        ABS .TFER-TDB     TDB length (header+data+links)
@EXIT   NOP               Current return address.
DATA    BSS <whatever>    TDB data.
@FROM   MPX               Link to source.
@TO     MPX               Link to destination.
.NFER   MPX               Entry point/return address.
        JSB $LIBR         Turn off interrupts,
        DEF TDB           and go re-entrant, saving old TDB if any.
        JSB .ENTC         Resolve link, adjust return point.
        DEF @FROM         Starting with @FROM.
        STA @EXIT         Save adjusted return address in TDB.
        ...
        ...               Copy ten words from @FROM,I to @TO,I
        ...
        JSB $LIBX         Restore interrupts,
        DEF TDB           and have $LIBX return through @EXIT.
        DEC 0 or 1        as always.
        ...
```

The differences are:

1. We are using the re-entrant format of $LIBR and $LIBX.

2. We must include the links in the data placed in SAM during re-entry. To do this we must make the link-list part of the TDB. The easiest way is to place the TDB above the links and the entry point, so that the links become the final data items in the data portion of the TDB.

3. $LIBR puts the raw, uncorrected return address from our entry point into TDB word 3. Because we have to skip over parameter DEF's in the caller, we must replace it with the true return address, which is generously supplied to us by .ENTC in the A register.

6-6. PRIVILEGED & RE-ENTRANT MODE WITH .ENTP

Although .ENTC is a useful routine, it requires a calling sequence which can only be generated in assembly language, and which suffers from a lack of flexibility due to the requirement that caller and callee agree on the number of parameters. What we really need is a privileged/re-entrant version of our old friend .ENTR.

Such a routine exists, and is known as .ENTP. It is available in the same firmware as .ENTR. For the calling program, whether calling a privileged or re-entrant subroutine, the calling sequence is identical to .ENTR, i.e.

```
      ...
      EXT SIGMA           For example.
      ...
      JSB SIGMA           Call the subroutine.
      DEF ]SIG            Define the return point.
      DEF SUMX            Define each parameter.
      DEF SUMXX           Define each parameter.
      DEF @N,I            Indirect as required.
]SIG  EQU *               Resume execution in caller.
      ...
```

In the subroutine, the privileged entry sequence is identical to that for .ENTC, except that we call .ENTP instead:

```
       NAM SIGMA,6  Compute standard deviation.
       ENT SIGMA
       EXT $LIBR,$LIBX
       ...
@SUMX  MPX               Link to SUM of X values.
@SUM2  MPX               Link to SUM of (X**2) values.
@N     MPX               Link to Number of values in sums.
SIGMA  MPX               Entry point/return address.
       JSB $LIBR         Turn off interrupts,
       NOP               and go privileged.
       JSB .ENTP         Resolve links, adjust return point.
       DEF @SUMX         Starting with @SUMX.
       ...
       ...               Compute standard deviation.
       ...
       JSB $LIBX         Restore interrupts,
       DEF SIGMA         and have $LIBX return through SIGMA.
       ...
```

In the re-entrant case, the subroutine's sequence is also identical to that used for .ENTC:

```
       NAM SIGMA,6  Compute standard deviation.
       ENT SIGMA
       EXT $LIBR,$LIBX
       ...
TDB    NOP            Re-entrant list header.
       ABS SIGMA-TDB  TDB length (header+data+links)
@EXIT  NOP            Current return address.
TEMP1  MPX            TDB data.
TEMP2  MPX            ""
AVERG  BSS 2          ""
@SUMX  MPX            Link to SUM of X values.
@SUM2  MPX            Link to SUM of (X**2) values.
@N     MPX            Link to Number of values in sums.
SIGMA  MPX            Entry point/return address.
       JSB $LIBR      Turn off interrupts,
       DEF TDB        and go re-entrant, saving old TDB if any.
       JSB .ENTP      Resolve links, adjust return point.
       DEF @SUMX      Starting with @SUMX.
       STA @EXIT      Save adjusted return address in TDB.
       ...
       ...            Compute standard deviation.
       ...
       JSB $LIBX      Restore interrupts,
       DEF TDB        and have $LIBX return through @EXIT.
       DEC 0 or 1     as always.
       ...
```

When using .ENTC and .ENTP, keep in mind that the state of the caller's registers is not necessarily preserved, and the values to be found in the registers are only those documented in the manuals.

6-7. .ZPRV AND .ZRNT EXPLAINED

You have doubtless noticed that all the privileged and re-entrant forms generate an interrupt at $LIBR (unless already privileged) and the re-entrant forms generate another at $LIBX. If the routine is in the memory-resident library, we can tolerate all this interrupt overhead because we gain the space afforded by having only one copy of the subroutine for many programs.

However, if the subroutine is called by a disc-resident program, that program gets its own copy of the routine. In this case there is no need to generate the interrupt, for no one else will attempt to enter our copy of the code. It seems like a lot of CPU time is going to be wasted servicing needless interrupts.

What we need is a way to go privileged/re-entrant only when really needed, that is, only when the routine is re-located by RTxGN into the memory resident library.

Such a mechanism exists, and curiously enough, it uses the RPL capability discussed in chapter 3. It comes in two flavors:

1. .ZPRV which "replaces" $LIBR/$LIBX in PRiVileged routines.
2. .ZRNT which "replaces" $LIBR/$LIBX in Re-eNTrant routines.

The calling sequence for .ZPRV and .ZRNT is structurally the same as for $LIBR/$LIBX, and the easiest way to implement them is to write your routines for $LIBR/$LIBX and then transpose them to .ZPRV/.ZRNT format. For example, in our original simple privileged routine of section 6-3:

```
               NAM SUBPR,06       NAM SUBPR,06          ..same..
               ...                ...
               ENT SUBPR          ENT SUBPR             ..same..
               ...                ...
               EXT $LIBR,$LIBX    EXT .ZPRV             different
               ...                ...
        SUBPR  MPX                MPX                   ..same..
               JSB $LIBX          JSB .ZPRV             different
               NOP                DEF EXIT              different
               ...                ...
               ...                ...                   Code is the same.
               ...                ...
               JSB $LIBX    EXIT  JMP SUBPR,I           different
               DEF SUBPR          DEF SUBPR             ..same..
               ...                ...
```

The rules for a privileged subroutine are:

1. Replace the JSB $LIBR with a JSB .ZPRV.
2. Replace the NOP with the address of what was the JSB $LIBX.
3. Replace the JSB $LIBX with an ordinary JMP <entry>,I.

.ZPRV is not software or firmware. It is never executed. It is used to alert RTxGN that some special processing is required. That processing depends on where the subroutine is used. If:

M: relocated into the memory resident area, the JSB .ZPRV is converted to a JSB $LIBR, the DEF <exit> is converted to a NOP, and the JMP <entry>,I is converted to a JSB $LIBX.

P: relocated with a partition-resident program, the JSB .ZPRV is converted to a 002001B (RSS) instruction. This results in a skip over the DEF <exit> and no call to $LIBR or $LIBX. The routine acts like a normal type 7 library routine.

In any case RTxGN creates an RPL record in the system relocatable library of the form:

```
        .ZPRV,RP,002001
```

This tells the on-line LOADR to treat all relocations calling .ZPRV in the same manner as process "P" in RTxGN.

The .ZPRV convention is also compatible with subroutines which call .ENTC or .ENTP.

The procedure for re-entrant subroutines is similar. Using the example of section 6-4, the conversion is:

```
          NAM SUBRE,6           NAM SUBRE,6        ..same..
          ...                   ...
          ENT SUBRE             ENT SUBRE          ..same..
          ...                   ...
          EXT $LIBR,$LIBX       EXT .ZRNT          different
          ...                   ...
SUBRE     MPX         SUBRE     MPX                ..same..
          JSB $LIBX             JSB .ZRNT          different
          DEF TDB               DEF EXIT           different
          ...                   ...
          ...                   ...                code is the same
          ...                   ...
          JSB $LIBX             JMP SUBRE,I        different
          DEF TDB               DEF TDB            different
          DEC <ret>             DEC <ret>          ..same..
          ...                   ...
TDB       NOP         TDB       NOP                ..same..
          ABS EOD-TDB           ABS EOD-TDB        ..same..
@EXIT     NOP         @EXIT     NOP                ..same..
DATA      BSS <size>  DATA      BSS <size>         ..same..
EOD       EQU *       EOD       EQU *              ..same..
```

The rules are the same as for privileged routines, and the way the generator handles the routine is the same as well. For memory resident routines, RTxGN changes the code back to what is actually required ($LIBR/$LIBX), for disc residents it changes the JSB .ZRNT to an RSS, and it RP's .ZRNT to an RSS in the library of the system it is building. .ZRNT is also compatible with routines which call .ENTC and .ENTP.

Since it is not possible to load on-line a program which accesses the memory resident library, the RPL insures that the copy of the subroutine appended to the program will not cause an interrupt by needlessly attempting to go privileged.

Caution: When the .ZPRV substitution is used, the subroutine will execute privileged only if it is memory resident. When called by a disc resident (or partition resident in RTE-M) program, it will not be privileged, i.e. the interrupt system will be on. Because of this, routines which are calling $LIBR/$LIBX in order to perform otherwise illegal tasks cannot use .ZPRV.

Summary: At this point we know how to write:

1. Normal subroutines, which run with the interrupt system on at all times.
2. Privileged subroutines, which run with the interrupt system off.
3. Re-entrant subroutines, which can be shared by multiple programs.

We also know how to pass parameters:

1. In the working registers.
2. Through fixed call-adjacent memory (.ENTC)
3. Through variable call-adjacent memory (.ENTR,.ENTP)

However, all of this is at the assembly level. In the next chapter we will discuss the interface with the higher level languages.

BENCHMARKING HP 1000 COMPUTERS

By Charles G. Fugee/Dynalectron Corp.

INTRODUCTION

At the Dynalectron Radar Backscatter Division at the RAT SCAT site on Holloman AFB in New Mexico, we currently are using six separate Hewlett-Packard computer systems. These systems are used for software development, data collection, and data processing. An accurate reference guide for the comparision of computing speeds among the systems was needed. Such a guide could aid in estimating processing time, and also in creating more efficient data processing programs.

The Radar Target Scatter Site (RAT SCAT) is an Air Force facility with major objectives being defined as concise, complete radar target backscatter measurement, determination of antenna gain and radiation patterns, electronic system tests, and infrared target signature measurement for the Department of Defense and all government sponsered programs.

TESTING PROGRAM

A benchmarking program was designed and implemented to provide us with the desired performance guide. A program called BCHMK (hardly original, but effective), computes various mathematical functions a fixed number of times, input by the user. The functions (standard FORTRAN IV library functions, except for two, developed for use at the RAT SCAT site) are listed below. The functions themselves are not complex or difficult, mainly because our data processing does not involve complicated manipulations of the data, but only simple ones (like additions, subtractions, averaging and such) repeated many times. This would seem ample justification for a test designed as such. [Editor's note: A copy of BCHMK can be obtained by writing the author (see address at end of article).]

INT-RL	Converts integers to reals
RL-INT	Converts reals to integers
FPTADD	Performs floating point addition
FPTSUB	Performs floating point subtraction
FPTMPY	Performs floating point multiplication
FPTDIV	Performs floating point division
INT-DP	Converts integers to double precision
DP-INT	Converts double precision to integers
FPT-DP	Converts floating point to double precision
DP-FPT	Converts double precision to floating point
DP-ADD	Performs double precision addition
DP-SUB	Performs double precision subtraction
DP-MPT	Performs double precision multiplication
DP-DIV	Performs double precision division
FPTSIN	Evaluates floating point sine function
FPTCOS	Evaluates floating point cosine function
FPSQRT	Evaluates floating point square root
FPATAN	Evaluates floating point arc-tangent function
FPTLOG	Evaluates floating point logarithms (base 10)
FPTEXP	Evaluates floating point exponentiation
FPT-LN	Evaluates floating point logarithms (base e)
FPV-DB	Converts volts to dbsm
FPDB-V	Converts dbsm to volts

RESULTS

The results of the program (see tables and graphs) revealed that, from an overall standpoint, using the Hewlett-Packard 2117 F series machine as a reference, the 2117 is approximately six times faster than the HP-2113, nine times faster than the HP-2112, and fourteen times faster than the HP-2100 system.

Some interesting results may be noted in the comparison of these systems. It can be noted in figures 2 and 3, that the HP-2100 system surpassed the HP-2112 in processing speed when performing a floating point sine function, and when performing a floating point conversion from dbsm (decibel/sq.meter) to volts. In the dbsm to volts conversion the margin of difference is not that great but in the floating point sine function there is quite a significant difference, with the 2100 approaching the speed of the 2113. Overall, of course, the 2100 was by far the slowest, but it is interesting to note the differences when they occur in an unexpected direction.

MACHINE SPECIFICS AND TESTING PROCEDURES

Some specifics about the machines involved, and the programming and statistical analysis techniques used to draw these conclusions follow. Each machine involved was running a system generated to use all the available hardware and firmware features of the machine. Thus, in the case of the 2117, its floating point processing box gave it the natural advantage in this number crunching race.

The 2100 machines are used primarily for data processing due to certain restrictions on the visibility of processed data. Thus, since we want to process data without it being visible to other users on the systems at RAT SCAT, we cannot use any type of shared resources systems such as DS/1000. Each of the 2100's we use is tied in with its own HP plotter and two tape decks, and shares a line printer.

The other three HP-1000 machines are operating under RTE-IVA. It might be of interest to know that our 2117 system, used for real-time data collection and processing, controls four separate radar inputs, eight Houston flat-bed plotters, two user terminals, and a TV monitor. Also worthy of note, with RAT SCAT being a remote operation, on a twenty four hour day (sometimes seven day week) schedule, all maintenance is performed by our highly qualified, Hewlett-Packard trained, digital maintenance staff.

The comparative runs were made at several values of the input variable NLOOP which represents the number of iterations for each operation, specifically in this survey, at 100, 500, 750, 1000, 2500, 5000 and 10000 iterations for each operation. All runs were made when there were no other users on the system. Runs were made with two to three other users on the system but showed a difference of several milliseconds from those runs when there were no other users on line. I felt this difference might be significant, and therefore restricted my tests to be performed only when the system was free (consistency of experimentation being of great importance in statistical testing). The total time for completing the total number of iterations (in years, days, hours, minutes, seconds, and milliseconds) is computed and recorded for each operation. The computing time was found by using an EXEC 11 call before and after each function was executed NLOOP times. The difference was calculated and output to the listing device. In addition to this, the name of the system being surveyed (supplied at run time by the user) and the number of iterations were both included. Extremely complex mathematical functions were not used in this survey simply because they are rarely used at RAT SCAT. On the other hand, more mundane arithmetic operations (subtractions, divisions, averaging and the like) are used ad infinitum, thus increasing the need for a handle on how long we take to do this as well as how to do it faster and cheaper in the computing sense of the word.

STATISTICAL ANALYSIS

After obtaining a large bulk of the survey results, a statistical analysis was initiated with the statistical packages currently under development at the RAT SCAT site. Standard univariate regression procedures were performed on the data in hopes of establishing some kind of linear relationship between iterations and computing time. The components of the regression line may be explained as follows. The dependent variable (y) represents the computing time for any function given NLOOP. The independent variable, therefore, represents the value of NLOOP (the number of iterations), and the slope represents the susceptibility of the computing time to a change in the number of iterations. The intercept parameter bears little significance other than offering a reference for the computing time of any function for one iteration, if you add the value of the slope to the intercept value. This, however, does not hold true for some of the faster executing functions, as you will end up with a negative time. This is pardonable if we remember that the formulae are only estimates derived from experimentation which is prone to random error.

At this point it seems worthwhile to briefly explain the meaning of the value designated "R-squared" in the output. R-squared is the decimal representation of the percentage of variation in the data that is accounted for by the fact that there is a linear trend in the data. Thus an R-squared value of 0.825 indicates 82.5% of the variation of the data is explained by the fact that the data lies along a straight line with a non-zero slope. The results listed below, for several of the functions used in the survey show a fairly close linear relationship, which is again illustrated in the accompanying figures. I find it interesting to note how close the slopes for the 2117 are to zero, indicating that it takes a rather overwhelming change in the number of iterations to change the computing time for that total number of iterations, while, for a machine like the 2100, any change in the number of iterations produces a marked change in computing time for the total group of operations.

CONCLUSION

It can be noted that, in the case of the 2117, for a floating point addition, increasing the number of iterations by a value of 1000 only adds two milliseconds to the computing time, whereas the same increase for the 2100 increases the computing time by 49 milliseconds, a substantial difference. In comparing the 2112 and the 2113, it should be noted that for the four functions examined in the regression results listed, the 2113 surpasses the 2112 in every case but for a floating point addition. Considering the similarity of these two machines, the most probable cause of this difference is that the 2113 uses high performance memory, whereas the 2112 is using standard performance memory.

The aim of this paper is not to point out why some of the machines are slower but merely the fact that they are and to illustrate a little more explicitly, the differences between the machines. It might be of some interest to know more about why the 2113, with high performance memory, is not faster than the 2112 with standard performance memory on a floating point addition, while it beats out the 2112 everywhere else. Also, the reader may wish to utilize this information to optimize his or her programming in reference to processing speed. In addition, this information could be used as a purchasing guide user is interested in the machine for high speed processing.

The author's address is: Charles G. Fugee
Hewlett-Packard Company
P.O. Box 105005
450 Interstate N. Parkway
Atlanta, Georgia 30339

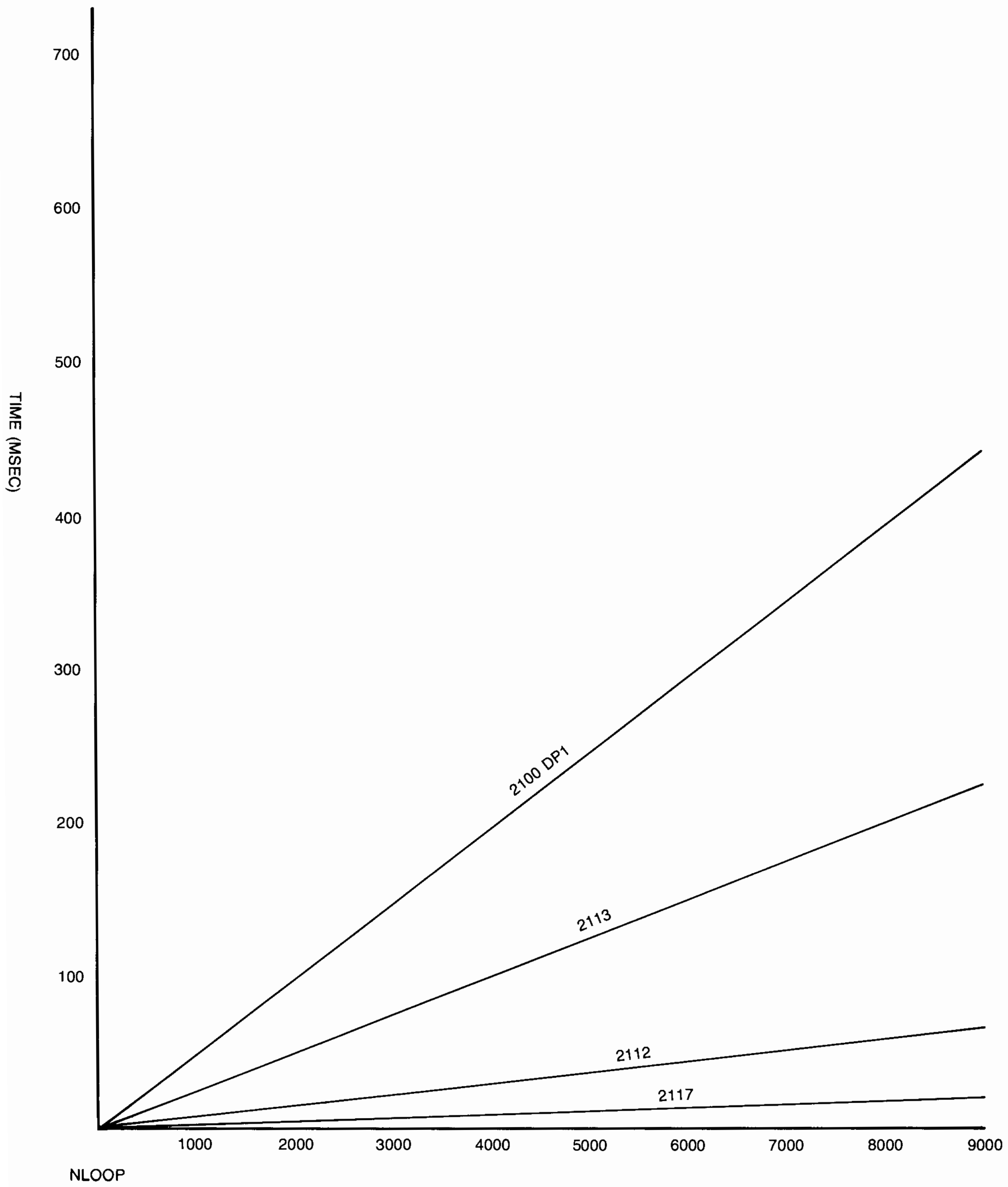


Figure 1. Double Precision Addition

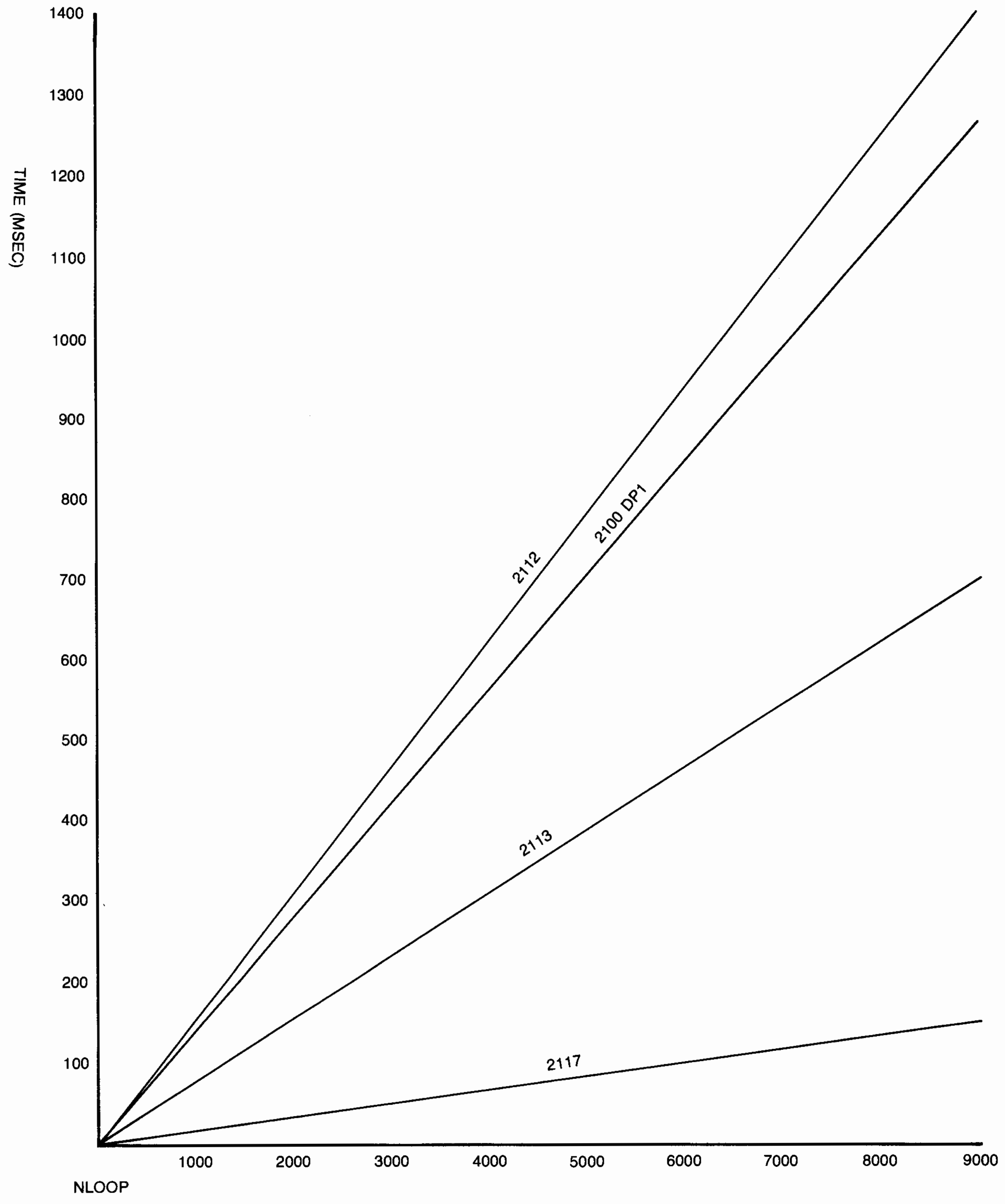


Figure 2. DBSM to Volts

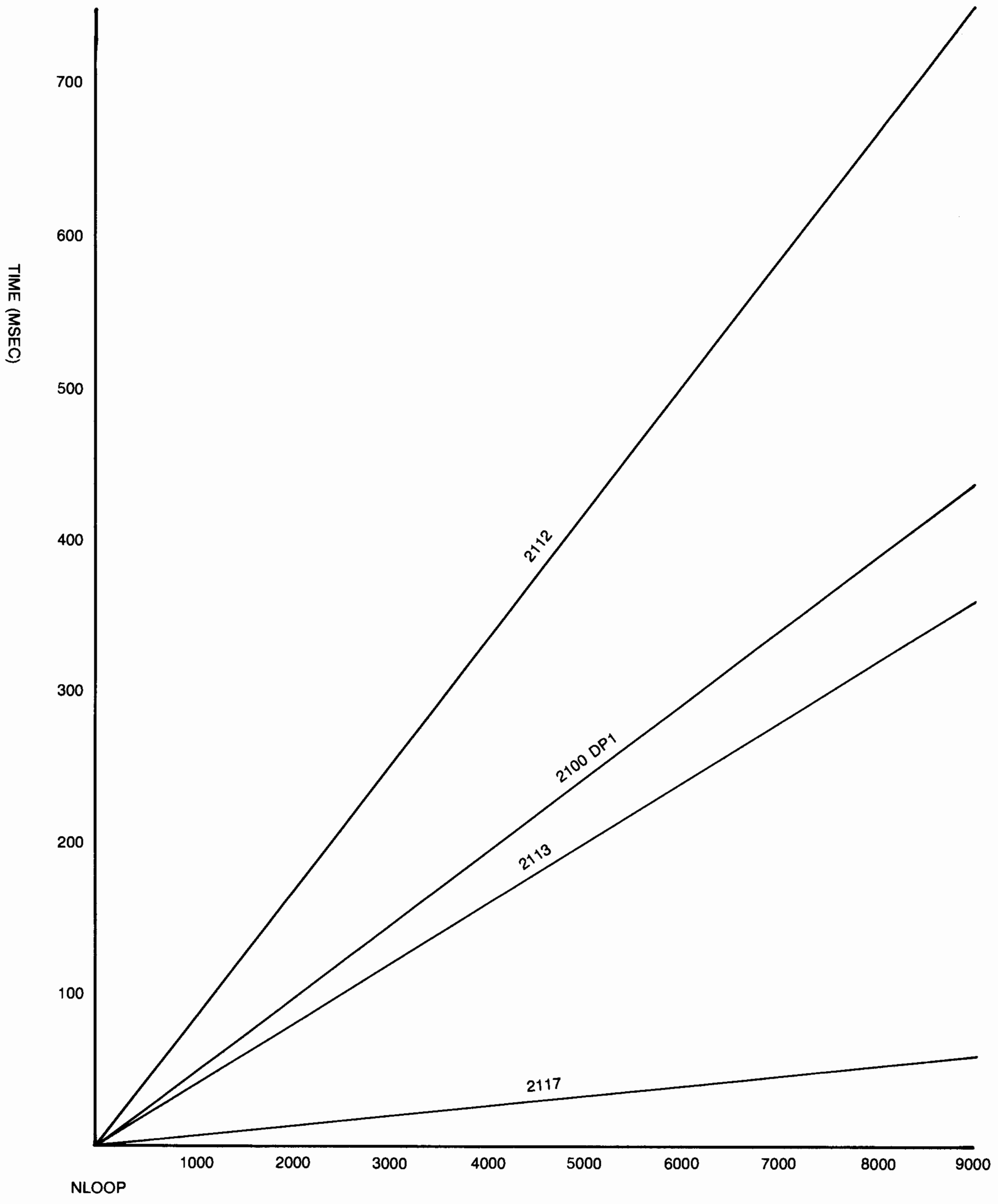


Figure 3. Floating Point Sine Function

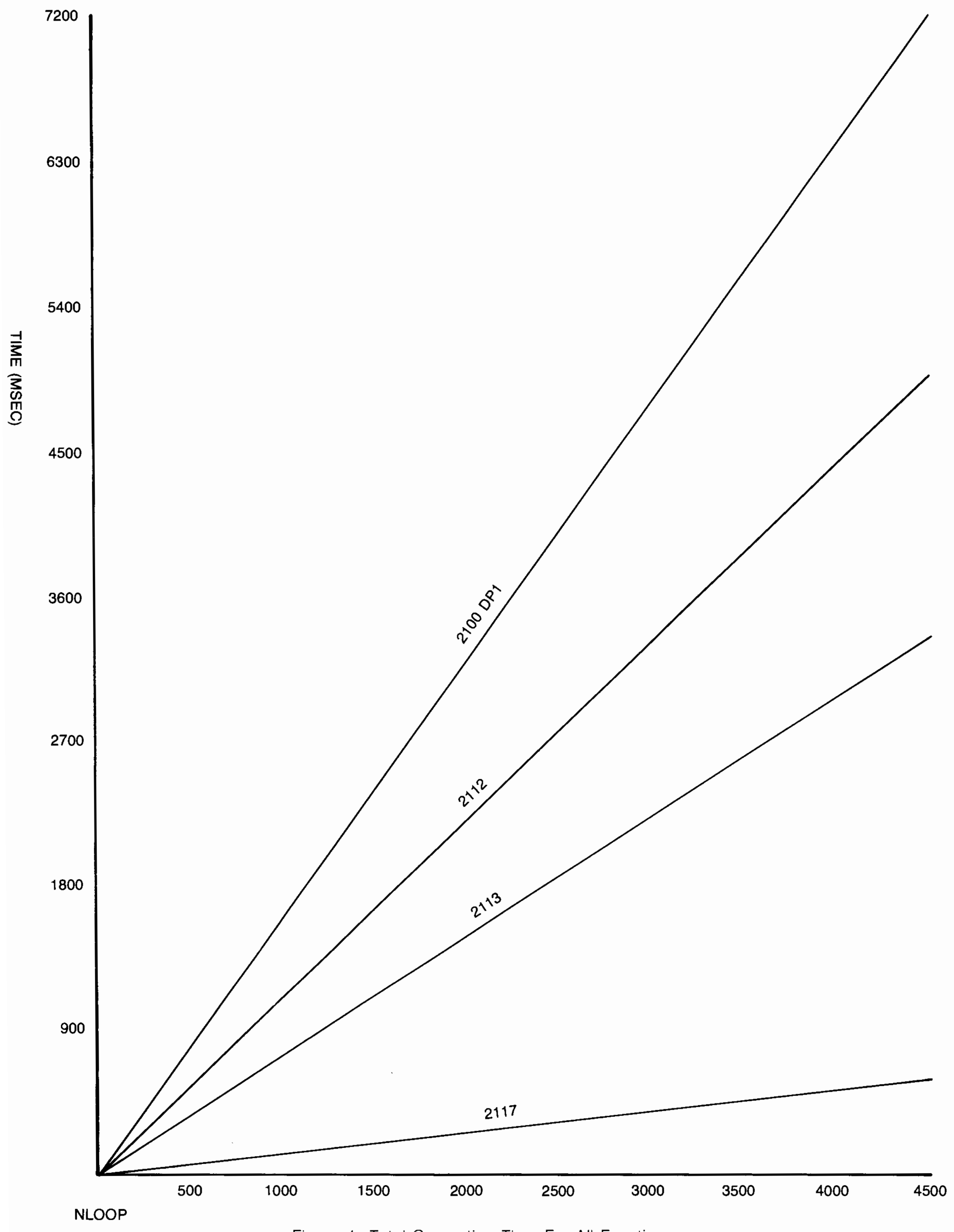


Figure 4. Total Computing Time For All Functions

Regression Results

HP 2100 Data Processing Machine #1

Function	Regression line formula	R-squared value
FPV-DB	y = 0.243 x + 0.767	1.000
FPDB-V	y = 0.138 x + 0.456	0.930
FPTSIN	y = 0.051 x + 0.275	0.771
FP-ADD	y = 0.049 x - 0.593	0.695

HP 2113 Data Collection Machine

Function	Regression line formula	R-squared value
FPV-DB	y = 0.138 x + 3.138	1.000
FPDB-V	y = 0.074 x + 5.902	0.917
FPTSIN	y = 0.040 x + 0.531	0.812
FP-ADD	y = 0.024 x + 0.489	0.715

HP 2100 Data Processing Machine #2

Function	Regression line formula	R-squared value
FPV-DB	y = 0.244 x - 0.111	0.930
FPDB-V	y = 0.138 x + 0.477	0.926
FPTSIN	y = 0.051 x + 0.233	0.675
FP-ADD	y = 0.049 x - 0.586	0.643

HP 2117 Data Collection Machine

Function	Regression line formula	R-squared value
FPV-DB	y = 0.026 x + 0.333	0.614
FPDB-V	y = 0.016 x + 0.149	0.543
FPTSIN	y = 0.006 x - 0.205	0.476
FP-ADD	y = 0.002 x + 0.415	0.420

HP 2112 Software Development Machine

Function	Regression line formula	R-squared value
FPV-DB	y = 0.278 x + 26.305	0.997
FPDB-V	y = 0.153 x + 2.232	0.919
FPTSIN	y = 0.103 x +108.461	0.793
FP-ADD	y = 0.087 x + 0.417	0.611

MODELING MULTIPLE LEVEL LINKED LISTS WITH IMAGE/1000

By Jim Burkett/HP Data Systems Division

INTRODUCTION

This article details some efforts expended to deal with multiple level linked lists using HP's IMAGE database management package. The application of the proposed method is focused on a problem faced by the ATS group at HP's Data Systems Division. Their database design and limited disc space restricted them to recording only ten options per product on an order when, in reality, more options were available. Benefits of the method include (1) reduced data redundancy, (2) reduced disc space required, and (3) applicability to other database design difficulties.

THE PROBLEM

The problem of limited number of options per item (10) on ATS systems was built into the Finished Goods Inventory (FGI) management system, and is built into other information processing systems at Data Systems Division. It was desired that the FGI management system be maintained, but that vital disc space be recovered. The subject of this article is how the FGI database was redesigned to allow for more than 10 options per product, and yet use less disc space rather than more.

THE SOLUTION CONCEPT

The constraint of allowing for a maximum of 10 options resided in a limitation of IMAGE/1000 to handle multi-level linked lists (Editor's note: a multi-level linked list would essentially be a detail dataset which serves as a master for another dataset.) This difficulty was overcome by implementing a user-invisible link from the product record in one detail dataset to an option combination chain in another detail dataset. This method condensed the data necessary to express option combinations and eliminated the redundancy of saving an option combination string for each product record.

The original method (see figure 1) imbedded into each product a 30-character string capable of holding 10 option numbers. That is; for every record in the database with a product number entry, there was a 30-character field to hold its associated options. The new method (see figure 2) separates the option strings from the product records and creates a chain of the individual options, each chain representing one original option string. This chain is then linked to the product record by an option combination key (through the option combination master dataset).

This method stores each unique option combination chain in the database only once. Product records with the same options are linked to the same option combination chain. There is a unique chain for each option combination but, there is not a unique chain for each product record. This approach can save a significant amount of disc space by eliminating data redundancy.

THE CONCEPT

ORIGINAL METHOD

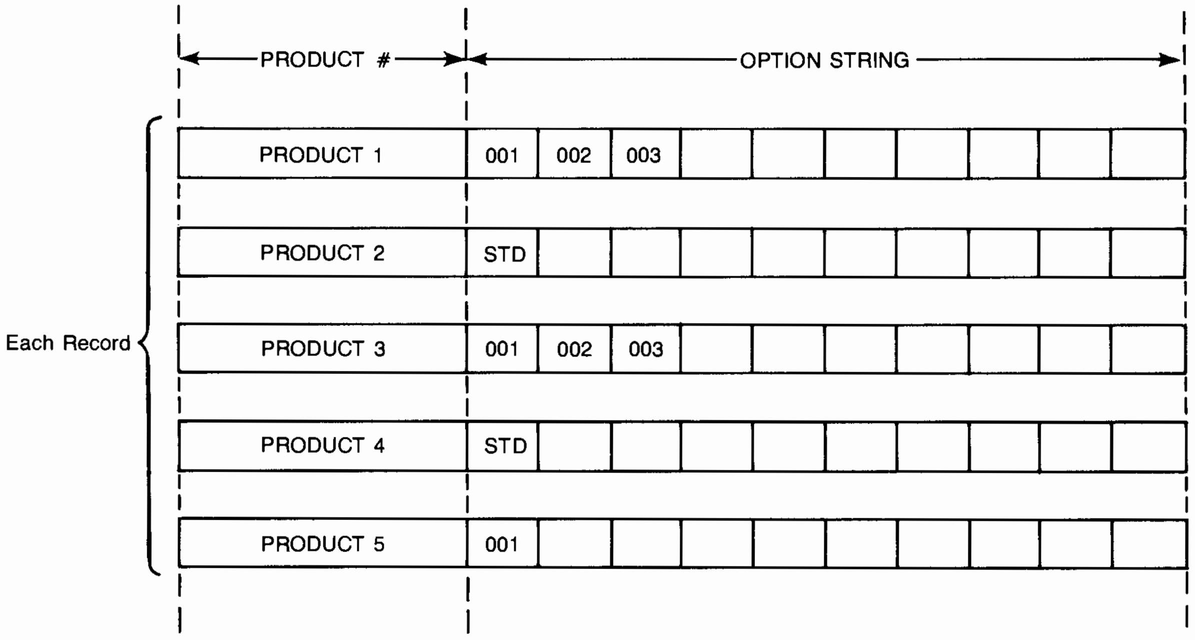


Figure 1

PROPOSED METHOD

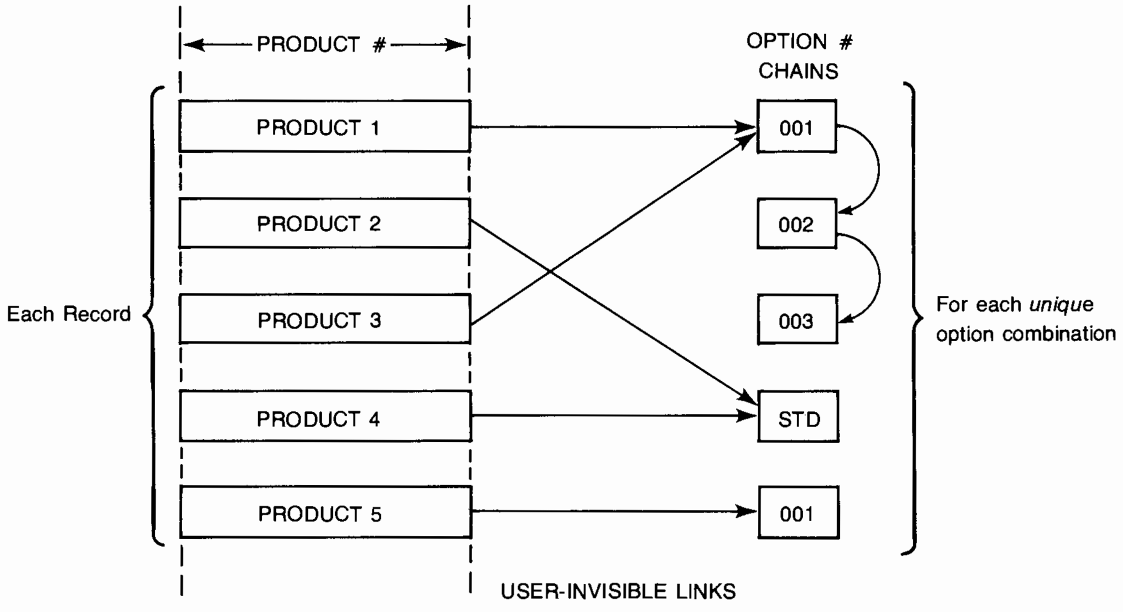


Figure 2

THE DATABASE

The link from a product record to an option combination chain in IMAGE/1000 was implemented with three automatic master datasets and two detail datasets.

The product number (PRODNO) is the key item from the product master dataset (PRDNUM) to the product detail dataset (PRDDET). The link to the option detail dataset (OPTDET) is maintained by an option combination key (OPTNKY). The option combination master dataset (OPTCOM) uses its key to link both the product number and option number detail datasets. An additional feature of the database schema is an option number master dataset (OPTNUM). Through this dataset a list of option combination chains containing a particular option number can be determined, and provide a linkage from the option number record to all associated product number records with that option (see figures 3 & 4).

THE DATABASE VIA IMAGE/1000

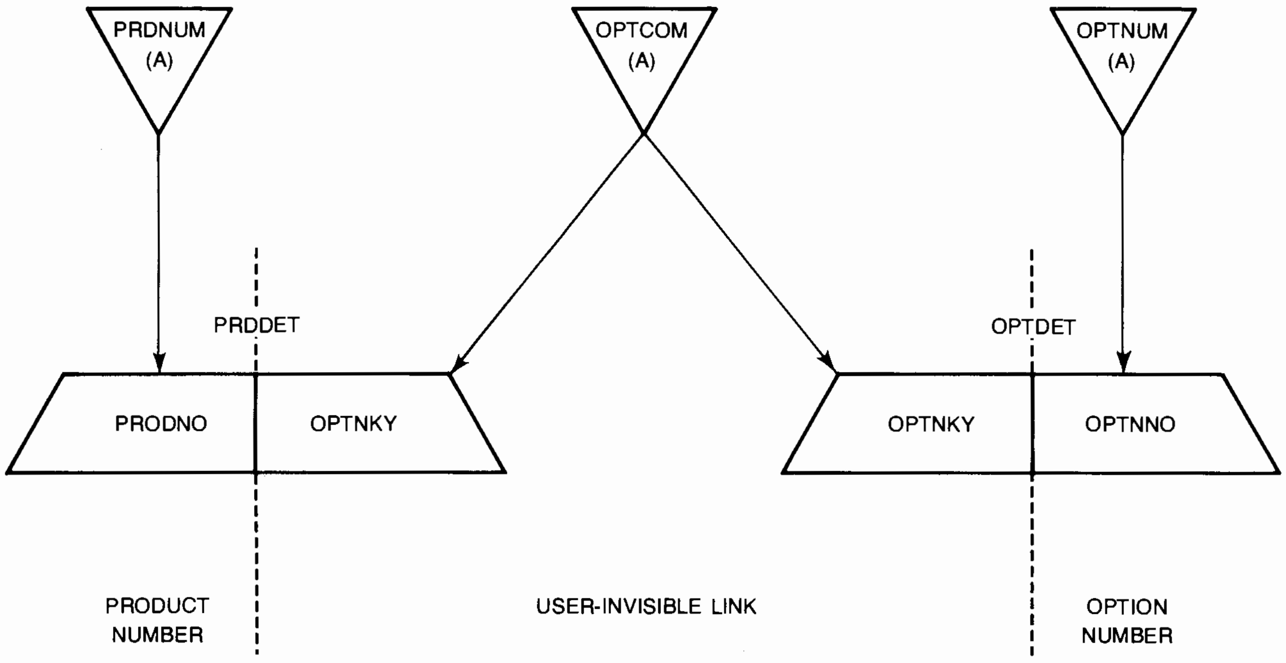


Figure 3

DATABASE IMPLEMENTED FOR PROGRAM TEST

```
HEWLETT-PACKARD IMAGE/1000 DATA BASE DEFINITION PROCESSOR

$CONTROL:;
BEGIN DATA BASE:
 FGIMGT:12:JB;  << FGI MANAGEMENT SYSTEM DATA BASE >>
LEVELS: ;
ITEMS:
 OPTNKY, I1;  << OPTION COMBINATION KEY   (PRDDET,OPTMAS,OPTDET) >>
 OPTNNO, X4;  << OPTION NUMBER            (OPTDET) >>
              <<                                   >>

SETS:
   << OPTCOM: OPTION COMBINATION MASTER --- (OPTNKY) >>
NAME: OPTCOM::JB,A;
ENTRY: OPTNKY(1);
CAPACITY: 150;
   << OPTNUM: OPTION NUMBER MASTER --- (OPTNNO)  >>
NAME: OPTNUM::JB,A;
ENTRY: OPTNNO(1);
CAPACITY: 100;
   << OPTDET: OPTION COMBINATION DETAIL --- (OPTNKY) >>
NAME: OPTDET::JB,D;
ENTRY: OPTNKY(OPTCOM),
       OPTNNO(OPTNUM);
CAPACITY: 300;
END.

NUMBER OF ERROR MESSAGES: 0000
NUMBER OF ITEMS: 002
NUMBER OF SETS: 03
ROOT FILE:  00148 WORDS,  00005 BLOCKS

CARTRIDGE NUMBER                        NUMBER BLOCKS REQUIRES
         19010                                0000000059


END DATA BASE DEFINITION
```

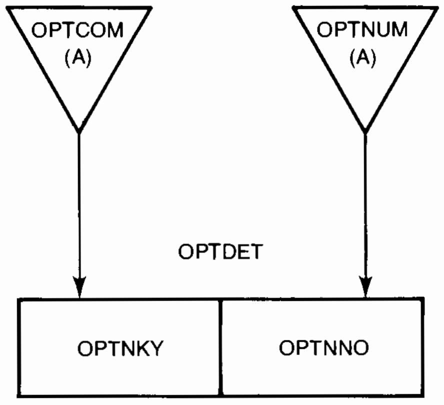


Figure 4

THE PROGRAM

A PASCAL/1000 program was written to test the concept. Input data was obtained from a strip tape currently used for the FGI management system, FGI/1000. As a first pass, only the option combination chain datasets were defined in the schema to sufficiently test the theory. A flow chart of the program is shown below.

TEST PROGRAM

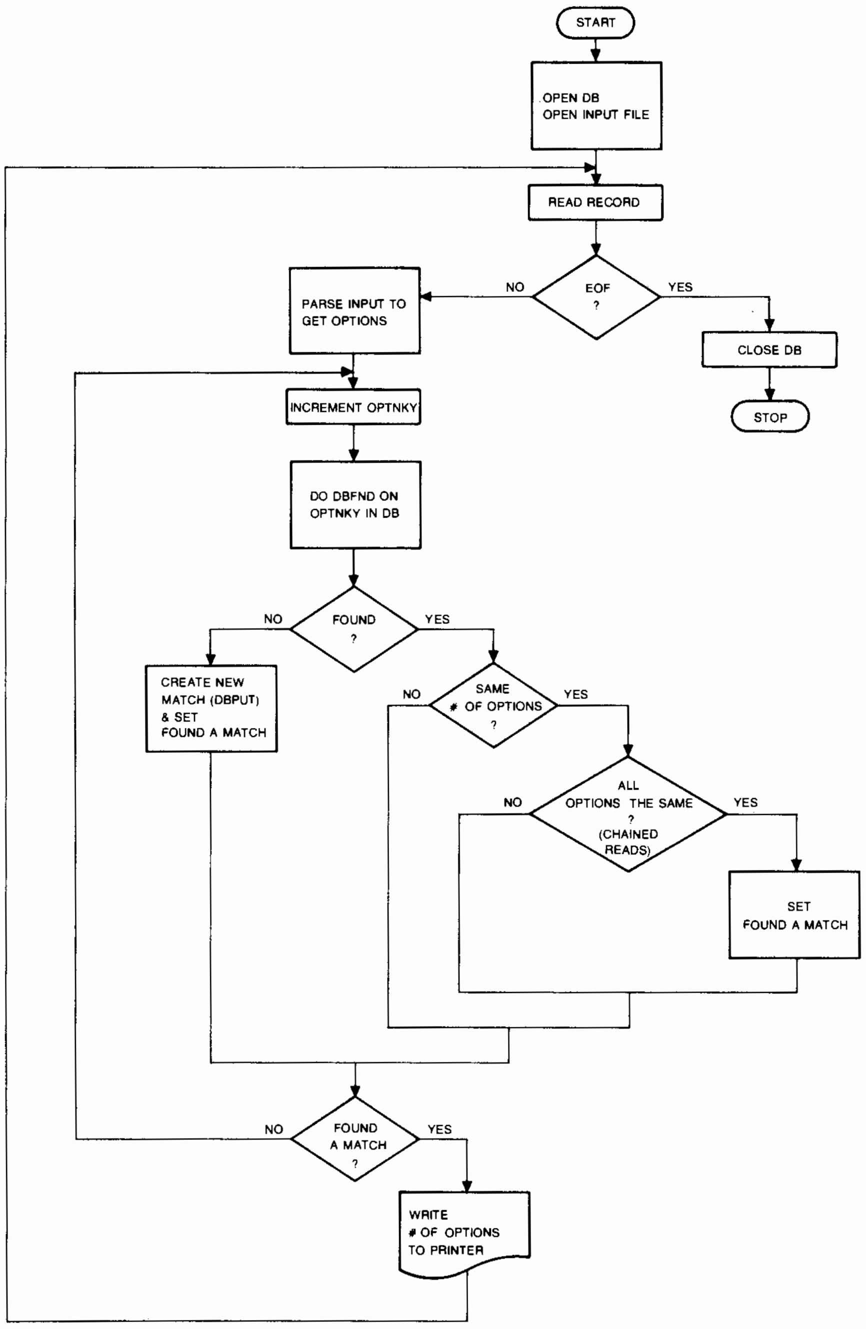


Figure 5

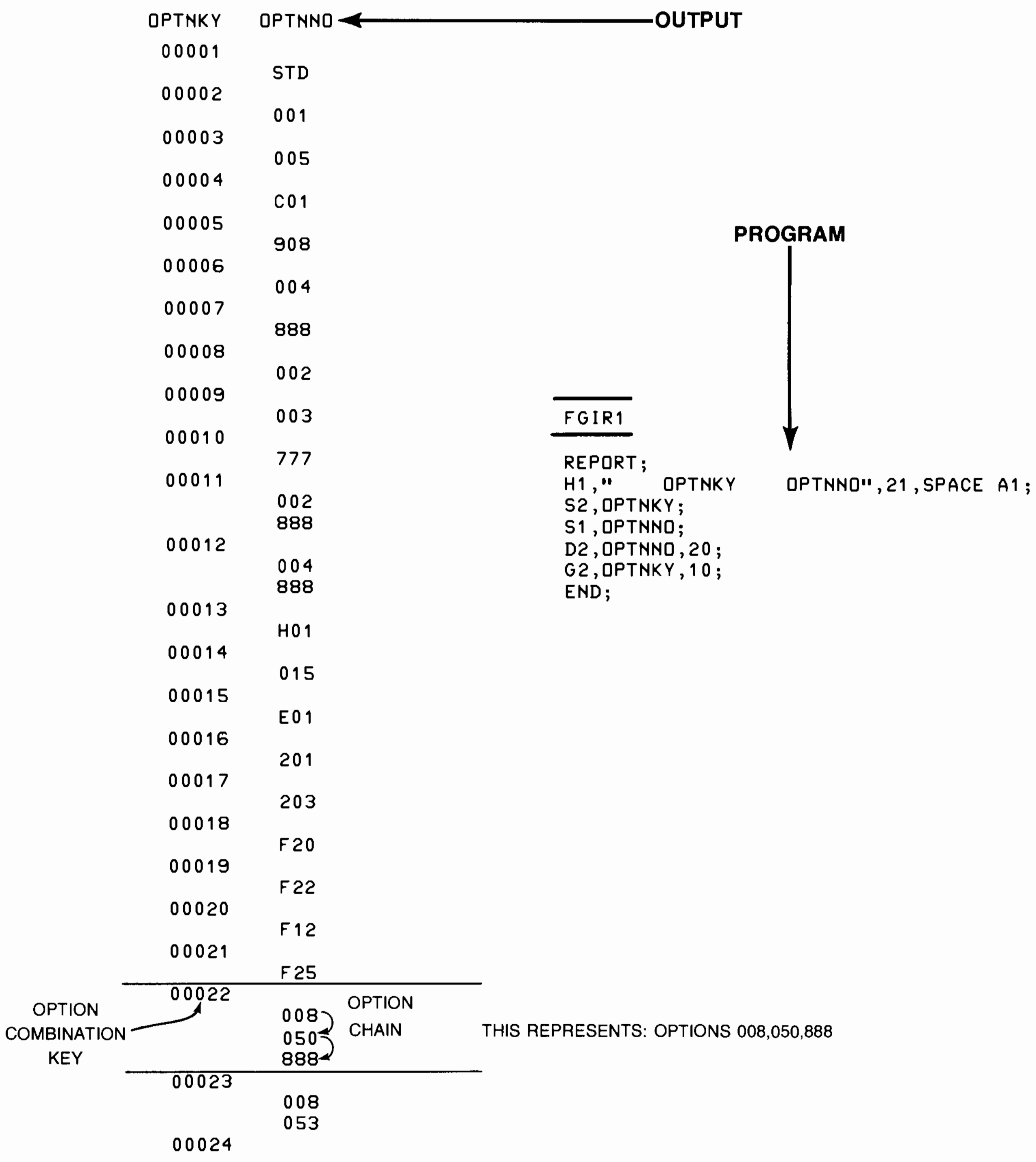
THE RESULTS

It worked! The FGI/1000 tape records were serially processed and feedback information was reported to the user in 50 record chunks. (Each line of output contained 50 digits, one digit for each record processed. Each digit represented the number of individual options associated with that record. The standard configuration was counted as one option.)

```
SUCCESSFUL DBOPN
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
11111111111111111112112111111111111111111111111111
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111132231111211
11111111111111111111121111111121321121112434444341
12322222321211111211111121113121311131111115311111
11212112111112111111111112143242124111111111111111
11111111111111111212111111131112111111111116321111
12111111111111111121111111111111111132341111111111
1111111111111111111111111111111112212211121
           543 TOTAL RECORDS PROCESSED
 SUCCESSFUL DBCLS, BYE!
```

Figure 6

Two summary reports were generated via QUERY/1000 to validate the process. "FGIR1" produces a report of each option combination key (OPTNKY) and its associated chain of option numbers (OPTNNO). "FGIR2" produces a report of each option number's (OPTNNO) appearance in a separate option combination chain (OPTNKY).



```
OPTNKY     OPTNNO
 00001
             STD
 00002
             001
 00003
             005
 00004
             C01
 00005
             908
 00006
             004
 00007
             888
 00008
             002
 00009
             003
 00010
             777
 00011
             002
             888
 00012
             004
             888
 00013
             H01
 00014
             015
 00015
             E01
 00016
             201
 00017
             203
 00018
             F20
 00019
             F22
 00020
             F12
 00021
             F25
 00022
             008
             050
             888
 00023
             008
             053
 00024
```

```
FGIR1

REPORT;
H1,"     OPTNKY     OPTNNO",21,SPACE A1;
S2,OPTNKY;
S1,OPTNNO;
D2,OPTNNO,20;
G2,OPTNKY,10;
END;
```

Figure 7

OUTPUT

```
OPTNNO    OPTNKY
 001
            00002
            00026
            00030
            00034
            00055
            00057
            00092
            00096
            00097
            00098
            00102
            00106
            00107
 002
            00008
            00011
            00034
            00060
            00094
            00102
            00103
            00104
 003
            00009
            00094
            00103
 004
            00006
            00012
            00057
            00099
            00104
 005
            00003
            00030
            00075
            00076
            00096
            00099
            00102
            00103
 006
            00095
 007
            00036
            00037
            00038
            00039
            00040
 008
            00022
```

OPTION NUMBER

OPTION COMBINATION KEYS

THIS SIGNIFIES THAT OPTION 007 WAS ASSOCIATED WITH THE FIVE OPTION COMBINATION KEYS LISTED

PROGRAM

```
FGIR2

REPORT;
H1,"    OPTNNO      OPTNKY",21,SPACE A1;
S2,OPTNNO;
S1,OPTNKY;
D1,OPTNKY,20;
G2,OPTNNO,10;
END;
```

Figure 7 (continued)

543 data records were processed from the FGI/1000 tape, however, only 111 unique option combinations were encountered. The table below contains additional data:

Table 1

# OF OPTIONS IN CHAIN	# OF RECORDS ENCOUNTERED	% OF TOTAL RECORDS	# OF UNIQUE COMBINATIONS	% OF TOTAL COMBINATIONS
1	475	87.5	49	44.2
2	40	7.4	38	34.2
3	16	2.9	15	13.5
4	10	1.8	7	6.3
5	1	.2	1	.9
6	1	.2	1	.9
7	0	.0	0	.0
8	0	.0	0	.0
9	0	.0	0	.0
10	0	.0	0	.0
Total	543	100.0	111	100.0

475 of the 543 records processed (87.5%) required only one option of which there were only 49 unique combinations. Therefore, space originally allocated for 475 30-character option strings was contained in a space of only 49 options. A reduction of 99.0% of the space previously required! Overall, 209 option number records were needed for the entire FGI/1000 tape data. A savings of 96.2%.

In terms of improvement to the FGI management system, 530 KBytes of disc storage were released for other usage. Potentially, 5.730 MBytes of disc space for option storage in the database could be reduced to approximately 12 KBytes. This is a potential reduction of 99.8%, or the equivalent of 10% of a 7920 disc. In terms of monthly disc space cost savings, 5.730 MBytes per month at $.0061 per sector is about $270, or annually, over $3200.

CONCLUSIONS

1. The problem of a 10 option limit per product can be overcome. The only limitation to option combination chains is the capacity of the dataset to hold the chains.

2. Substantial amounts of disc space can be saved. Not only in this application, but in others as well.

3. A significant difficulty with IMAGE/1000 can be bypassed. Multi-level linked list applications can be modeled within the limits of IMAGE/1000 by implementation of internalized user-invisible links.

INCREASING SYSTEM AVAILABILITY THROUGH REDUNDANT COMPONENTS

Jim Bridges/HP Data Systems Division

INTRODUCTION

Computer systems often perform vital roles in the operations of many businesses. It is hard to imagine a commercial airline without a system to manage reservations. In manufacturing, computer systems are being used to control the production of most of the goods we purchase; everything from automobiles to zippers.

In many cases the role of the computer system is so important that its failure causes major disruption leading to loss of profitabilty, customer dissatisfaction and possible human injury. Therefore, system designers put a great deal of effort in devising techniques to ensure that the system will not fail at a critical moment or, if failure occurs, that it will be of short duration. In other words, they design for high availability.

This article will present some of the tools which HP Data Systems can provide to the system designer interested in high availability systems. It will also cover the general concepts involved in building such systems. A basic list of these tools is given below.

Hardware:

93550A	I/O Switch Unit
93768A	Watchdog Timer
93762A	Processor Interconnect Kit
93770A	Time-of-day Clock/Time Base Generator
13037B	Multi-ported Disc Controller

Software:

Drivers for the above, plus
93581C Twin-Disc Driver Manager

The 13037B Disc Controller is a standard product. The other items are available through your HP sales representative. Contact your local HP office for more information.

AVAILABILITY DEFINED

System availability basically means the percentage of time that the system is 'up' or operational. Availability can be predicted on a statistical basis if the failure rate can be reduced to a constant. The failure rate is usually expressed as a percentage of the units which fail per 1000 hours of operation.

During its normal lifetime a product is either available (able to be used) or unavailable (being repaired). The availability thus becomes:

$$A = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

where

MTBF = mean time between failures
MTTR = mean time to repair

Notice that the the availability may be increased either by increasing the MTBF or decreasing the MTTR.

Since any component is either available or unavailable, the sum of the availability and unavailability is 1 (or 100%). In equation form, this is:

$$A + U = 1$$

which gives

$$U = 1 - A$$

or

$$U = \frac{MTTR}{MTBF + MTTR}$$

Basically, these equations mean that a system spends all its useful life in either of two states: 'up' or 'down'.

The MTBF is a widely used measure of comparison between various components. It may also be used to evaluate the cost effectiveness of any measures designed to improve the failure rate, i.e. how much do you get for what it costs.

The MTBF is not a specification for a product. Nor will HP guarantee an MTBF for any product. Since it is a number derived by statistical methods, it is not possible to verify the accuracy of MTBF figures by measurement unless a large number of units are included in the sample. Individual units will always be found that perform much worse or much better than the MTBF.

The sample calculations in this article will use MTBF figures which are only approximately correct and have been chosen primarily to make the calculations easier. Actual MTBF figures for almost any HP product are available from the division which manufactures the product. The MTBF will be quoted only in situations where a need to know exists and there is reasonable assurance that the number will not be misused.

MTTR is a term which varies in definition. Some people include travel time to the site, while others measure only actual repair time. Repair time figures are available from HP and are based upon the reports turned in by Customer Engineers. For any calculation of availability or unavailability, the MTTR must include the entire time the component is not working. Typically, a failed component will result in a service call the next day. Therefore, in the sample calculations in this article, we will use an MTTR of 24 hours.

RELIABILITY

Reliability is defined as the probability that a component remains available after a given time period since the last repair cycle. As the time since the last repair increases, there is a decreasing probability that the system will continue to operate without another failure.

The probability of success at any point in this time period is given by:

$$P = e^{-ft}$$

where

P = probability that the component is available.
t = time since the last repair.
f = 1/MTBF.
e = base for natural logarithms.

The MTBF and time must be in compatible units (usually hours). The probability P ranges from 1 at start of the component's life and decreases exponentially toward 0. Usually it is quoted as a percentage value. For example, at time t =MTBF the probability that the component is still available is

$$P = e^{-1} = 36.7\ \%$$

We say that the system is 36.7% reliable at this point. More correctly, in a large sample, only 36.7% of the systems would still be up at this time.

The failure rate is usually expressed as a percentage of the total units which fail per 1,000 hours of operation. For example, for an MTBF of 10,000 hours:

$$\begin{aligned} f = \frac{1}{10{,}000} &= .0001 \text{ failures/hour} \\ &= .1 \text{ failures/thousand hours} \\ &= 10\% \end{aligned}$$

EFFECT OF ADDING COMPONENTS ON AVAILABILITY

The addition of components can either increase or decrease net availability, depending upon how they are added. Adding components in series is similiar to adding links in chain. If one link in the chain breaks, then entire chain is lost. The combination of a disc, a terminal and a CPU are in series because if any one fails, then the entire system fails.

The other approach is to combine components in parallel such that there are multiple paths for success. Thus if a system has two discs and can operate with only one of the pair, the discs are said to be in parallel. The components are sometimes referred to as "redundant".

A system with components in series has a net availability which is less than the availability of any of its components, i.e. it has a lower MTBF. The MTBF for a system is derived from the calculation of the probability that the system is still up after a given time. In this case, the individual probabilities are multiplied. For example, assume two components with MTBF of M hours and N hours. Then

$$P_{\text{system}} = e^{-t/M} \times e^{-t/N}$$

The net MTBF is given by:

$$\frac{1}{\text{MTBF}} = \frac{1}{M} + \frac{1}{N}$$

If the individual MTBFs are equal, then the net MTBF is half that of either component.

Note that, since the reciprocal of the MTBF is merely the failure rate, this equation says that the net failure rate is the sum of the individual failure rates.

For example, let's calculate the net MTBF for a system with a CPU, disc and a terminal (using approximate numbers).

Component	MTBF	Failure Rate (% per 1000 hours)
Disc	6000	16.67
CPU	7000	14.29
Terminal	8000	12.50
Net failure rate =		43.46

The equivalent MTBF is 2300 hours.

The computed MTTR will be a weighted mean of the individual MTTR's. However, we will assume that a service person actually arrives the day after the failure and fixes the problem within 24 hours after the system goes down.

Using MTTR = 24 hours, the availability of the above system becomes:

$$A = \frac{2300}{2300 + 24} = 98.97\%$$

$$U = \frac{24}{2300 + 24} = 1.03\%$$

Now let's compute the availability of two of these systems connected in parallel. For the moment, assume that equivalent inputs are tied together on the two systems.

To find the MTBF for parallel components, the mathematics is a little more complicated. To find the probability of successful operation, we first find the probability that a failure of one component is followed by a failure of the second component. This becomes the probability that the combination fails. As the probability of success goes down the probability of failure goes up. Thus

$$P_f + P_s = 1$$

where

P_f = Probability of failure

P_s = Probability of success

Thus

$$P_f = 1 - e^{-t/M}$$

For parallel systems we would find the net probability of failure as follows:

$$P_{system} = P_{f1} \times P_{f2}$$

where P_{f1} and P_{f2} are the individual probabilities of failure.

The net MTBF is not easily expressed in terms of the individual MTBFs. However, the availability calculation is much simpler.

$$A_{system} = 1 - U_{system}$$

$$U_{system} = U_1 \times U_2$$

We will ask you to accept the latter formula without derivation. It is analogous to the formula for computing net probability of failure.

If the two systems are identical, then the availability is

$$A_{system} = 1 - U^2$$

Plugging in the numbers for the previous system (MTBF of 2300 hours and MTTR of 24 hours), we have:

$$A_{system} = 1 - \left| \frac{24}{2300 + 24} \right|^2 = 1 - (.0103)^2 = 98.98\%$$

This is equivalent to a down time of less than 1 hour per year (24 hours/365 days) on a statistical basis. This refers to the period of time during which both systems are unavailable.

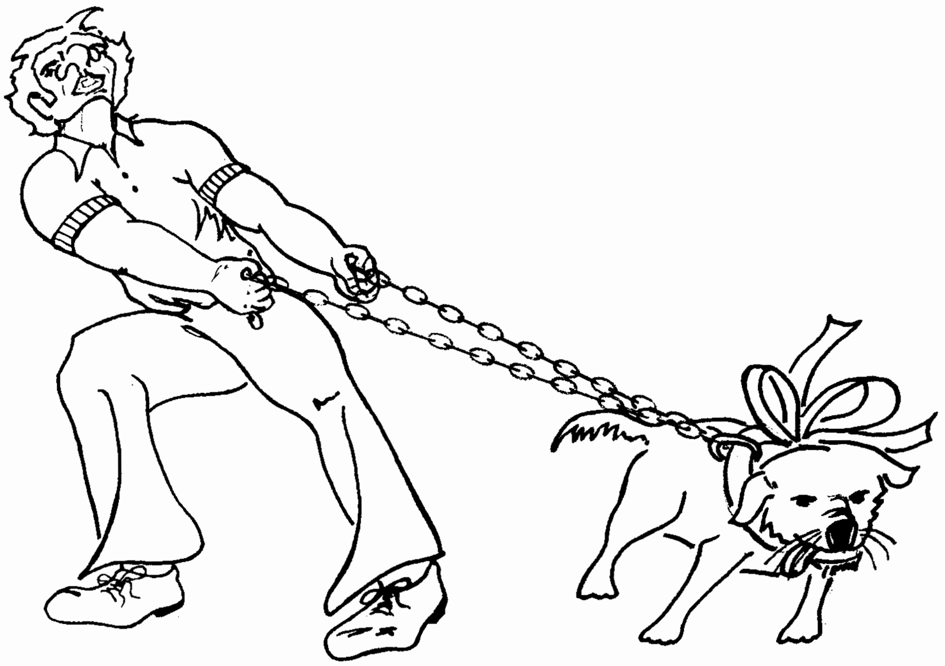


Figure 1. Series versus Parallel, Components. The leashes are in parallel but the collar is in series

METHODS OF COMBINING SYSTEMS FOR HIGH AVAILABILITY

Parallel (redundant) systems are joined together at the I/O points. The method of joining the systems must be highly reliable since the system is not useful (effectively down) if it cannot access the I/O devices.

With HP systems, there are three common methods of putting two systems in parallel.

1. Parallel Sensors. For example, if temperature measurements are made through thermocouples, then each system will have its own thermocouples. This method is preferred since everything is parallel except the device being measured.

2. The 12979B Switchable I/O Extender. See figure 2. The extender contains the I/O bus and interface cards and may be switched to either CPU.

3. The 93550A I/O Switch Unit. See figure 3. The I/O cards and I/O backplane are duplicated on each CPU. The I/O switch unit provides a dual port into the I/O device.

The 93550A Switch Unit is a special whereas the 12979B is a standard product. As such, the 12979B is more well known and lower priced. But the 93550A has a much greater MTBF and is therefore a better choice. Either the I/O extender or the switch unit are placed in series with the parallel systems; therefore, the MTBF of the extender or the switch (whichever is used) limits the net MTBF of the entire system.

The primary reason for the better MTBF of the 93550A is that it is basically a mechanical device with very simple electronic control and/or manual control (via front panel). The primary elements of the 93550A are:

- Multiple-pole relays for switching between redundant interfaces in the two computers.
- Simple switching logic (about 25 integrated circuits).
- A small power supply used only to control the relays and the associated switching logic.

By contrast, the I/O extender must have a very large power supply to support the requirements of the various interface cards which mount in the extender. The I/O cards are not redundant and therefore, also tend to limit the MTBF of the configuration.

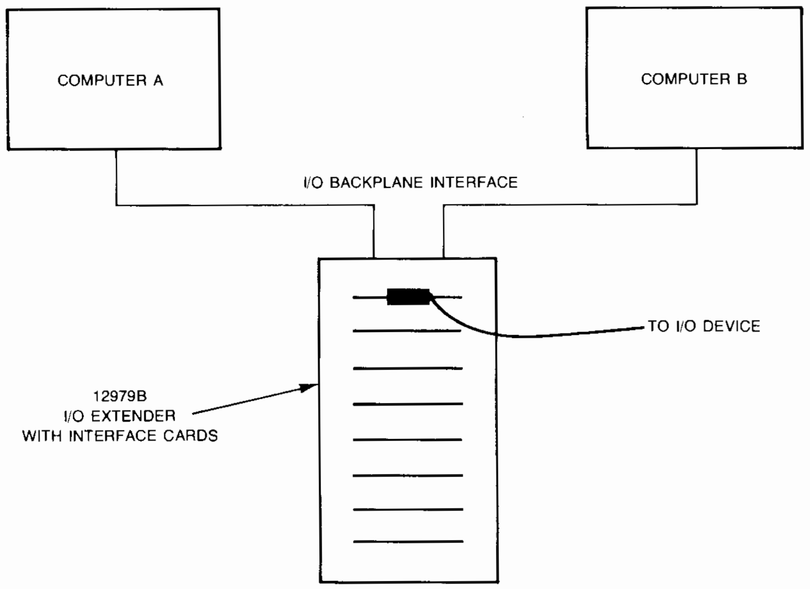


Figure 2. The 12979B Switchable I/O Extender. The extender supplies power for the interface cards. Interface cards are not redundant.

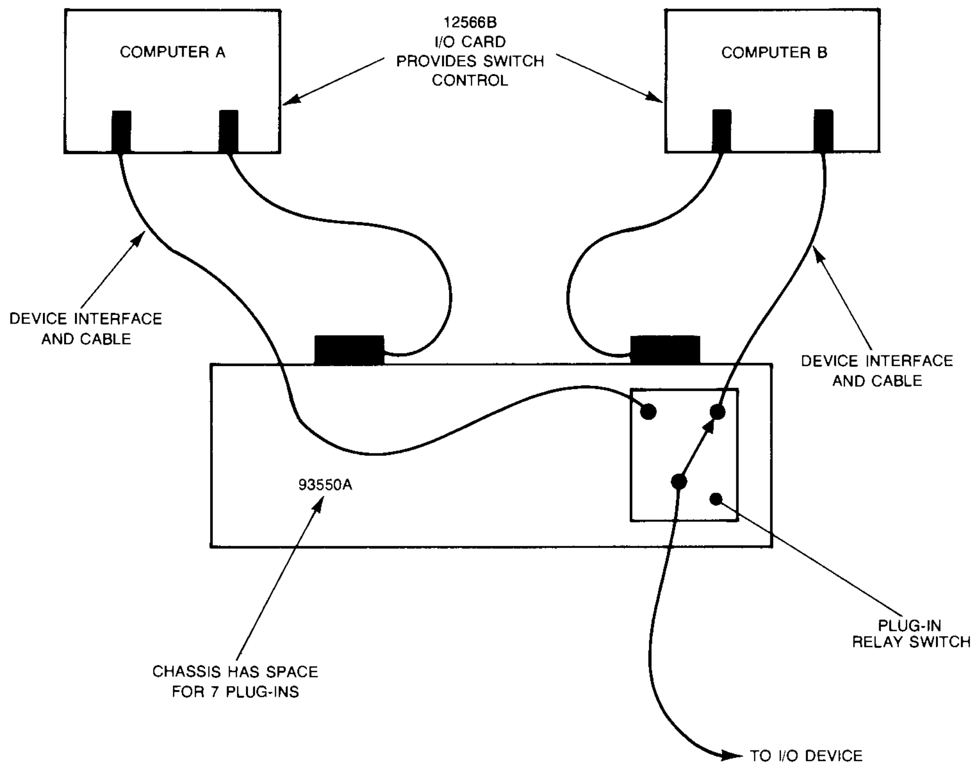


Figure 3. The 93550A I/O Switch Unit. The unit supplies power only for its switching logic. Interface cards are redundant.

DISC USAGE

If the two systems in parallel use the multi-ported controller (13037B) to interface to the disc drives, then we can couple both CPUs to both discs as shown in figure 4. This increases the availability because it increases the number of successful paths; i.e. either CPU can use either disc.

In addition, this configuration provides the potential to avoid switchover to the other CPU upon disc failure. While we must plan to make CPU switchover possible, it is a good idea to reduce the chances of its occurrance because there is usually some penalty. For example, even with no data loss on switchover, there will be some delay for switchover to take place.

Consider what happens if the multi-ported connection is not used: a failure of any component on one system followed by a failure of any component on the second system causes the configuration to fail. (Consider the components as disc, CPU and terminal.) However, with the multi-ported connection, a disc failure on one system must be followed by a failure of any other component on that system before switchover to the other CPU is necessary. A third failure must then occur before the new configuration fails.

The potential to avoid switchover upon disc failure is realized only if we can treat both discs as a single disc with high availability. But we can do this only if the contents of both discs are always identical. Then it won't matter which disc fails. The twin disc driver (HP 93581C) is the software tool that makes this possible.

The twin disc driver is actually a driver manager, although it appears as a driver to the system and to all programs running in the system. All requests which would normally go directly to the disc driver go instead to the manager. Neither the system nor any program realizes that this is happening. The manager calls the actual disc driver as a subroutine to do the actual I/O operation.

If the I/O operation is a write, then the buffer given to the manager is written to both discs (in sequence). If the operation is a read, then only one disc is read since the contents of both are identical. The driver manager will chose alternate discs for reads so that each disc is used equally.

The key to continuous operation is the state table which is maintained for the discs. There are four possible states for each disc, although some combinations are not allowed (e.g. both discs down):

UP	Available for read and/or write
DOWN	Failure found while in use
STANDBY	Available for write
OFF-LINE	Not connected (ignore)

Normal operation is with both discs in the up state. If a catastrophic error occurs on either disc when it is called by the driver manager, the manager declares the disc down and continues using the other disc. Then it schedules a monitor program. It passes to the monitor the disc state table and the error code returned by the disc driver. The system and other programs, however, see no change except that it now takes less time to write to the disc.

The writes are done in sequence to each disc using the same DMA channel. This naturally leads to a concern about the speed of I/O to the disc — each write takes twice as long as to a single disc. However, in most systems, 75% to 85% of all disc accesses are reads and reads are not done twice.

A slightly modified DVR32 is required for use by the driver manager. The modified driver is supplied as part of the supported driver manager package. The changes were kept to a minimum so that it would be easy to modify new versions of DVR32 as they were released. The changes permit reconfiguration to the select code each time DVR32 is enterred and also ensures that DVR32 will always act as a closed subroutine (i.e. so that it doesn't jump into the system unknown to the driver manager).

The standby state permits initial installation of the second disc or re-installation of a repaired disc with no system down time. To do this, a utility program makes a series of special read requests (an EXEC read which uses a subfunction code) to the driver manager. Each request causes the driver manager to do an internal copy from the up disc to the standby disc, using whatever buffer is provided by the program. For example, if a track buffer is used, then the disc is copied by making the special read request to each track on the disc. When the copy is complete, the program issues a request to the driver manager to declare the disc up.

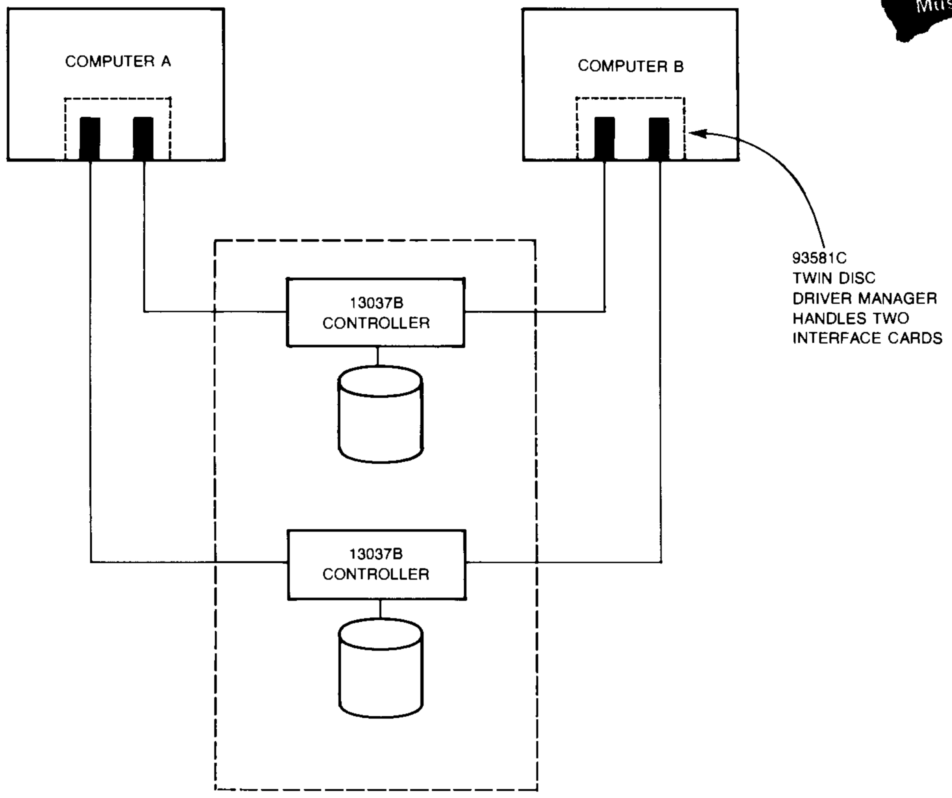


Figure 4. Twin Discs Operate as Single High Availability Disc

CPU SWITCHOVER

The switchover from a failed active CPU to the standby CPU can be easy or difficult, according to the demands of the application. The simplest switchover procedure is completely manual: the operator observes a failure, manually switches the devices to the new CPU and restarts the application programs by enterring commands on the system console. It is possible, however, to achieve more automatic operation with a "watchdog timer". This is described later in this article.

At the other end of the spectrum in complexity is the application which continues to run in the new CPU from the point of failure. Except for a delay needed to accomplish the switchover, the failure has not hindered the application. This type of system operation is usually possible only through specialized design which takes advantage of the application to simplify the requirements.

An example of a specialized design is a system in which the inputs consist of temperature measurements at a slow sampling rate. The input devices (e.g. thermocouples) are duplicated for each CPU and each CPU takes the data in parallel (synchronization of sampling times is not required). Only one CPU creates outputs, however, and this is called the active CPU.

The other CPU is a standby. The standby doesn't do anything useful for the application — it just waits for the active CPU to fail. If the active CPU does fail, then the standby takes over and becomes active. If this happens, then failed CPU must be fixed as quickly as possible for another CPU failure would down the system.

There is usually an upper limit to the switching time which can be tolerated and this may influence system design. Generally, the fastest switching time that can be achieved with HP 1000 hardware is on the order of several hundred milliseconds. The practical switching time is more often limited by software, however, and may range from several seconds to a minute or more. This software overhead results from the need to determine what the active CPU was doing at the time of failure so that the standby can "pick up the trail".

The switchover time may be broken down into several parts, as shown in figure 5. Switchover begins with a failure but is not complete until the standby CPU regains complete control of the application.

It is often not easy to decide what kind of failure should cause a switchover. There are some easy decisions, of course, such an entire CPU failure (loss of power?). But there may be many other situations which can cause the application programs to malfunction while other programs in the same CPU operate perfectly; e.g., a failure of an interface card. Since the use of the 93550A provides for duplicate interfaces (one per CPU), a switchover is beneficial in this case.

A more subtle problem may be a "gradually deteriorating" CPU. For example, there may be several memory parity errors occuring. The RTE-IV system will attempt to recover by downing partitions in which the parity errors occur. Thus the programs continue to run. But, eventually, there may be a serious degradation of performance and (possibly) an irrecoverable CPU error. Should you wait for final failure or switchover while the application is still "healthy"? Also, performance slowdown may occur for other reasons which are temporary (everyone on the system at once). Since there may be a penalty for CPU switchover (delay, perhaps a loss of data, etc), we might want to make certain that operation will benefit from a switchover.

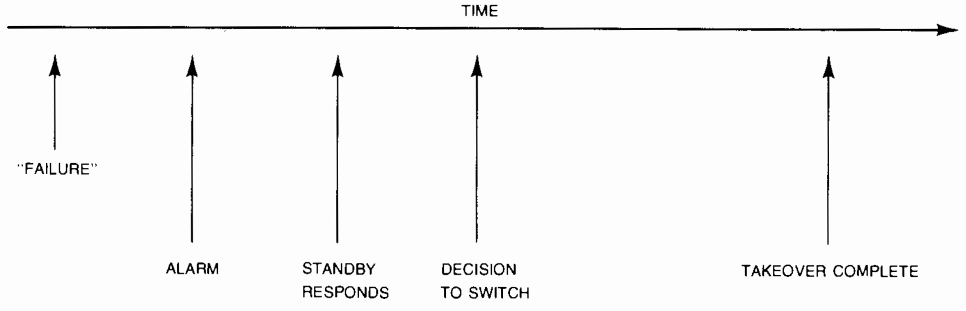


Figure 5. CPU Switchover Time has Several Components

WATCHDOG TIMER COMMUNICATES FAILURE

The 93768A Watchdog Timer is an I/O card that may be programmed either to create an alarm and/or detect an alarm. A watchdog in one CPU may be cross-coupled to a watchdog in the other CPU to permit either CPU to detect a failure of the other.

The card has many possible modes of operation but one major application is an RTE driver (DVS36) that has been written for redundant CPU combinations. This driver assumes a cable which is available from HP. Reference figure 6 with the following discussion of how the watchdog is used.

A data register (not shown) loads the counter upon receipt of a reset pulse from the remote CPU. The counter is decremented by one each time it gets a tick from the local clock. If the counter reaches zero the relay opens (also opens if power is lost) sending an alarm to the remote watchdog. When watchdog detects the alarm, an "emergency handler" program is scheduled and the switchover sequence begins.

Each CPU has a program in the time list which calls the watchdog driver to output a pulse. The pulse resets the counter in the remote watchdog to the value contained in the remote's data register, preventing it from reaching zero.

If the pulse program fails to run, the alarm will occur. However, the pulse program will do some sort of status checking and, if it decides that the CPU is in "bad health", it may deliberately avoid sending the pulse.

Notice that both watchdogs are in series and a failure of either can result in either no signal or no signal detection. The link is effectively in series with the active CPU since the standby CPU cannot distinguish between a CPU failure and a link failure. However, the MTBF of the link created by two watchdogs in series is at least an order of magnitude greater than of the CPU itself.

It would be helpful, of course, if we could distinguish between a link failure and a CPU failure. If only the link has failed, there is no purpose in switching CPUs. (We would like to know about the failure, however.) Not only are we no better off after the switchover but we have incurred all the problems of switching - including a delay and (possibly) a loss of data. Therefore, we might wish to make the link even more reliable by duplicating it, i.e. providing a pair of watchdogs in each CPU. Another method would be to use some sort of file communication on the multi-ported discs in conjunction with the single watchdog link. If link failed, then an attempt would be made to verify if the CPU were still up via file communication.

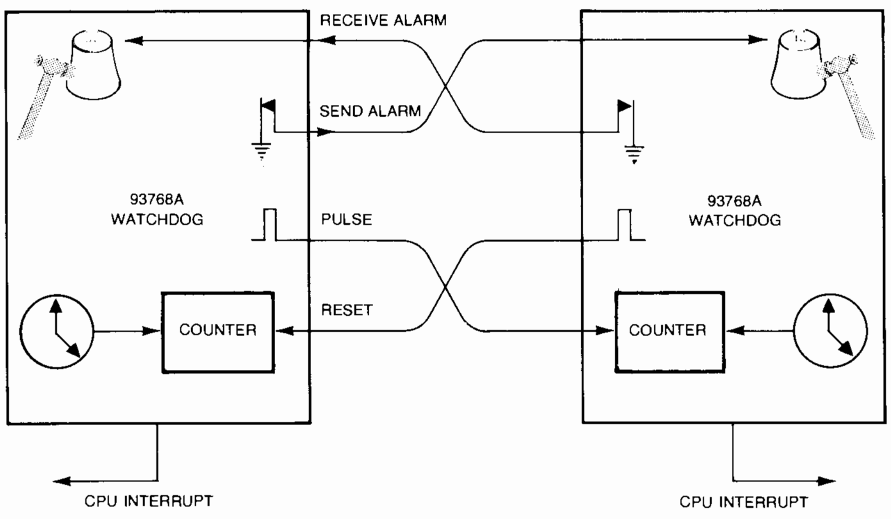


Figure 6. A Pair of Watchdog Timers Alert either CPU to a Failure of the other CPU

MORE COMPLEX CPU COMMUNICATION LINK

The watchdog timer provides a go no-go type of communication link. Often this is all that is required. Neither computer knows anything of what the other is doing unless one computer fails.

If the systems are to share the load in some manner, then more extensive CPU communication may be needed. For example, suppose half of the input devices are on one CPU and half on the other. If one CPU fails, its devices are switched to the surviving CPU, which then carries the total load.

There may be an output device (perhaps a printer) which is not duplicated. Therefore, if the CPU which is not attached to the printer wants to print something, it must route the message through the other CPU or else gain control of the printer. The CPU would not normally gain control without some protocol so the latter approach also implies a communications link.

A DS/1000 link be desirable for this type of communication because it is reliable and easy to use. But more importantly, the buffers transferred between CPUs are relatively small in this instance. If we were transferring very large buffers, we would be more interested in the potential speed of the hardware line.

For example, suppose we have a fixed overhead of 4 milliseconds to transfer data across the link. Consider two links with speeds of 30 KHz and 300 KHz (assume words, not bytes). If the buffer size is 300, then the total transfer times are:

4 + (300/30) = 4 + 10 = 14 ms for the 30 KHz link

4 + (300/300) = 4 + 1 = 5 ms for the 300 KHz link

If the buffers are output lines to a device, the size will more likely be 30 rather than 300. In this case, the times are:

4 + (30/30) = 4 + 1 = 5.0 ms for the 30 KHz link

4 + (30/300) = 4 + .1 = 4.1 ms for the 300 KHz link

If you determine that a high speed link is more suited to your application, then you may be interested in the 93762A Processor Interconnect Kit. With its RTE driver, this link is specified at 300 KHz words.

"PICKING UP THE TRAIL"

Once the standby CPU has decided to take control, the simplest part of switchover is gaining access of the peripherals. If the peripherals are on a 93550A Switch Unit, this is accomplished by a call to the switch driver.

After this point, the standby CPU might simply reset all the peripherals and start out as if nothing had happened before switchover — a "cold restart". If this is all that is desired, then the application software does not need to know of the twin system environment. Any existing software package can be transferred to the twin system without modification.

If, however, we would like the standby to continue handling the situation from the point of failure, then we must either have a special system designed for this purpose or we must write the programs to leave a "trail" of checkpoints which the standby CPU can pick up.

The programs in the standby CPU are assumed to be identical copies of the programs previously running in the active CPU. Thus, each time the application starts, whether in the active CPU or in the standby CPU, it must examine the checkpoints to determine where in sequence it must start.

The trail of checkpoints is most easily left on the disc, since the multi-ported drives are accessible to both CPUs. The checkpoints could conceivably could be left in the memory of the standby. For example, as each step is taken in the active CPU, it might be logged in a memory table in the standby CPU. Memory logging would probably not be any faster than disc logging due to the overhead in the communication link. It might, in fact, be slower but it could save wear and tear on the disc drive. Disc logging could reduce the MTBF of the drives by accelerating the rate of mechanical failures.

As an example, let's consider the requirements of a security/access system. The example has various badge readers spread around the entry/exits of the building and solenoids and sensors to control the doors. The sensors can report whether the solenoid locks are open or closed and whether the door itself is open or closed.

The badge readers are assumed to have a "ready" lamp. (No specific model badge reader is intended to be implied.) The lamp goes off while the transaction is in progress. When the transaction is in progress, the lamp is off. A buzzer sounds to indicate that the lock is open and stops when the door is locked again.

The unlock/lock mechanism and buzzer are under control of a device at the station The signal is given only to unlock. The local mechanism begins a timeout and sounds a buzzer. At the end of the timeout, the buzzer stops and the lock closes automatically.

Given this situation, we can list the separate steps that the program may perform. The steps are shown below. The arrow (>) indicates a checkpoint where information is posted. The "post" operation refers to the logging of the information in a table (on disc or in the standby's memory). There is a separate section of the table for each station.

1. Read the badge.

> 2. Post the badge ID.

3. Turn off the "ready" lamp.

> 4. Examine data base to see if ID valid. Assuming it is valid, post a "valid ID" flag.

5. Unlock the door.

6. Allow a program timeout greater than the lock cycle. At the end of the cycle, verify that the lock is closed and possibly that the door itself is closed.

> 7. Transaction complete. Post a dummy ID to indicate no transaction in progress.

8. Turn on the "ready" lamp at the station.

We can test whether this procedure works by examining the job of standby after CPU switchover. The first step would be to see if the individual stations are active via the ID posted. If the ID is a dummy, then the station may be simply reset to the "ready" state. If a failure occurred between steps 1 and 2 above then the person will have to input his badge a second time.

If a "valid ID" flag has been posted, then the standby still does not know whether the door was actually given an unlock command. It could read the door open sensor or lock sensor but if the sensors indicated both were closed, it could mean that the operation completed before the standby looked at the station. One option might be to issue the unlock command anyway, possibly repeating the operation just completed.

This example is presented only to show the possibilities. Any application with a repetitive transaction at each station may be analyzed in this manner to show the particular audit trail that is required. The analysis may also suggest a station design which makes the software job easier. If the station can always tell you what state it is in, then an audit trail is not needed. However, this requirement will generally increase the station cost and decrease its MTBF. This can lead to proposals to duplicate stations and even greater expense.

OPERATIONS MANAGEMENT

SUMMARY

The design of high availability systems can follow two approaches. One approach is to incorporate only those components with high reliability. This approach would simply eliminate as a possibility any component with an MTBF below a certain cutoff level.

When the choice of components is limited to only a few that can perform the desired function, then the second approach is to increase the availability by incorporating components in parallel combinations. This provides multiple paths for success and increases the overall availability. This approach attempts to eliminate overall failure due to a failure of a single component.

Inevitably, the system will have "weak points" where it is impossible or impractical to duplicate components. These weak points are almost always where the system receives its inputs and creates its outputs. This makes it difficult to design any high availability system for general purpose application.

CPU switchover is the most difficult aspect of systems with redundant components. Operating systems and programs will almost always have tables or information that will be lost (at least in part) when switchover occurs. The simplest approach is to break the application into a series of steps which can be posted to the disc. Thus, the failed CPU leaves a "trail" telling the standby CPU where to continue operation.

CPU-CPU communication can be complex if load-sharing is required. However, if the application can permit one CPU to be strictly in a standby mode, then the watchdog timer provides a simple method of communicating CPU failure and initiating switchover.

The twin disc driver provides the designer with an opportunity to avoid CPU switchover due to a disc failure and, at the same time, provide automatic back-up of data. It also provides the advantage of permitting re-installation of into the system without interrupting operation.

JOIN AN HP 1000 USER GROUP!

Here are the groups that we know of as of October 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
Boston	LEXUS P.O. Box 1000 Norwood, Mass. 02062
Chicago	Jim McCarthy Travenol Labs 1 Baxter Parkway Mailstop 1S-NK-A Deerfield, Illinois 60015
Greenville/N. C.	Henry Lucius American Hoerschst Greer, South Carolina (803) 877-8741
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
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

NORTH AMERICAN HP 1000 USER GROUPS (CONTINUED)

Area	User Group Contact
Toronto	Nancy Swartz Grant Hallman Associates 43 Eglinton Av. East Suite 902 Toronto M4P1A2
Washington/Baltimore	Paul Taltavull 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	E. van Ocken University of Antwerp (RUCA) Groenenborgerlaan 171 2020 Antwerp Belgium
France	Jean-Louis Rigot Technocatome TA/DE/SET Cadarache BP.1 13115 Saint Paul les Durance France
Germany	Hermann Keil Vorwerk+Co Elektrowerke Abt. TQPS Rauental 38-40 D-5600 Wuppertal Germany
The Netherlands	Mr. Van Putten Institute of Public Health Anthony Van Leeuwenhoeklaan 9 Postbus 1 3720 BA Bilthoven The Netherlands
South Africa	Andrew Penny Hewlett-Packard South Africa Pty. private bag Wendywood Sandton, 2144 South Africa
United Kingdom	Dave Thombs (Vice-Chairman) MQAD, Royal Arsenal East WOOLICH, London SE18 England

ANOTHER NEW USERS GROUP!

By John Gwyther/HP Melbourne

The number of European users groups continues to increase. The latest one was formed in Australia, has at least 40 members, and has been going strong since July, 1980. For those of you who are interested the address is:

Norm Kay
C.S.I.R.O.
Division of Protein Chemistry
Bayview Avenue
Clayton, Victoria 3168
Australia

FORTRAN 4X NOW AVAILABLE!

By Jim Williams/HP Data Systems Division

Our new, enhanced FORTRAN compiler, called FORTRAN 4X, product number 92834A, is now available through the normal ordering process. All ordering information, including option specifications, appears in the sales training brochure, on the CPL, and in the data sheet.

All customers who purchase the RTE-IVB operating system will continue to receive the RTE FORTRAN IV compiler. An accompanying order must be placed to receive the greatly enhanced FORTRAN 4X compiler at a cost of $1000.00.

Until the 2101 PCO revision to the software, the runtime libraries required by FORTRAN 4X will be shipped with the compiler. Configuration and installation instructions may be found on the production software in the first file on the media option, called "FTN4X."

FORTRAN 4X and RTE FORTRAN IV are compatible software, with a few exceptions delineated in the FORTRAN 4X reference manual, part number 92834-90001.

Printed in U.S.A. 11/80 **Part No. 5951-6111**