Worldwide Response Center

HP 3000 APPLICATION NOTE #43

RUN TIME ABORTS

HEWLETT PACKARD

July 15, 1988
Document P/N 5958-5824R2832A

RESPONSE CENTER APPLICATION NOTES

HP 3000 APPLICATION NOTES *are published by the Worldwide Response Center twice a month and are distributed with the Software Status Bulletin. These notes address topics where the volume of calls received at the Center indicates a need for addition to or consolidation of information available through HP support services.*

Following this publication you will find a list of previously published notes and a Reader Comment Sheet. You may use the Reader Comment Sheet to comment on the note, suggest improvements or future topics, or to order back issues. We encourage you to return this form; we'd like to hear from you.

Run Time Aborts

1. INTRODUCTION

This note is intended for programmers who want the ability to trace, through the abort address, the location of an error in their program.

A run time abort is MPE's mechanism to handle an irrecoverable situation encountered when a process is executing and there is no user trap facility in place.

In this note, an examination of some of the causes of these run time aborts will be made. The steps that MPE executes once the decision to abort has been made, will be explained. The components of an abort and the messages will be explained in order to use that information, along with a compiler listing and PMAP, to trace the address to the source of the abort. Also included, are two examples of abort situations with detailed instructions on locating the source statement causing the abort.

2. WHAT CAUSES A RUN TIME ABORT?

An explanation of some of the causes of these aborts and why the decision to terminate an executing process was made, will help with the error detection process.

A. LIBRARY ROUTINE (SUBSYSTEM'S LIBRARIES):

A process might be aborted because a subsystem library routine has encountered a problem. Suppose a READ is done from a FORTRAN program. A FORTRAN library routine is called to do the I/O. If an unexpected EOF is detected or a data format problem occurs and the library routine cannot complete the operation, it may have to abort the program.

Library routines called by COBOL, SORT, RPG, BASIC (compiled), etc. may all encounter similar situations. Many times it is possible to programmatically tell the library routine, in advance, what to do if a particular error occurs. Programmers can even write their own customize error recovery routines called user trap routines.

B. MPE INTRINSICS:

An MPE intrinsic may abort a process - the cause depends very much on each individual intrinsics' requirements. The abort could be caused by a missing parameter, a bad parameter value or address being passed, or a parameter that is the wrong data type. Another possibility is that, in order to use the intrinsic, the program file must have some special capability (i.e., DS, MR, PH) but has not been prepped with these capabilities. In any case, the intrinsic decides the situation requires special attention and it asks MPE to abort the program.

C. HARDWARE/MPE:

For the protection of other users and the system as a whole, the HP3000 routinely checks for errors. These errors detected by the hardware or MPE could cause an abort. For instance, if a data value maximum is exceeded during an arithmetic operation, the hardware detects the problem. An example would be attempting to add +1 to the integer value 32767 which would result in an INTEGER OVERFLOW. The hardware might also encounter a bad instruction which would result in an INVALID INSTRUCTION error or an invalid address for code which would result in a CST VIOLATION.

MPE will abort a program when a stack requires more space than the programmer has specified as necessary or when a stack requires more that the maximum stack space allowed by MPE (32K). This results in a STACK OVERFLOW. This could be caused by the data stack actually being too large or possibly a recursive procedure call or looping situation. Each time the procedure is called, the data used by that procedure is placed on top of the stack. In an unchecked recursive call, data would continually be added to the stack until a STACK OVERFLOW occurred.

Another cause for aborts is the improper indexing of arrays or the destruction of pointers by other programming errors. If the index or pointer references an area that does not lie within the bounds of the stack, a BOUNDS VIOLATION will occur and MPE will abort the program.

This, of course, is not intended to be a complete list, but just some of the more common causes of abort situations the programmer may encounter.

3. THE STEPS IN MPE'S ABORT MECHANISM

If there is no user trap in effect and the system has made the decision to abort the process, MPE executes the following steps:

A. The MPE error routines will print the abort address(es) and error messages.

B. The process resources are given back as in a normal program termination. All files are closed. For new files, the data may be lost. Extra data segments (if private) are deleted. All RINS are unlocked and the data stack is deleted. The code segments are then unloaded.

C. The Command Interpreter prints the final line - the abnormal termination message.

4. THE COMPONENTS OF AN ABORT MESSAGE

This is the standard format of an abort message:

```
                    #1             #2      #3      #4
          _______________   ___   ____   _____
          [              ] [    ] [   ] [       ]
ABORT   :PRMAST.PUB.PAYROLL.%0.%65:SYSL.%214.%1153

PROGRAM ERROR #20 :STACK OVERFLOW
[                               ]
              v
             #5
 _______________________________________________
[                                                ]
PROGRAM TERMINATED IN AN ERROR STATE.  (CIERR 976)
```

#1. This is the name, group, and account of the program that has aborted.

#2. The first set of octal numbers is the abort address within the user code. The first of these numbers is the user code segment where the abort occurred, code segment 0. The second octal number is the code offset into this segment, the location of the instruction within the segment, %65.

#3. This information and the following set of octal addresses will only appear if the abort occured while executing SL code. SYSL indicates that the user code was calling a routine in SL.PUB.SYS when the abort occurred. This information also might have been PUSL indicating an address location in the SL of the local PUB group, or GUSL indicating an address in the SL of the local group other than PUB.

#4. This next set of octal numbers are the code segment number and the code offset location that was being executed within the SL.

#5. These message lines are the error messages giving the program error number and the cause of the abort, and CI's abnormal termination message.

5. WHAT IS NEEDED TO TRACE THE ABORT?

A. COMPILER LISTING

The following listings are needed to trace the abort address.

The programmer will need a compiler source listing that includes the code offsets and/or a symbolic table map. The different languages use options specified on the $CONTROL line of the source file:

```
COBOLII          $CONTROL MAP, VERBS
FORTRAN77        $CONTROL CODE_OFFSETS
PASCAL           $CONTROL TABLES, CODE-OFFSETS
SPL              $CONTROL MAP, ADR, INNERLIST
BBASIC           (LINE) 0001 GLOBAL COPTION ID, LABEL
BASIC/3000       $CONTROL MAP
RPG              $CONTROL MAP, CODE
```

Depending upon the compiler used, these listings will have some differences. In some cases, the compiler listing will have two columns of numbers on the left side of the source statements. One column is the sequence numbers or editor line numbers. The other column is the starting location of the machine instruction code for each source statement. In other cases, the code offsets will be listed below all of the source statements, listing the editor statement number next to its corresponding starting code location. See the compiler listing examples in appendix A and B of this note.

Note that the starting code locations are not consecutive locations because one high-level language statement can cause the compiler to issue many machine level instructions.

B. PMAP

Next, the programmer will need a Pmap. To get a Pmap listing, do the following:

```
:FILE SEGLIST;DEV=LP          (or the LDEV # of a printer)
:PREP FTNUSL,FTNP;PMAP
```

Refer to the Fortran example of the Pmap in appendix A of this note for the following explanation.

The Pmap for the Fortran program, FTNP, contains two code segments:

```
PROCESSDATA -- segment 0;
MAINPROG    -- segment 1.
```

The routines called from each of the code segments are listed below the segment name. The first routine that is listed within PROCESSDATA is PROCESSRTN. This routine is assigned STT 1 (Segment Transfer Table). This routine begins at word 0 of the code segment and, therefore, has a value of 0 under the "CODE" heading. The "ENTRY" point in the code segment is located at word 24. Each word of code for PROCESSRTN then follows until the last word of code is reached. This is a routine whose code is internal to this code segment.

The following routines beginning with FTN__ are Fortran routines that do the error trapping, range checking, and I/O. These are external routines residing in an SL. Notice that there are ?'s under the heading "SEG" for all of the external routines. These will be resolved at LOAD time when the :RUN command is executed. These external routines are CALLED from this code segment but the actual code resides elsewhere. The second internal routine that is listed is SUM, STT 2, beginning at word 743 in this code segment. The internal routine, ZERO, STT 3, begins at word 1062.

In code segment 1, MAINPROG, are three types of routines. The first routine that is listed is MAIN___ which is the main program outer block. This is internal to this code segment. The second routine (and most of the others) are external routines whose code resides in an SL. The third routine, PROCESSRTN, is external to this code segment, but does reside with this program file. Its segment number is already resolved as residing in code segment 0, PROCESSDATA.

With these listings the abort address can now be traced.

6. HOW TO TRACE THE ABORT ADDRESS.

A. FORTRAN77 EXAMPLE

Please refer to the Fortran compiler listing and PMAP in Appendix A of this note for this tracing example.

```
ABORT   :FTNP.PUB.FTNACCT.%0.%1030
PROGRAM ERROR #24 :BOUNDS VIOLATION
```

The first octal number in this abort example (%0) is the program-relative number of the code segment in which the abort occurred. The second octal number (%1030) is the address of the instruction that was executing when the abort occurred. This instruction could not successfully complete. Therefore, identifying this instruction will give an important clue to the cause of the abort.

Another clue to keep in mind is the TYPE of error. Because this particular abort is a bounds violation, look for an operation that attempted to load or store outside the bounds of the data stack.

1) Locate the Segment.

Looking at the Pmap for FTNP.PUB.FTNACCT, locate the code segment which has the same relative segment number that appears in the abort message. In this example, code segment 0 is PROCESSDATA.

2) Locate the Routine/Procedure.

Next, determine which routine in PROCESSDATA was executing when the abort occurred. In the Pmap for this example, locate the second octal number (%1030). Begin by looking down the "CODE" column to find a code location that is less than the abort location, but the next location in this column is greater than the abort location. The addresses shown on the Pmap and in the abort message, are "absolute" addresses which are code locations relative to the start of the code segment. What is needed, however, is a code location relative to the start of one of the routines in the segment.

In this example, the abort occurred somewhere in the SUM routine. Code for this routine starts at location %743 which is less than the abort address, %1030, and the next code location is %1062 which is greater than %1030.

At this point the programmer has the general location of the problem. In many cases this is sufficient. If not, continue the trace to the specific code location.

Now convert the abort location to a routine-relative location. Do this by subtracting the starting code location for SUM from the abort code location. The result is the abort location relative to the start of the routine.

```
%1030
% 743     (This is OCTAL subtraction.)
-----
%  65
```

The abort occurred while executing the 65th code instruction in the routine SUM.

3) Locate the Instruction.

To locate the instruction, refer next, to the compiler listing. Just how to locate the source statement that corresponds to the relative location that we have calculated, depends on the compiler used. Each may provide different information concerning code locations.

In this example, the program is a Fortran 77 program. The code locations on this listing were obtained by compiling with $CONTROL CODE_OFFSETS. These code offsets are listed after all the source statements. The numbers under the "STMT" heading on the CODE_OFFSETS listing correspond to the statement numbers on the left side of the compiler source listing. Each statement number on the CODE_OFFSETS listing has a program code location (P-LOC) value associated with it. Looking at these P-LOC values, find a value that has a starting code location less than the calculated relative abort location, %65, with the next starting code location greater than %65. The code location that qualifies is at location %46, at statement number 6. This is the statement that was executing when the abort occurred.

4) Determine the cause

By examining the source statement, the reason for the abort may be obvious. The program can be corrected and the trace was successful.

Many times it is not so obvious, so a few items need to be considered:

a. The type of abort that occurred;

b. What typically causes this type of abort;

c. What the code is actually doing when the source statement is executed.

In this example, the type of abort is a bounds violation. The most likely cause is a subscript going out of bounds. This possibility should be checked first. This source statement would result in code that loads a subscripted array element to the top of stack (TOS) and adds it to a simple variable already loaded on TOS; the result is stored back into the simple variable.

The subscript, I, happens to be the loop index. The bounds of the loop determines the values that will be used to subscript the array. The limit and step for the loop are actually passed in the parameters to the routine, SUM. It is very possible that the limit is too high and is out of the true bounds of the array (and our stack as well).

Now find where SUM is called from PROCESSDATA. It is being called from two locations. Examine the parameters that are being passed. What determines these parameter values and what are the values? If this is not clear,a PRINT or WRITE statement could be added to show what these values are, before each call to SUM. A debugger could also be used to verify the values.

In this case, both calls to SUM are being passed the array, DATA. So where could the bounds of DATA be exceeded? DATA is dimensioned (12,NYRS). The LIMIT parameter will be either of these dimensioned, 12 or NYRS. In the line 16.000, SUM is called with the second parameter set to 122 instead of 12. This is the cause of the problem.

B. COBOLII EXAMPLE

Please refer to the COBOLII compiler listing and Pmap in Appendix B of this note for this tracing example. The Fortran 77 example provides more detailed information for tracing, so both examples should be read to have a good understanding of this process.

```
ABORT  :COBP.PUB.COBACCT.%2.%154:SYSL.%43.%3476
PROGRAM ERROR #24  :BOUNDS VIOLATION
```

In this COBOLII example, the abort actually occurred while executing a routine in SL.PUB.SYS. However, the programmer should start with the abort location in the program code where the SL routine was called. This is most likely a Library routine or intrinsic call that was caused to abort by an error in the program code.

1) Locate the Segment

Again use the Pmap to identify the segment reported by the abort address. The abort occurred in code segment %2. Looking at this example's Pmap, segment 2 is 100PROCESSDA02 which can also be identified in the source compiler listing as 100-PROCESS-DATA SECTION 02 in the main program.

2) Locate the Procedure

To determine which procedure was executing when the abort occurred, find a code location in the Pmap under the "CODE" heading that is less than the abort code location, %154, but the next location in this column is greater than %154. In this code segment, there is only one procedure that is internal to this segment, 100PROCESSDA02. (The segment name and the procedure have the same name.) The procedure COBEXSUB is called from this segment but resides in code segment 0 as shown by a 0 under the heading "SEG". All of the other routines are COBOLII Library routines residing in the SL. Therefore, the abort occurred while executing the %154 instruction in the the procedure 100PROCESSDA02. If there had been another internal procedure in this segment with a starting code location greater than the abort location, then the routine-relative location would need to be calculated as in the Fortran 77 example.

3) Locate the Instruction

From the Pmap, go to the PROCEDURE/VERB MAP of the main program where 100-PROCESS-DATA is located.

Note that each program and subprogram has its own Symbol Table Map and Procedure/Verb Map. Unlike the Pmap which is combined, these are maps of each individual program that is compiled with $CONTROL MAP, VERBS.

This listing shows each procedure in this program and its relative PB (program base) location. Remember that the second half of the program's abort address is the code offset into the segment where the error occurred. This location, %154 is the %154 code instruction in the procedure, 100-PROCESS-DATA. Find this location by looking down the column labeled PB-LOC. The value needed is a PB-LOC value that is greater than the abort location, but less than the following PB-LOC value.

In this example, the largest PB-LOC is %131. Because there is no other PB-LOCation in this procedure, the verb located at %131 was the last to execute. The error occurred trying to execute this last DISPLAY statement.

4) Determine the Cause

Next, look at the source listing and locate this source DISPLAY statement to determine why the abort occurred. This statement is

```
DISPLAY TAB-PLAYER-NAME(PLX), TAB-PLAYER-NUM(PLY),
        TAB-BATTING-AVER(PLX).
```

The usual cause for a bounds violation is the improper indexing of an array. This array, TAB-PLAYER-RECORD, is indexed by PLX; however, the DISPLAY of one element in this array is subscripted by PLY. In Working-Storage, PLY has a value of 1000 which is meant to be the array index limit. This is in error because the array is defined as occurring 100 times.

The bounds error could have occurred while attempting to add elements to the array that exceeded its limit. Although the bounds violation would probably not have occurred while attempting to add the 101st entry, it would have occurred when the program tried to add an entry in a location that exceeded the data limit of the stack. Other data on the stack could have been overwritten before the bounds error occurred.

The array element, TAB-PLAYER-NUM, is mistakenly indexed by PLY which has a value of 1000. The COBOLII display routine in the SL attempted to display a location beyond the bounds of the stack and the abort occurred.

7. SUMMARY

The examples that were used are very simple programs, but the steps for tracing a run time abort are exactly the same for a large 10,000 line program with many subprograms, as well as, a small 100 line program. Take the steps one at a time.

1) Run a compiler listing with code locations or a map.

2) Prep with the Pmap option.

3) From the abort address, locate the code segment number on the Pmap. 4) Determine what procedure or routine was being called or performed at the time of the abort.

5) Then use the code offsets on the compiler listing or the map to identify the statement or instruction that was being executed at the time of the abort.

6) Once the instruction has been located, determine the possible causes for the abort. Knowing the program logic, the data,and what results are expected, is very useful in determining the likely cause. Locating the abort location sometimes is not conclusive. The abort could be the result of other programming errors and, therefore would point to the location of the abort, but not to the error itself.

   For example, if a call was made to transfer data base information to a buffer and the buffer is too small to contain all of the information, then the following area of the data stack already containing valid data could be overwritten. If this overwritten area of the stack contains a stack marker, then a CST violation could occur. The program could not branch to a valid code location to continue executing. The abort address would point to the PCAL instruction that was made to an invalid location. It would not point to the transfer of data to the buffer which is the actual error.

```
1
  PAGE    1  HEWLETT-PACKARD   HP32116A.00.11
    HP FORTRAN 77  (C) HEWLETT-PACKARD CO. 1986   WED, APR  6, 1988,  9:07 AM


     0       1.000      $CONTROL SEGMENT 'MAINPROG'
     0       2.000      $CONTROL USLINIT, CODE_OFFSETS, RANGE
     1       3.000            WRITE(6,*)  "HOW MANY YEARS?"
     2       4.000            READ(5,*)  NYRS
     3       5.000            CALL PROCESSRTN(NYRS)
     4       6.000            STOP
     5       7.000            END
     0       8.000

                               C O D E   O F F S E T S

 STMT  P-LOC     STMT  P-LOC     STMT  P-LOC     STMT  P-LOC     STMT  P-LOC
    1 000003        2 000024        3 000035        4 000037        5 000042


    NUMBER OF ERRORS =      0   NUMBER OF WARNINGS =    0
    PROCESSOR TIME    0: 0: 1   ELAPSED TIME        0: 0: 9
    NUMBER OF LINES =       8
```

```
   0     1.000      $CONTROL SEGMENT 'PROCESSDATA'
   0     2.000      $CONTROL CODE_OFFSETS, RANGE
   1     3.000            SUBROUTINE PROCESSRTN(NYRS)
   2     4.000            REAL DATA in (12,NYRS), TOTAL(NYRS), AVG(12)
   2     5.000      C
   3     6.000            PRINT *,"enter data now"
   4     7.000            read  *,   DATA in
   5     8.000            WRITE(6,*) "after read"
   5     9.000      C
   6    10.000            PRINT *,"call to zero"
   7    11.000            CALL ZERO(TOTAL,NYRS)
   8    12.000            CALL ZERO(AVG,12)
   8    13.000      C
   9    14.000            PRINT *,"first call to sum"
  10    15.000            DO I = 1, NYRS
  11    16.000  1         TOTAL(I)=SUM(DATA in(1,I),122,1)
  12    17.000  1         END DO
  12    18.000  1   C
  13    19.000            PRINT *, "second call to sum"
  14    20.000            DO I=1,12
  15    21.000  1         AVG(I)=SUM(DATA in (I,1),NYRS,NYRS)/NYRS
  16    22.000  1         END DO
  16    23.000  1   C
  17    24.000            WRITE(6,600) "YEARLY TOTALS",(I,TOTAL(I),I=1,NYRS)
  17    25.000      C
  18    26.000            WRITE(6,600) "MONTHLY AVGS",(I,AVG(I),I=1,NYRS)
  19    27.000        600 FORMAT(1X,S/(I4,2X,F16.3))
  19    28.000      C
  20    29.000        700 RETURN
  21    30.000            END
```

C O D E   O F F S E T S

STMT	P-LOC	STMT	P-LOC	STMT	P-LOC	STMT	P-LOC	STMT	P-LOC
2	000026	2	000030	3	000074	4	000115	5	000133
6	000154	7	000175	8	000200	9	000206	10	000227
11	000237	12	000323	13	000332	14	000353	15	000355
16	000447	17	000457	18	000556	20	000660	21	000661

```
NUMBER OF ERRORS =      0   NUMBER OF WARNINGS =    0
PROCESSOR TIME    0: 0: 2   ELAPSED TIME       0: 0:12
NUMBER OF LINES =     30
```

```
1
 PAGE     1  HEWLETT-PACKARD   HP32116A.00.11
   HP FORTRAN 77  (C) HEWLETT-PACKARD CO. 1986   WED, APR  6, 1988,  9:11 AM

    0      1.000      $CONTROL SEGMENT 'PROCESSDATA'
    0      2.000      $CONTROL CODE_OFFSETS, RANGE
    1      3.000            SUBROUTINE ZERO(ARY, LIMIT)
    2      4.000            REAL ARY(LIMIT)
    3      5.000            DO 1 I = 1, LIMIT
    4      6.000  1       1 ARY(I) = 0.0
    5      7.000            RETURN
    6      8.000            END

                                 C O D E   O F F S E T S

 STMT  P-LOC     STMT  P-LOC     STMT  P-LOC     STMT  P-LOC     STMT  P-LOC
    2 000003        3 000005        4 000015        5 000044        6 000045


   NUMBER OF ERRORS =      0   NUMBER OF WARNINGS =     0
   PROCESSOR TIME   0: 0: 1   ELAPSED TIME      0: 0: 8
   NUMBER OF LINES =       8
```

```
1
  PAGE     1  HEWLETT-PACKARD   HP32116A.00.11
    HP FORTRAN 77  (C) HEWLETT-PACKARD CO. 1986   WED, APR  6, 1988,  9:12 AM


     0      1.000      $CONTROL SEGMENT 'PROCESSDATA'
     0      2.000      $CONTROL CODE_OFFSETS, RANGE
     1      3.000            FUNCTION SUM(ARY, LIMIT, STEP)
     2      4.000            REAL ARY(LIMIT)
     3      5.000            INTEGER STEP
     4      6.000            XSUM = 0.0
     5      7.000            DO I = 1, LIMIT, STEP
     6      8.000  1         XSUM = XSUM + ARY(I)
     7      9.000  1         SUM  = XSUM
     8     10.000  1         END DO
     9     11.000            RETURN
    10     12.000            END

                         C O D E   O F F S E T S

STMT   P-LOC     STMT   P-LOC     STMT   P-LOC     STMT   P-LOC     STMT   P-LOC
   2 000003         4 000005         5 000007         6 000046         7 000070
   8 000071         9 000112        10 000113


    NUMBER OF ERRORS =      0    NUMBER OF WARNINGS =     0
    PROCESSOR TIME    0: 0: 1    ELAPSED TIME       0: 0: 8
    NUMBER OF LINES =      12


PROGRAM FILE FTNP.PUB.FTNACCT

PROCESSDATA        0
   NAME             STT  CODE ENTRY SEG
   PROCESSRTN         1     0    24
   FTN_RANGE_ERR      4                ?
   FTN_E_RSLE         5                ?
   FTN_S_RSLE         6                ?
   FTN_E_WSLE         7                ?
   FTN_S_WSLE        10                ?
   FTN_DO_R4IO       11                ?
   FTN_DO_I4IO       12                ?
   FTN_DO_CHIO       13                ?
   FTN_E_WSFE        14                ?
   FTN_S_WSFE        15                ?
   FTN_DO_R4IOA      16                ?
   SUM                2   743   744
   FTN_LOOP_ERR      17                ?
   ZERO               3  1062  1063
   SEGMENT LENGTH        1150
MAINPROG           1
   NAME             STT  CODE ENTRY SEG
   MAIN__             1     0     1
   FTN_S_STOP         2                ?
```

```
    PROCESSRTN         3              0
    FTN_E_RSLE         4              ?
    FTN_S_RSLE         5              ?
    FTN_E_WSLE         6              ?
    FTN_S_WSLE         7              ?
    FTN_DO_I4IO       10              ?
    FTN_DO_CHIO       11              ?
    FTN_F_EXIT        12              ?
    TERMINATE'        13              ?
    SEGMENT LENGTH         74
 PRIMARY DB           0    INITIAL STACK   10240    CAPABILITY        600
 SECONDARY DB         0    INITIAL DL          0    TOTAL CODE       1244
 TOTAL DB             0    MAXIMUM DATA        ?    TOTAL RECORDS      12
 ELAPSED TIME    00:00:01.918               PROCESSOR TIME    00:00.711
1
```

```
PAGE 0001   HEWLETT-PACKARD 32233A.01.07  [74] COBOL II/V    FRI, APR  8, 1988,
                10:57 AM Copyright HEWLETT-PACKARD CO. 1987

00001 COBCNTL   001000*    COBCNTL.PUB.SYS  Defaults are:
00002 COBCNTL   002000*CONTROL LIST,SOURCE,NOCODE,NOCROSSREF,ERRORS=100,NOVERBS,
                WARN
00003 COBCNTL   003000*CONTROL LINES=60,NOMAP,MIXED,QUOTE=",NOSTDWARN,SYNC16
00004           001000$CONTROL USLINIT,MAP,VERBS
00006           001100
00007           001200 IDENTIFICATION DIVISION.
00008           001300
00009           001400 PROGRAM-ID.    COBOLEX.
00010           001500 AUTHOR.  NA RESPONSE CENTER.
00011           001600 DATE-WRITTEN.  MAR. 15, 1988.
00012           001700
00013           001800 ENVIRONMENT DIVISION.
00014           001900
00015           002000 CONFIGURATION SECTION.
00016           002100 SOURCE-COMPUTER.  HP-3000.
00017           002200 OBJECT-COMPUTER.  HP-3000.
00018           002300
00019           002400 SPECIAL-NAMES.
00020           002500     CONDITION-CODE IS C-C.
00021           002600
00022           002700 INPUT-OUTPUT SECTION.
00023           002800 FILE-CONTROL.
00024           002900    SELECT PLAYER-FILE            ASSIGN TO "PLAYER,DA".
00025           003000
00026           003100
00027           003200 DATA DIVISION.
00028           003300
00029           003400 FILE SECTION.
00030           003500
00031           003600 FD  PLAYER-FILE                  LABEL RECORDS ARE STANDA
                RD
00032           003700                                  RECORD CONTAINS  80 CHAR
                ACTERS.
00033           003800
00034           003900 01  FD-PLAYER-RECORD.
00035           004000     05  STAT-REC                 PIC X(80).
00036           004100
00037           004200 WORKING-STORAGE SECTION.
00038           004300
00039           004400 01  PLAYER-RECORD.
00040           004500     05  FILLER                   PIC X(2).
00041           004600     05  PLAYER-NUM               PIC 9(4).
00042           004700     05  AT-BATS                  PIC S9(4).
00043           004800     05  HITS                     PIC S9(4).
00044           004900     05  PLAYER-NAME              PIC X(62).
00045           005000     05  BATTING-AVER             PIC S9V999 VALUE ZERO.
00046           005100
```

```
005200 01  OUT-PLAYER-RECORD.
005300     05  FILLER                    PIC X(2).
005400     05  OUT-PLAYER-NUM            PIC 9(4).
005500     05  OUT-AT-BATS               PIC S9(4).
005600     05  OUT-HITS                  PIC S9(4).
005700     05  OUT-PLAYER-NAME           PIC X(62).
005800     05  OUT-BATTING-AVER          PIC S9V999.
005900
```

```
006000 01  TAB-PLAYER-RECORD.
006100     05  DISPLAY-TABLE OCCURS 100 TIMES INDEXED BY PLX.
006200         10  FILLER                 PIC X(2).
006300         10  TAB-PLAYER-NUM        PIC 9(4).
006400         10  TAB-AT-BATS           PIC S9(4).
006500         10  TAB-HITS              PIC S9(4).
006600         10  TAB-PLAYER-NAME       PIC X(62).
006700         10  TAB-BATTING-AVER      PIC S9V999.
006800
006900 01  PLY                            PIC S9(4) COMP VALUE 100
0.
007000
007100 01  END-OF-FILE-IND                PIC X VALUE "N".
007200     88  END-OF-FILE                      VALUE "Y".
007300
007400************************
007500*  BEGIN MAIN PROGRAM  *
007600************************
007700
007800 PROCEDURE DIVISION.
007900
008000 000-MAIN-ROUTINE SECTION 01.
008100
008200     MOVE "N" TO END-OF-FILE-IND.
008300
008400     OPEN INPUT PLAYER-FILE.
008500
008600
008700     SET PLX TO 1.
008800     PERFORM 100-PROCESS-DATA
008900         UNTIL END-OF-FILE.
009000
009100**************************
009200*    DO MORE PROCESSING  *
009300**************************
009400
009500     SET PLX TO 1.
009600     PERFORM 200-RPT-DATA UNTIL PLX = PLY.
009700     CLOSE PLAYER-FILE.
009800     STOP RUN.
009900
010000 100-PROCESS-DATA SECTION 02.
010100
010200     READ PLAYER-FILE RECORD INTO PLAYER-RECORD
010300                          AT END MOVE "Y" TO END-OF-FILE-IND.
010400
010500     CALL "COBEXSUB" USING HITS, AT-BATS, BATTING-AVER.
010600
010700     IF PLX < PLY THEN
010800         SET PLX UP BY 1.
010900     MOVE HITS TO TAB-HITS(PLX).
011000     MOVE AT-BATS TO TAB-AT-BATS(PLX).
011100     MOVE BATTING-AVER TO TAB-BATTING-AVER(PLX).
```

```
011200     MOVE PLAYER-NUM TO TAB-PLAYER-NUM(PLX).
011300     MOVE PLAYER-NAME TO TAB-PLAYER-NAME(PLX).
011400     DISPLAY  TAB-PLAYER-NAME(PLX), TAB-PLAYER-NUM(PLY),
011500              TAB-BATTING-AVER(PLX).
```

```
011600
011700
011800 200-RPT-DATA SECTION 03.
011900
012000     SET PLX UP BY 1.
012100     MOVE TAB-HITS(PLX) TO OUT-HITS.
012200     MOVE TAB-AT-BATS(PLX) TO OUT-AT-BATS
012300     MOVE TAB-BATTING-AVER(PLX) TO OUT-BATTING-AVER.
012400     MOVE TAB-PLAYER-NUM(PLX) TO OUT-PLAYER-NUM.
012500     MOVE TAB-PLAYER-NAME(PLX) TO OUT-PLAYER-NAME.
012600
012700     CALL INTRINSIC "PRINT" USING OUT-PLAYER-RECORD,
012800                                  -80,%0.
012900
013000
```

```
 LINE# LVL SOURCE NAME                         BASE DISPL   SIZE   USAGE     CATEGORY
                  R O J BZ

 FILE SECTION

 00000 FD  PLAYER-FILE                         Q+2: 000332  000106 SEQUENTIAL
 00034 01   FD-PLAYER-RECORD                   Q+2: 000444  000120 DISP       AN
 00035 05   STAT-REC                           Q+2: 000444  000120 DISP       AN

 WORKING-STORAGE SECTION

 00039 01  PLAYER-RECORD                       Q+2: 000564  000120 DISP       AN
 00040 05   FILLER                             Q+2: 000564  000002 DISP       AN
 00041 05   PLAYER-NUM                         Q+2: 000566  000004 DISP       N
 00042 05   AT-BATS                            Q+2: 000572  000004 DISP       NS
 00043 05   HITS                               Q+2: 000576  000004 DISP       NS
 00044 05   PLAYER-NAME                        Q+2: 000602  000076 DISP       AN
 00045 05   BATTING-AVER                       Q+2: 000700  000004 DISP       NS
 00047 01  OUT-PLAYER-RECORD                   Q+2: 000704  000120 DISP       AN
 00048 05   FILLER                             Q+2: 000704  000002 DISP       AN
 00049 05   OUT-PLAYER-NUM                     Q+2: 000706  000004 DISP       N
 00050 05   OUT-AT-BATS                        Q+2: 000712  000004 DISP       NS
 00051 05   OUT-HITS                           Q+2: 000716  000004 DISP       NS
 00052 05   OUT-PLAYER-NAME                    Q+2: 000722  000076 DISP       AN
 00053 05   OUT-BATTING-AVER                   Q+2: 001020  000004 DISP       NS
 00055 01  TAB-PLAYER-RECORD                   Q+2: 001024  017500 DISP       AN
 00056 05   DISPLAY-TABLE                      Q+2: 001024  000120 DISP       AN
                   O
            PLX                                Q+2: 000000  000002 INDEX NAME
 00057 10   FILLER                             Q+2: 001024  000002 DISP       AN
 00058 10   TAB-PLAYER-NUM                     Q+2: 001026  000004 DISP       N
 00059 10   TAB-AT-BATS                        Q+2: 001032  000004 DISP       NS
 00060 10   TAB-HITS                           Q+2: 001036  000004 DISP       NS
 00061 10   TAB-PLAYER-NAME                    Q+2: 001042  000076 DISP       AN
 00062 10   TAB-BATTING-AVER                   Q+2: 001140  000004 DISP       NS
 00064 01  PLY                                 Q+2: 020524  000002 COMP       NS
 00066 01  END-OF-FILE-IND                     Q+2: 020526  000001 DISP       AN
 00067 88   END-OF-FILE
```

```
 LINE# LVL SOURCE NAME                     BASE DISPL   SIZE   USAGE     CATEGORY
                R O J BZ

 STORAGE LAYOUT              (#ENTRYS)    (VALUES IN WORDS)

          INDEX TABLE           (1)      Q+1: 000000  000001
          START TABLE           (3)      Q+1: 000001  000006
          DISPLAY BUFFER                 Q+1: 000007  000144
          USER LABEL POINTER             Q+1: 000153  000002
          FILE TABLE            (1)      Q+1: 000155  000043
          TALLY                          Q+1: 000220  000002
          USER STORAGE                   Q+1: 000222  010032
          RUNNING PICTURES               Q+1: 010254  000003
          FIXUP AREA            (1)      Q+1: 010257  000011

 POINTER AREA

          DB-5   CURRENT VALUE OF Q FOR STORAGE AREA
          DB-4   'PARM=' WORD - SWITCHES
          Q+1    WORD ADDRESS OF STORAGE AREA
          Q+2    BYTE ADDRESS OF STORAGE AREA
          Q+3    DECIMAL POINT & COMMA
          Q+4    # PARMS AND CURRENCY SIGN
          Q+5    BYTE ADDRESS OF 9 WORD TEMPCELLS
          Q+6    WORD ADDRESS OF 1 WORD TEMPCELLS
          Q+7    BYTE ADDRESS OF LITERAL POOL
          Q+10   PLABEL OF SORT OR MERGE OUTPUT
          Q+11   WORD ADDRESS OF START TABLE
          Q+12   WORD ADDRESS OF USER LABEL POINTER
          Q+13   PREVIOUS VALUE OF DB-5
          Q+14   RESERVED
```

```
PAGE 0006/COBTEXT  COBOLEX         PROCEDURE/VERB MAP
 LINE #    PB-LOC   #    PROCEDURE NAME/VERB                       INTERNAL NAME

 00075     000003    0 000-MAIN-ROUTINE                           000MAINROUTI01'
 00077     000003                 MOVE
 00079     000006                 OPEN
 00082     000036                 SET
 00084     000040                 PERFORM
 00090     000052                 SET
 00091     000054                 PERFORM
 00092     000071                 CLOSE
 00093     000076                 STOP
 00095     000003      100-PROCESS-DATA                           100PROCESSDA02'
 00098     000003                 READ
 00098     000003                 MOVE
 00098     000025                 MOVE
 00100     000030                 CALL
 00102     000037                 IF
 00103     000046                 SET
 00104     000054                 MOVE
 00105     000065                 MOVE
 00106     000076                 MOVE
 00107     000107                 MOVE
 00108     000120                 MOVE
 00110     000131                 DISPLAY
 00113     000003      200-RPT-DATA                               200RPTDATA03'
 00115     000003                 SET
 00116     000010                 MOVE
 00117     000034                 MOVE
 00118     000045                 MOVE
 00119     000056                 MOVE
 00120     000067                 MOVE
 00123     000100                 CALL

0 ERRORS, 0 QUESTIONABLE, 0 WARNINGS

    DATA AREA IS %010270 WORDS.
    CPU TIME = 0:00:04.  WALL TIME = 0:00:09.
```

```
C
PAGE 0001   HEWLETT-PACKARD 32233A.01.07  [74] COBOL II/V    FRI, APR  8, 1988,
                  11:07 AM Copyright HEWLETT-PACKARD CO. 1987

COBCNTL   001000*    COBCNTL.PUB.SYS  Defaults are:
COBCNTL   002000*CONTROL LIST,SOURCE,NOCODE,NOCROSSREF,ERRORS=100,NOVERBS,
          WARN
COBCNTL   003000*CONTROL LINES=60,NOMAP,MIXED,QUOTE=",NOSTDWARN,SYNC16
          001000$CONTROL SUBPROGRAM, MAP, VERBS
          001100
          001200 IDENTIFICATION DIVISION.
          001300
          001400 PROGRAM-ID.    COBEXSUB.
          001500 AUTHOR.  NA RESPONSE CENTER.
          001600 DATE-WRITTEN.  MAR. 15, 1988.
          001700
          001800 ENVIRONMENT DIVISION.
          001900
          002000 DATA DIVISION.
          002100
          002200 WORKING-STORAGE SECTION.
          002300
          002400 LINKAGE SECTION.
          002500
          002600 01  HITS                          PIC S9(4).
          002700 01  AT-BATS                       PIC S9(4).
          002800 01  BATTING-AVER                  PIC S9V999.
          002900
          003000************************
          003100*  BEGIN SUB  PROGRAM  *
          003200************************
          003300
          003400 PROCEDURE DIVISION USING HITS, AT-BATS, BATTING-AVER.
          003500
          003600 000-SUB-ROUTINE.
          003700
          003800     MOVE 0 TO BATTING-AVER.
          003900     COMPUTE BATTING-AVER = HITS / AT-BATS.
          004000
          004100 GOBACK.
```

```
LINE# LVL SOURCE NAME                      BASE DISPL   SIZE    USAGE     CATEGORY
                R O J BZ


LINKAGE SECTION

00021 01   HITS                            Q+20 000000  000004 DISP       NS
00022 01   AT-BATS                         Q+21 000000  000004 DISP       NS
00023 01   BATTING-AVER                    Q+22 000000  000004 DISP       NS
```

```
LINE# LVL SOURCE NAME                        BASE DISPL   SIZE   USAGE      CATEGORY
                R O J BZ

STORAGE LAYOUT                 (#ENTRYS)     (VALUES IN WORDS)
          FIRST TIME FLAG                    Q+1: 000000  000001
          START TABLE            (1)         Q+1: 000001  000002
          USER LABEL POINTER                 Q+1: 000003  000002
          TALLY                              Q+1: 000005  000002
          RUNNING PICTURES                   Q+1: 000007  000003
          FIXUP AREA             (1)         Q+1: 000012  000011
          9 WORD TEMP CELLS      (3)         Q+1: 000023  000033

POINTER AREA

          DB-5   CURRENT VALUE OF Q FOR STORAGE AREA
          DB-4   'PARM=' WORD - SWITCHES
          Q+1    WORD ADDRESS OF STORAGE AREA
          Q+2    BYTE ADDRESS OF STORAGE AREA
          Q+3    DECIMAL POINT & COMMA
          Q+4    # PARMS AND CURRENCY SIGN
          Q+5    BYTE ADDRESS OF 9 WORD TEMPCELLS
          Q+6    WORD ADDRESS OF 1 WORD TEMPCELLS
          Q+7    BYTE ADDRESS OF LITERAL POOL
          Q+10   PLABEL OF SORT OR MERGE OUTPUT
          Q+11   WORD ADDRESS OF START TABLE
          Q+12   WORD ADDRESS OF USER LABEL POINTER
          Q+13   PREVIOUS VALUE OF DB-5
          Q+14   RESERVED
          Q+15   TO Q+17 WORD ADDRESSES FOR PARMs/EXTs
          Q+20   TO Q+22 BYTE ADDRESSES FOR PARMs/EXTs
```

BACK ISSUE INFORMATION

Following is a list of the Application Notes published to date. If you would like to order single copies of back issues please use the *Reader Comment Sheet* attached and indicate the number(s) of the note(s) you need.

Note #	*Published*	*Topic*
1	2/21/85	*Printer Configuration Guide (superseded by note #4)*
2	10/15/85	*Terminal types for HP 3000 HPIB Computers (superseded by note #13)*
3	4/01/86	*Plotter Configuration Guide*
4	4/15/86	*Printer Configuration Guide - Revised*
5	5/01/86	*MPE System Logfile Record Formats*
6	5/15/86	*Stack Operation*
7	6/01/86	*COBOL II/3000 Programs: Tracing Illegal Data*
8	6/15/86	*KSAM Topics: COBOL's Index I/O; File Data Integrity*
9	7/01/86	*Port Failures, Terminal Hangs, TERMDSM*
10	7/15/86	*Serial Printers - Configuration, Cabling, Muxes*
11	8/01/86	*System Configuration or System Table Related Errors*
12	8/15/86	*Pascal/3000 - Using Dynamic Variables*
13	9/01/86	*Terminal Types for HP 3000 HPIB Computers - Revised*
14	9/15/86	*Laser Printers - A Software and Hardware Overview*
15	10/01/86	*FORTRAN Language Considerations - A Guide to Common Problems*
16	10/15/86	*IMAGE: Updating to TurboIMAGE & Improving Data Base Loads*
17	11/01/86	*Optimizing VPLUS Utilization*
18	11/15/86	*The Case of the Suspect Track for 792X Disc Drives*
19	12/01/86	*Stack Overflows: Causes & Cures for COBOL II Programs*
20	1/01/87	*Output Spooling*
21	1/15/87	*COBOLII and MPE Intrinsics*
22	2/15/87	*Asynchronous Modems*
23	3/01/87	*VFC Files*
24	3/15/87	*Private Volumes*
25	4/01/87	*TurboIMAGE: Transaction Logging*
26	4/15/87	*HP 2680A, 2688A Error Trailers*
27	5/01/87	*HPTrend: An Installation and Problem Solving Guide*
28	5/15/87	*The Startup State Configurator*
29	6/01/87	*A Programmer's Guide to VPLUS/3000*
30	6/15/87	*Disc Cache*
31	7/01/87	*Calling the CREATEPROCESS Intrinsic*
32	7/15/87	*Configuring Terminal Buffers*
33	8/15/87	*Printer Configuration Guide*
34	9/01/87	*RIN Management (Using COBOLII Examples) (A)*
34	10/01/87	*Process Handling (Using COBOLII Examples) (B)*
35	10/15/87	*HPDESK IV (Script files, FSC, and Installation Considerations)*
34	11/01/87	*Extra Data Segments (Using COBOLII Examples) (C)*
36	12/01/87	*Tips for the DESK IV Administrators*
37	12/15/87	*AUTOINST: Trouble-free Updates*
38	1/01/88	*Store/Restore Errors*
39	1/15/88	*MRJE Emulates a HASP Workstation*
40	2/01/88	*HP 250 / 260 to HP 3000 Communications Guidelines*
41	4/01/88	*MPE File Label Revealed - Revised 6/15/88*
42	7/15/88	*System Interrupts*
43	7/15/88	*Run Time Aborts*

READER COMMENT SHEET

Worldwide Response Center Supports
HP 3000 Application Note 43: RUN TIME ABORTST
(July 15, 1988)

We welcome your evaluation of this Application Note. Your comments and suggestions help us to improve our publications. Please explain your answers under Comments, below, and use additional pages if necessary.

Is this Application Note technically accurate? ☐ **Yes** ☐ **No**

Are the concepts and wording easy to understand? ☐ **Yes** ☐ **No**

Is the format of this Application Note convenient in size, arrangement and readability? ☐ **Yes** ☐ **No**

Comments and/or suggestions for future Application Notes:

This form requires no postage stamp if mailed in the U.S. For locations outside the U.S., your local HP representative will ensure that your comments are forwarded.

FROM: **Date** __________

Name ______________________________

Company ______________________________

Address ______________________________

____

____

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1710, SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Application Note Comments
Hewlett-Packard Worldwide Response Center
3300 Scott Boulevard
Santa Clara, CA 95054

FOLD

FOLD