APSIM64/APDEBUG64 MANUAL

860-7489-001B

by FPS Technical Publications Staff

APSIM64/APDEBUG64 MANUAL

860-7489-001B

Publication No. 860-7489-001B
March, 1984

NOTICE

The information in this publication is subject to change without notice.

Floating Point Systems, Inc. accepts no liability for any loss, expense, or damage resulting from the use of any information appearing in this publication.

Printed in USA

CONTENTS

CHAPTER 5 (cont.)

CHAPTER 6 ACCESSING REGISTER AND MEMORY LOCATIONS

APPENDIX A FIELD MNEMONICS

ILLUSTRATIONS

TABLES

TABLES (cont.)

CHAPTER 1

INTRODUCTION

1.1 PURPOSE

This manual explains how to debug AP programs using APSIM64 and APDEBUG64.

1.2 CONVENTIONS

This manual uses the following conventions:

- The Attached Processor is called the AP, and the computer to which the AP attaches is called the host.
- In examples of dialogue at the terminal, user input is underlined to distinguish it from program or system output. Also, all user input at the terminal is terminated with a carriage return.
- The syntax of the commands depends upon the host. In this manual, examples of dialogue at the terminal use a generic syntax that is not meant to correspond to any existing host. Host-specific syntax appears in the appropriate host manual (listed in Table 1-1).
- In command syntax, uppercase characters must be entered exactly as shown.
- In command syntax definitions, underlined lowercase characters represent parameters. These parameters are replaced with unique, user-defined names.
- In command syntax descriptions, the uppercase letters of a command indicate the shortest legal command name abbreviation. The lowercase letters following the shortest legal command name abbreviation are optional.
- Square brackets ([ ]) enclosing a parameter in a command syntax definition indicate that the parameter is optional.
- An ellipsis (...) following a parameter in a command syntax definition indicates that the parameter can be repeated.
- Double asterisks indicate exponents (e.g., 2**3).

1.3 RELATED MANUALS

Table 1-1 lists FPS manuals that contain more detailed information about the AP. Table 1-2 lists related ANSI publications.

Table 1-1 Related FPS Manuals

PUBLICATION	PUBLICATION NO.
APFTN64 User's Guide	860-7479-001
APAL64 Programmer's Guide	860-7484-000
APAL64 Programmer's Reference Manual	860-7485-000
APLINK64 Manual	860-7486-000
FPS-164 Operating System Manual, Vols. 1, 2, & 3	861-7491-000
Volume 1 SJE	860-7491-004
Volume 2 APEX64/SUM	860-7491-005
Volume 3 File and Memory Management	860-7491-006
FPS-164 VAX Host Manual	860-7493-001
FPS-164 IBM/MVS Host Manual	860-7494-003
FPS-164 IBM/CMS Host Manual	860-7494-002

Table 1-2 Related ANSI Publications

PUBLICATION	PUBLICATION NO.
American National Standard Programming Language FORTRAN	ANSI X3.9-1978

1.4 TERMS AND ACRONYMS

Table 1-3 defines terms used throughout this manual.

Table 1-3 Terms and Acronyms

TERM	MEANING
APAL64	AP assembly language.
APEX64	AP executive. A set of host-resident routines that control data transfer and synchronization between the host and the AP.
APFTN64	AP FORTRAN.
breakpoint	A user-defined PS location at which APDEBUG64 halts a program, allowing the user to examine and modify the contents of memories and registers.
HASI	Host/AP software interface. Host FORTRAN program created by APLINK64 to handle communication between AP subroutines or functions and a host mainline program running under APEX64.
JDL	Job Definition Language. The System Job Executive's interactive command language, used to transfer data and programs between the AP and the host and to control program execution under the SJE operating system.
main memory	FPS-164 main memory. Stores the Single User Monitor (SUM), and data and program instruction words.
MD	Main data. The portion of main memory allocated for data storage.

Table 1-3 Terms and Acronyms (cont.)

TERM	MEANING
overlay	A logical segment of an AP program called into memory during processing. Overlaying allows the AP to execute programs too large for its physical memory.
PS	Program source. The portion of main memory allocated for storage of program instruction words.
SJE	System Job Executive. The operating mode that processes complete user jobs on the AP, using Job Definition Language (JDL).
SPAD	Scratch pad. Performs 32-bit integer and logical operations.
SUM	Single User Monitor. AP-resident portion of the AP operating system that starts and stops user tasks, performs memory allocation and initialization, and provides system services and breakpoint support.
tracepoint	A user-defined PS location at which APDEBUG64 halts a program, and displays specified memory and register information.
watchpoint	A user-defined variable reference at which APDEBUG64 halts the program whenever the variable is read and/or written.

CHAPTER 2

APSIM64/APDEBUG64 OVERVIEW

2.1 AP SIMULATION LIBRARY (APSIM64)

APSIM64 is a library of host FORTRAN routines providing a complete functional simulation of the host-dependent APEX64 software environment and an exact simulation to the bit level of the AP hardware environment. This exact duplication allows the APSIM64 library to be loaded in place of the APEX64 library to provide a simulation of the complete APEX64/AP software/hardware environment. APSIM64 has no command set, but it can be used with APDEBUG64 to do interactive debugging within a simulated AP environment. All APDEBUG64 commands described in this manual can be used for debugging on the simulator, exactly as if debugging on the AP.

NOTE

APSIM64 can be used without APDEBUG64 by linking the APSIM64 library instead of the APEX64 library. Used alone, the simulator executes AP routines and returns results exactly as the AP would. However, because the simulator does not have the execution speed of the AP, the simulator alone has limited usefulness.

All user applications that access the AP through the standard APEX64 calls can use the simulator with or without the debugger. These applications include all APMATH64 routines and all code generated by APAL64 and APFTN64.

APSIM64 includes the following features:

- Accurate bit-wise simulation of the entire AP instruction set.
- Functional simulation of the APEX64 interface and management functions.
- APSIM64 can be used with APDEBUG64 to do interactive debugging on a simulated AP.

APSIM64 has the following restrictions:

- SJE main programs cannot be run on the simulator.
- I/O is not simulated. Therefore, APSIM64 can only be used with routines meant to run under the APEX64 operating system option with no I/O.
- Only 64K of AP main memory (MD) is simulated.
- Programs using the synchronized user-directed call (UDC) method (described in the APLINK64 Manual, listed in Table 1-1) can cause problems if the AP goes into a busy loop waiting for data. With the hardware, the loop is broken when the data is passed, but with the simulator, the data is never passed and the simulator loops indefinitely.

2.2 AP INTERACTIVE DEBUGGER (APDEBUG64)

Debugging is the process of locating and removing mistakes in a program. APDEBUG64 provides an interactive method of debugging AP application programs. The user can run portions of an AP program, stop and examine the results, make program modifications, and then continue program execution.

APDEBUG64 allows the user to do the following:

- Execute programs in single steps.
- Interrupt program execution using breakpoints, watchpoints, and tracepoints.
- Display and alter the contents of AP register and memory locations.
- Display and change the values of global and local variables.
- Evaluate symbolic expressions.
- Display elapsed execution time.
- Gain access to overlay modules.
- Use APDEBUG64 with APSIM64 to do interactive debugging on a simulated AP.

APDEBUG64 can be called as a feature of the Single Job Executive (SJE) operating system, as a host FORTRAN-callable subroutine under APEX64, or as a stand-alone program from the host operating system.

The SJE version of APDEBUG64 is used to debug complete programs on the AP. The APEX64 FORTRAN-callable version is used to debug AP load modules in the host/AP execution environment. The stand-alone version is used to debug individual AP load modules.

Both the APEX64 FORTRAN-callable version and the stand-alone version of the debugger can be used with APSIM64.

2.3 USING APSIM64/APDEBUG64

To use APSIM64/APDEBUG64, use the following procedure:

1. Decide whether to debug on a simulated AP (APSIM64) or on the actual AP. (If the program is written to run under SJE, it cannot be run on the simulator.)

2. Compile APFTN64 routines using the DEBUG option. (This automatically selects the LIN, NAM ALL, and OPT 0 compiler options.) Assemble APAL64 routines using the NAM LOCAL UNREF option. (The APFTN64 User's Guide discusses compiler options, and the APAL64 Programmer's Guide discusses the assembler options.)

3. Link the object modules using APLINK64 with the SYM option. This builds the symbol table used by the debugger. (Refer to the appropriate host manual, listed in Table 1-1, for the syntax of the APLINK64 command). The APLINK64 Manual, listed in Table 1-1, discusses the linker.)

4. For APEX64 programs, link the desired libraries with the main program and the host/AP software interface (HASI), using the host linker. To debug on the simulator, link in both the APDEBUG64 and the APSIM64 libraries, in that order. To debug on the AP, link in the APDEBUG64 and APEX64 libraries, in that order. (Refer to the APLINK64 Manual and the FPS-164 Operating System Manual, Vol 2, APEX/SUM for more information on the HASI. Both manuals are listed in Table 1-1.)

5. Start the debugger, using the appropriate invoking procedure, described in Chapter 3.

6. Open the symbol table for the module to be debugged. Select the APDEBUG64 SYM option (described in Section 3.3.3) when the debugger is invoked for stand-alone access or execute the SY debugger command (described in Section 4.3) for SJE and host program-callable access.

7. Set breakpoints, examine the program, and make modifications as desired.

8. Enter the QUIT command (described in Section 3.4) to end the debugging session.

2.4 EXAMPLE DEBUGGING SESSIONS

The following examples show typical debugging sessions. Lines entered by the user are underlined. The asterisk (*) prompts the user for input. A dollar sign ($) precedes predefined memory and register names on output to distinguish them from any user-defined symbol names with the same name. Double quotes (") precede comment lines.

The following is part of the APFTN64 listing of the program ADDMAT. The APFTN64 listing shows the source line numbers used by APDEBUG64. (It is often helpful to refer to a listing while debugging an APFTN64 program.)

SOURCE INPUT FILE: ADDMAT.FOR

```
LINE NUMBER                    SOURCE INPUT

  (00001)          PROGRAM ADDMAT
  (00002)          INTEGER XMAT (10), YMAT (10), ZMAT (10)
  (00003)          DO 50 I=1,10
  (00004)          XMAT(I)=I
  (00005)          YMAT(I)=10-I
  (00006)          ZMAT(I)=XMAT(I)+YMAT(I)
  (00007)          WRITE(5,*) 'XMAT=',XMAT(I),'YMAT=',YMAT(I),'ZMAT=',ZMAT
  (00008)  50      CONTINUE
  (00009)          STOP
  (00010)          END
```

In the following example debugging session, the debugger is invoked in SJE to debug the program ADDMAT. Notice that the debugger displays the last location examined each time it encounters a breakpoint.

```
$ sje
SJE-I-WELCOME, SJE REL E00-000  VER 2.00     06/22/83  11:45.

SJE> attach/wait
SJE-I-ATTACH, Assigned AP number   1.
SJE> copyin/b addmat.img
SJE-I-COPYIN, File copied in.
SJE> debug/defer
SJE-I-DEBUG, Debugger activated.
SJE> addmat.img
APDEBUG64,  REL E00-000 VER 3.00
LANGUAGE IS FTN
* "
* "open symbol file 'addmat.asy' and local symbols
* "for the module 'addmat' (Section 4.3)
* sym addmat 'addmat.asy'
* "
* "check that symbols are open (Section 4.3)
* sym
SYMBOL FILE OPEN:    ADDMAT.ASY
GLOBAL SYMBOLS OPEN
LOCAL SYMBOLS OPEN:  ADDMAT
* "
* "set a breakpoint at the line labelled 50 (Section 5.3)
* b .50
* "
* "run the program ADDMAT (Section 5.7)
* r
XMAT=            1 YMAT=            9 ZMAT=           10
PROG BREAK  IN ADDMAT AT LINE $8  (.50) ;  6240.667 us.
* "
* "examine the first elements of arrays XMAT, YMAT, and ZMAT (Sec. 6.3)
* e xmat(1)
XMAT(1) = 1
* e ymat(1)
YMAT(1) = 9
* e zmat(1)
ZMAT(1) = 10
* "
* "set a breakpoint at source line 8
* "breakpoint to be taken every fourth loop (Section 5.3)
* b/every:4 $8
* "
* "run the program ADDMAT (Section 5.7)
* r
XMAT=            2 YMAT=            8 ZMAT=           10
XMAT=            3 YMAT=            7 ZMAT=           10
XMAT=            4 YMAT=            6 ZMAT=           10
XMAT=            5 YMAT=            5 ZMAT=           10
PROG BREAK  IN ADDMAT AT LINE $8  (.50) ;  15501.000 us.
ZMAT(1) = 10
* r
XMAT=            6 YMAT=            4 ZMAT=           10
XMAT=            7 YMAT=            3 ZMAT=           10
```

```
XMAT=             8 YMAT=             2 ZMAT=            10
XMAT=             9 YMAT=             1 ZMAT=            10
PROG BREAK  IN ADDMAT AT LINE $8  (.50) ;  24761.333 us.
ZMAT(1) = 10
* "
* "examine element 6 of the array ZMAT (Section 6.3)
* e ZMAT(6)
ZMAT(6) = 10
* "
* "exit from the debugger (Section 3.4)
* q
SJE-I-EXIT, Program exit.
SJE> detach
SJE-I-DETACH, AP detached.
SJE> q
SJE-I-QUIT, SJE stopped.
$
```

The following example illustrates the debugger being called automatically from a HASI running under APEX64 control, and invoking a subroutine.

```
OK, Run PROGX
APDEBUG64, Release E00
LANGUAGE IS FTN
*"open local symbol table
*sy progx.sym
*"set a breakpoint at line 68 in the module csubi in the overlay
*"char (Section 5.3)
*br char\\csub1\$68
*"start program execution (Section 5.7)
*r
PROG BREAK  IN CSUB1 AT LINE $68  (.L0001) ;  597.833 us.
*"examine array A3 and the variable C9 (Section 6.3)
*e a3(1):a3(4); e c9
A3(1) = 'A31'
A3(2) = 'CBA'
A3(3) = '654'
A3(4) = 'FED'
C9 = 'C568  GHI'
*"continue program execution (Section 5.7)
*r
PROGRAM RETURN TO HOST ;  600.667 us.
*"end debugging session (Section 3.4);
*"return to user program on host (PROGX)
*q
```

In this example, the simulator is called as a stand-alone program from the host operating system and automatically invokes the debugger.

```
OK, APSIM64 USER.PROG SYM
APDEBUG64, Release E00
AP 1   ASSIGNED
LANGUAGE IS FTN
*"put debugger in APAL64 mode (Section 4.2)
*lang $apal
LANGUAGE IS APAL64
*"check state of symbol file and local symbol table (Section 4.3)
*sy
SYMBOL FILE OPEN:    USERPROG.SYM
GLOBAL SYMBOLS OPEN
LOCAL SYMBOLS OPEN:  PROG1
*"change local symbol table's load module (Section 4.3)
*sy vma
*"check state of symbol file and local symbol table (Section 4.3)
*sy
SYMBOL FILE OPEN:    USERPROG.SYM
GLOBAL SYMBOLS OPEN
LOCAL SYMBOLS OPEN:  VMA
*"set breakpoint at the location labelled 'loop' (Section 5.3)
*b/every:2 loop
*"examine the variable 'fa' (Section 6.3)
*e fa
$FA = -1.00000000000000000
*"start program execution, stopping at the second breakpoint
"(Section 5.7)
*r/2
PROG BREAK  IN VMA AT LOOP+1 ;  17.333 us.
$FA = 8.00000000000000000
PROG BREAK  IN VMA AT LOOP+1 ;  19.000 us.
$FA = 9.00000000000000000
*"step and proceed (Section 5.10)
*s/p/100
PROG STEP   IN VMA AT LOOP+2 ;  19.167 us.
$FA = 1.00000000000000000
PROG STEP   IN VMA AT LOOP+3 ;  19.333 us.
$FA = 9.00000000000000000
PROG STEP   IN VMA AT LOOP ;  19.500 us.
$FA = 1.00000000000000000
PROG STEP   IN VMA AT LOOP+1 ;  19.667 us.
$FA = 5.090000000000000060
PROG STEP   IN VMA AT LOOP+2 ;  19.833 us.
$FA = 0.00000000000000000
PROG STEP   IN VMA AT LOOP+3 ;  20.000 us.
$FA = 5.090000000000000060
PROG STEP   IN VMA AT DONE ;  20.167 us.
$FA = 0.000000000000000000
PROG STEP   IN USER.PROG AT USER.PROG+19 ;  20.333 us.
$FA = 0.000000000000000000
PROG BREAK  IN USER.PROG AT USER.PROG+32 ;  20.500 us.
$FA = 0.000000000000000000
```

```
*"examine SPADs 0 through 3 (Section 6.3)
*e sp(0):sp(3)
$SP(00) = 00000060
$SP(01) = 00000001
$SP(02) = 00000065
$SP(03) = 00000001
*"zero all SPADs (Section 6.9)
*z sp
*"examine SPADs 0 through 3 (Section 6.3)
*e sp(0):sp(3)
$SP(00) = 00000000
$SP(01) = 00000000
$SP(02) = 00000000
$SP(03) = 00000000
*"check execution time since last run or step command
*" (Section 5.13)
*time
2.667 us.
*"examine MA (Section 6.3)
*e ma
$MA = 00000026
*"end debugging session (Section 3.4)
*q
```

CHAPTER 3

RUNNING APSIM64/APDEBUG64

3.1 INTRODUCTION

This chapter describes the commands that invoke and exit APSIM64 and APDEBUG64 and some special features for doing batch debugging. Table 3-1 lists these commands and their functions:

Table 3-1 Commands for Running APSIM64/APDEBUG64

COMMAND	MINIMUM ABBREVIATION	FUNCTION
APSIM64	APSIM64	invokes stand-alone version of the debugger running on the simulator
APDEBUG64	APDEBUG64	invokes stand-alone version of the debugger
DEBUG	DEBUG	invokes APDEBUG64 from SJE
INIT	INI	initializes the AP
QUIT	Q	exits from APDEBUG64 or APSIM64
LOAD	LO	loads modules into the AP during debugging
"		comment line (for batch debugging)
;		command separator (for batch debugging)
%		command continuation (for batch debugging)

3.2 INVOKING APSIM64

APSIM64 can be invoked in the following two ways:

- Called automatically by APEX routines called in the host program or HASI.
- Stand-alone access from the host operating system. (This version of APSIM64 automatically invokes APDEBUG64 on the simulated AP.)

The next two sections describe how to invoke APSIM64.

3.2.1 Host Program-callable APEX64 Access

Calling the simulator from a host program running under APEX64 allows the user to develop (and debug) an AP module within its complete operating environment using a simulated AP, instead of a real AP. In this case, the simulator is called from within the host program by the same routines that would normally call the AP.

To access APSIM64 from within a host program, use the host linker to link the program with the APSIM64 library instead of the APEX64 library. If debugging is desired, link the APDEBUG64 library at the same time. When the host program is started, the HASI automatically calls the simulator instead of the AP. The AP routines run on the simulated AP.

3.2.2 Stand-alone Access

The stand-alone version of APSIM64 combines the simulator with the debugger, allowing the user to debug individual load modules on a simulated AP.

The following host system-level command invokes the simulator and the debugger (refer to the specific host manual, listed in Table 1-1, for the exact host command syntax):

APSIM64 [loadmodfile]... [params]

For a description of the parameters (params) used in this command, refer to Table 3-2 in Section 3.3.3.

The QUIT command (described in Section 3.4) returns control to the host operating system.

3.3 INVOKING APDEBUG64

APDEBUG64 can be invoked in the following three ways:

- a call from the Single Job Executive (SJE) operating system
- a call from within a HASI program in the APEX64 environment
- stand-alone access from the host operating system

APDEBUG64 functions the same way no matter how it is invoked. The next three sections describe each method.

3.3.1 Single Job Executive (SJE) Access

Under the SJE operating system option, the user can invoke APDEBUG64 before executing the program, after the program is interrupted, or after completing the program. The SJE access method supports debugging with overlays, while the other two methods do not.

The following SJE command invokes the debugger:

DEBUG [/option]

In this command, the option is either NOW or DEFER. The NOW option starts the debugger immediately. The DEFER option defers execution until the next user program is initiated. The default is DEFER.

The QUIT command (described in Section 3.4) returns control to SJE.

3.3.2 Host Program-callable APEX64 Access

Calling the debugger from a HASI program running under APEX64 allows the user to debug an AP module within its complete operating environment rather than as an isolated load module. In this case, the debugger is called from within the HASI program after the load module and data are transferred to the AP.

To access APDEBUG64 from within a HASI, use the host linker to link the program with the APDEBUG64 library. When the host program is started, the debugger is automatically called from the HASI just before each execution of an AP routine, allowing the user to set breakpoints, examine AP locations, step through the AP code, and make changes where appropriate.

The user can also call the debugger by inserting a "CALL DBUG64" statement in the host program at the desired location.

CAUTION

Calling APDEBUG64 from within a host program using the CALL DBUG64 statement is not the preferred method and must be used with great care. The user must transfer all data and load modules to the AP before the call is executed. In addition, when the user quits APDEBUG64, control returns to the host FORTRAN program, which might call APEX64 to start the load module executing again.

The QUIT command (described in Section 3.4) returns control to the user program.

3.3.3 Stand-alone Access

The stand-alone version of APDEBUG64 allows the user to debug individual load modules.

The following host system-level commands invoke the debugger (refer to the specific host manual, listed in Table 1-1, for the exact host command syntax):

APDEBUG64 [loadmodfile]... [params]

The user can supply the parameters (params) defined in Table 3-2.

Table 3-2 User-supplied Parameters (Stand-alone Access)

PARAMETER	MINIMUM ABBREVIATION	MEANING
[NO]LIST [filnam]	[NO]L	[Do not] create a listing file. The default is NOL.
[NO]SYMBOLS [filnam]	[NO]SYM	[Do not] obtain symbols from symbol name file. The default is NOSYM.
APNUM apnumber	AP	Select AP number apnumber. The default is AP 0 (zero), which means any available AP.

Table 3-2 User-supplied Parameters (Stand-alone Access) (cont.)

PARAMETER	MINIMUM ABBREVIATION	MEANING
PSSIZE n	PS	Specify the minimum amount of PS memory needed in the requested AP. The debugger rounds up to the nearest 4K if n is not a multiple of 4096. The default is 4K words of PS memory.
MDSIZE n	MD	Specify the minimum amount of MD memory needed in the requested AP. The debugger rounds up to the nearest 4K if n is greater than zero and not a multiple of 4096. If n is -1, the debugger allocates for MD all memory remaining after PS is allocated. The default is 4K words of MD memory.
[NO] TMRAM	[NO]TM	[Do Not] assign TMRAM to this job. User cannot deposit to TMRAM during the debugging session unless TMRAM is explicitly assigned. Default is NOTMRAM.
[NO] WAIT	[NO]W	[Do Not] wait for the AP if it is busy. Default is NOWAIT.

A listing output file can be generated which contains everything shown on the user's terminal (including error messages). Default output file extensions depend upon the host.

When the debugger is invoked, it initializes the AP and loads any specified load module(s) before entering interactive mode.

The stand-alone debugger only allocates 4K words of memory for MD and PS. If the load modules require more than 4K words of PS or MD memory, the user must allocate sufficient memory, using the PSSIZE and MDSIZE parameters, described above. (The debugger displays an error message if a module too large for the default PS and MD settings is loaded.)

The QUIT command (described in Section 3.4) returns control to the host operating system.

3.4 EXIT FROM APDEBUG64 (QUIT)

To exit from the debugger, enter:

```
Quit
```

When running the stand-alone version, the QUIT command returns the debugger to the host operating system. When running the host program-callable version, the QUIT command returns the debugger to the calling routine.

When running under the SJE operating system option, the QUIT command returns control to SJE.

Before quitting, all files opened by APDEBUG64 are closed, and all breakpoints are cleared.

3.5 LOADING THE AP (LOAD)

The user can load new load modules into the AP at any time during interactive debugging using the following command:

```
LOad ['file']
```

The file argument specifies the name of the file containing the desired load module. The file name must be enclosed by apostrophes. If no file name is entered, the debugger reloads any load modules specified with the APDEBUG64 command (described in Section 3.3.3).

3.6 INITIALIZING THE AP (INIT)

Use the INIT command to initialize the AP and reset PS size, MD size, or TMRAM access during a debugging session.

The INIT command initializes the AP by calling the APEX64 routine APINIT. To initialize the AP, enter:

```
INIt [ /PSSIZE:n ] [ /MDSIZE:n ] [/TMRAM]
```

The PSSIZE:n MDSIZE:n, and TMRAM parameters have the meanings described in Table 3-2.

3.7 BATCH DEBUGGING

APDEBUG64 is meant to be used interactively. Hosts that do not support interactive dialogue severely restrict the debugger's usefulness. If the host does not support interactive dialogue, debugging can be done as a batch job using host command procedures.

The following sections describe APDEBUG64 features which are useful for doing batch debugging.

3.7.1 Comment Line (")

To indicate a comment line to the debugger, enter the following:

```
"comments
```

APDEBUG64 ignores the remainder of the input line, as in the following example:

```
*E LIST:LIST+20:5 "Examine the 1st row of LIST
```

3.7.2 Command Separator (;)

A semicolon (;) separates multiple commands on one line, as in the following example:

```
*R LOOPX ; E COUNT
```

The syntax of a command is checked when it is executed. If a syntax error is detected in a string of commands, execution stops immediately, the remaining commands are ignored, and the debugger returns to the terminal for further input.

3.7.3 Command Continuation (%)

To continue a command on the next line, end the current line with a percent sign (%). If the line contains a comment, the % must be placed before the beginning of the comment, as in the following example:

```
*DEPOSIT VAR100+20:VAR100+30:2 % "This line continues
 =5432.98E+50
```

CHAPTER 4

SETTING I/O PARAMETERS

4.1 INTRODUCTION

This chapter describes the commands that select the input and output formats used when examining and changing registers and memory locations. Table 4-1 lists these commands and their functions:

Table 4-1 Input/Output Format Commands

COMMAND	MINIMUM ABBREVIATION	FUNCTION
LANGUAGE	LA	set the language mode
SYMBOL	SY	open symbol table
RADIX	RA	set integer radix
FORMAT	FO	set I/O format

APDEBUG64 selects the proper radix and format for output depending on the particular register or memory location that is open and the setting of the above commands. For input, values must be entered in the global radix unless the radix overrides are used (described in Section 4.4); any format can be used.

4.2 SET THE LANGUAGE MODE (LANGUAGE)

To set the language mode of APDEBUG64, enter one of the following:

```
LAnguage [$APAL]

LAnguage [$FTN]
```

The $APAL argument sets the language mode to APAL64. The $FTN argument sets the language mode to APFTN64. The dollar signs ($) preceding APAL and FTN identify these arguments as APDEBUG64-defined symbols, not user-defined symbols (described in Section 5.2). The choice of language mode affects the operation of the debugger in the following ways:

- In FTN mode, the default output radix is decimal. In APAL mode, the default output radix is hexadecimal. (Refer to Section 4.4.)

- In FTN mode the default output format is the data type of the variable examined. In APAL mode, the default output format is determined by the register or memory location being examined. (Refer to Section 4.5.)

- In FTN mode a breakpoint occurs before the source line is executed. In APAL mode, the breakpoint occurs after the source line is executed. (Refer to Section 5.3.)

- In FTN mode, a program step is defined as one source code statement. In APAL mode, a step is defined as one APAL64 instruction. (Refer to Section 5.10.)

- When printing a procedure call traceback in FTN mode, the debugger displays the source code line number and corresponding PS address at which the procedure was called. In APAL mode, the debugger displays only the PS address. (Refer to Section 5.15.)

- The relational operators for evaluating expressions are different in FTN and APAL modes. (Refer to Table 6-4.)

- In FTN mode, the DISPLAY command displays the value of a value expression. In APAL mode, the DISPLAY command displays the address of a value expression. (Refer to the example in Section 6.5.)

The default language mode is FTN. Whenever the debugger opens the local symbol table for a subprogram, the language mode is set to the language mode of the routine.

If no arguments appear with the LANGUAGE command, the debugger displays the current language mode.

4.3 OPEN OR CLOSE SYMBOL TABLES (SYMBOL)

Opening symbol tables allows any local or global symbol names appearing in APAL64 or APFTN64 source statements to be used in command input and output.

Global symbols are entry points or variable names that can be referenced by another program unit. Local symbols are variable names and program locations (such as statement labels) which can be referenced only from within the program unit defining them.

The following command opens or closes a global or local symbol table:

SYmbol [/CL] [name] ['file']

The CL (close) modifier directs the debugger to close any global or local symbol tables that are currently open.

The name argument specifies the name of the object module whose local symbol table is to be opened; any currently open local symbol table is closed. The name has the form [overlayname\\]modulename. The modulename is the name used in the APPROGRAM, APROUTINE, SUBROUTINE, FUNCTION, or ENTRY statement in the module. If the main program has no PROGRAM statement, the default name (generated by the APFTN64 compiler) is $MAIN$. The default name can be used for the name argument in either FTN or APAL mode.

Opening a local symbol table also opens the global symbol table. If name is specified along with the CL modifier, only the local symbol table of the module is closed; the global symbol table remains open. If name is not specified along with the CL modifier, the global and local symbol tables and the symbol file itself are closed.

The file argument identifies the file containing the global or local symbol table. Opening a new symbol table file automatically closes the current symbol table file.

Unless a module name is specified with the SYMBOL command, opening a symbol file automatically opens the local symbols for the object module pointed to by the current program location. If a symbol file is open, APDEBUG64 automatically opens the local symbols for the AP routine in which the program stopped after a RUN or STEP command. (If the stand-alone or SJE version is invoked and the user has not run a program yet, there is no current program location. In this case, the user must specify a module name to open a local symbol table.)

If all modifiers and arguments are omitted, the debugger displays the name of the symbol file that is currently open and the name of the object module whose local symbols are open.

Example:

- Open the local symbols for the object module named PROG1; examine the instructions at the symbolic locations PROG1, PROG1+1, and PROG1+2, and modify one of the instructions.

```
*SY PROG1 'PRGSYM'
*E PROG1:PROG1+2
 PROG1 = 42A8000000000030
 PROG1+1 = 42EC000000000000
 PROG1+2 = 0000000000000000
*D PROG1+1<MA>=3
*E
 PROG1+1 = 42EC000000000030
*
```

4.4 SET INTEGER RADIX (RADIX)

The following command sets or lists the integer input/output radix:

RAdix [/radix]

A slash (/) always precedes the radix modifier.

The value of the radix modifier can be B (binary), O (octal), D (decimal), or Z (hexadecimal). The default radix is hexadecimal in APAL64 mode and decimal in FTN mode. If the modifier is omitted, the debugger displays the current radix.

The current integer radix can always be overridden for integer input by entering the integer in the form, r'n', where r is the radix indicator described above and n is the integer number. For input, integers of all radices can be optionally signed (+ or -). A minus (-) means two's-complement negation. Unless hexadecimal numbers are preceded by Z', they must always begin with a decimal digit (0-9) so that they can be distinguished from symbols.

Binary, octal, and hexadecimal integers are displayed with leading 0's to the width of the item being displayed. Decimal integers are displayed without leading zeros. Negative integers in all radices are displayed left-justified and blank-filled.

This command sets the default radix for both input and output.

Example:

- Examine SPAD register Z'A' in decimal, binary, octal, and hexadecimal.

```
*RA/D
*RA
 RA = D
*E SP(Z'A')
$SP(10) = 32768
*E/B
$SP(10) = 00000000000000001000000000000000
*E/O
$SP(10) = 00000100000
*E/Z
$SP(10) = 00008000
*
```

SPAD register 10 (decimal) has the following value:

```
32768 (radix 10)
00000000000000001000000000000000 (radix 2)
00000100000 (radix 8)
00008000 (radix 16)
```

4.5 SET OUTPUT FORMAT (FORMAT)

The FORMAT command overrides the default output format for the contents of an AP functional unit (memory or register).

NOTE

For FORTRAN variables, the default output format is the data type of the variable and can only be overridden by using a format specifier on the E (examine) command (Section 6.3), + or - (succeed or precede) command (Section 6.4), or DI (display) command (Section 6.5).

The following command sets or lists the output format for register and memory contents:

FOrmat [/format] [funit]

A slash (/) always precedes the format modifier.

The format modifier is one of the following:

R Specifies real (floating-point) numbers of the form:

+x.yE+z

where x is the possible size of the integer part, y is the possible size of the fraction part, and z is the possible size of the exponent (all in decimal). The symbol + indicates that either a plus (+) or a minus (-) can be used. This is the default format for MD, TM, DPX, DPY, FA, and FM.

F Specifies field format. Each of the field names are displayed along with the field value. Valid fields for PS memory and the status and I/O registers are listed in Appendix A. This format is not used for input. This format is the default for the APSTAT register and the user's copy of the I/O registers. (Refer to the description of the REG argument, below.)

xIn Specifies n-bit byte integer format, where n is a decimal integer between 1 and 64. The bytes are right-justified in the word, so the left-most byte can contain fewer than the full n bits. Commas separate the bytes. The x is replaced with S, U, or nothing, as follows:

SIn Specifies signed integers.

UIn Specifies unsigned integers. UI64 is the default for I/O. UI32 is the default for PS memory.

In Is the same as S when in decimal radix and the same as U when in all other radices. I32 is the default for SPAD and SRS.

A Specifies an ASCII constant of the form:

A'c'

where c is a sequence of one to eight ASCII characters. Apostrophes (') in c must appear twice for each desired occurrence (e.g., 'I''M DONE'). The debugger left-justifies and blank-fills each 8-bit byte (e.g., 'DONE ').

The FORMAT command applies to output only. Values can be entered in any format. Values can be entered in two or more bytes only if the format for the functional unit being deposited into is specified as bytes, or the byte format is specified on the deposit command.

Note that only the SI32 and SI64 formats can produce negative integers.

The funit argument specifies the functional unit to which the format applies. This argument can be any of the following:

DPX	TM	FM
DPY	SP	IO
MD	REG	SRS
PS	FA	$ALL

The memory name (DPX, DPY, MD, PS, TM, or SP) specifies that the format applies to that particular memory. The MD and TM pipeline register contents are displayed in the same format as the contents of the MD and TM memories. The contents of SPFN are displayed in the same format as the SPAD memory (SP).

REG specifies that the format applies to the APSTAT (AP status) register and the user's copy of the I/O registers. (The user's copy contains the I/O registers' values when breakpoints occur; these values are restored when program execution resumes.)

FA and FM specify that the format applies to the adder pipeline contents and the multiplier pipeline contents, respectively.

IO specifies the AP's copy of the I/O registers. (The AP's copy contains the I/O registers' values when the SUM is running; these are real-time values.)

SRS specifies that the format applies to the subroutine return stack.

$ALL specifies that the format applies to all functional units.

If the modifier and argument are omitted, the debugger displays the current format settings.

These commands, arguments, and modifiers are illustrated in the following examples:

- Set the integer radix to octal and examine DPX register 6 in each of four numeric data word formats (real, 64-bit integer, 32-bit integer, and 53-bit integer).

```
*RA/O
*FO/R DPX
*E DPX(6)
$DPX(06) = -1.0
*E/I64
$DPX(06) = 1000200000000000000000
*E/I32
$DPX(06) = 20004000000,00000000000
*E/I53
$DPX(06) = 2000,200000000000000000
*
```

DPX register 6 contains the following:

```
-1.0 (floating)
1000200000000000000000 (I64 format)
20004000000,00000000000 (I32 format)
2000,200000000000000000 (I53 format)
```

- Examine main memory locations 100 through 103 in character string format and deposit a correct character string into location 103.

```
*E/A MD(100):MD(103)
$MD(100) = A'This     '
$MD(101) = A'example  '
$MD(102) = A'isn''t    '
$MD(103) = A'two long'
*D MD(103)=A'too long'
*
```

MD location 103 contained "two long" and now contains "too long".

CHAPTER 5

EXECUTING PROGRAMS

5.1 INTRODUCTION

The typical strategy when debugging a program is to set a breakpoint at one or more key locations and then run the program. Various data locations are examined when the program stops at the breakpoint. This strategy results in alternately running a program and examining the results.

The commands listed in Table 5-1 control AP program execution.

Table 5-1 Execution Commands for AP Programs

COMMAND	MINIMUM ABBREVIATION	FUNCTION
BREAK	B	set or list breakpoints
TRACE	T	set or list tracepoints
WATCH	W	set or list a watchpoint
CLEAR	C	clear break/trace/watch points
RUN	R	run user program
attention key		stop user program
PTIMEOUT	PTI	set or list program time-out period
STEP	S	single step through program
WHERE	WH	print current location
INVOKE	INV	invoke a subroutine or function
TIME	TI	print elapsed execution time
POVERLAY	PO	print overlay structure diagram
PTRACE	P	print a procedure call traceback
FBUF	FB	fill command buffer
BUFFER	BU	toggle command buffer switches
XBUF	XB	execute command buffer
LIST	LI	open or close a listing file
STATUS	STA	print debugger status

This chapter discusses these commands and the parameters used with them in detail.

5.2 ADDRESS EXPRESSION PARAMETERS

Whenever addexp appears as a command argument, it is an address expression. Address expression parameters are used with the BREAKPOINT, TRACEPOINT, and WATCHPOINT commands described in Sections 5.3, 5.4, and 5.5.

The next two sections define FTN and APAL64 address expression command parameters.

5.2.1 FTN Address Expressions

In FTN mode, an address expression is any of the following:

- A variable name.
- An array element name. (Subscript expressions cannot contain function references.)
- A line number at an executable statement (prefixed by a dollar sign ($)). (Refer to Section 5.3 for a list of executable APFTN64 statements.)
- A statement label (prefixed by a period (.)).
- A procedure entry name (defined by a SUBROUTINE, FUNCTION, APROUTINE, or ENTRY statement).

The type of the address expression can be PS (when the symbols used are line numbers, statement labels, or entry points), MD (when the symbols used are variables or array elements), or untyped (when no symbols are used or when the symbols used have been defined in a parameter statement).

5.2.2 APAL64 Address Expressions

In APAL64 mode, an address expression is any valid APAL64 expression with the following restrictions and additions:

- The value of a register specifier is the contents of the register. It is untyped.
- The following special symbols designate the contents of various AP registers:

@MA	main memory addresses
@PSA	program source addresses
@DPA	data pad X addresses
@DA	input/output device addresses
@SRA	subroutine return stack addresses
@TMA	table memory addresses

These symbols are predefined symbols. The symbol value is the current contents of the register specified (i.e., a memory address). Symbol type is determined by the type of the memory to which it refers.

The memory type of symbols used in the expression determines the type of the expression. All symbols and the type operator in the expression must be of the same type except for untyped, which is compatible with any other type. An expression is untyped if it contains no symbols. An expression can be made untyped by using the INT operator. The APAL64 Programmer's Reference Manual (listed in Table 1-1) contains more information on APAL64 expressions.

5.2.3 Local Symbol Overrides

If a local symbol has the same name as an AP register or memory specifier, the local symbol takes precedence. Prefixing the register or memory name with a dollar sign ($) overrides this precedence. The prefix causes the symbol to be interpreted as an APDEBUG64-defined symbol rather than a user-defined symbol.

On output, a dollar sign ($) precedes predefined memory and register names to distinguish them from any user symbols with the same name.

5.2.4 Path Names

A path name can be used wherever a local symbol is used. A path name references a local symbol outside the module whose symbol table is currently open. For example, module name, entry point, and local symbols defined in an overlay must be qualified with the overlay name to distinguish them from symbols of the same name in either the main program or in another overlay.

A path name has one of the following forms:

```
overlayname??modulename?localsymbol
overlayname??modulename
modulename?localsymbol
overlayname??globalsymbol
overlayname
localsymbol
```

The backslash (\) can be used instead of the question mark (?). For example:

```
overlayname\\modulename\localsymbol
```

Two question marks (or backslashes) separate an overlay name from a module name, and a question mark (or single backslash) character separates a module name from a local symbol. The overlay name is required only when the path name refers to a symbol defined in an overlay segment.

5.3 SET OR LIST BREAKPOINTS (BREAK)

A breakpoint is a user-defined PS location at which APDEBUG64 halts a program, allowing the user to examine and modify the contents of memories and registers. No more than two breakpoints or tracepoints can be set at one time. To set or list a breakpoint, enter one of the following:

```
Break [/option...] [/ARG] addexp

Break [/option...] [/ARG] $CALLS [$RETURNS]

Break [/option...] [/ARG] $RETURNS

Break [/option...] $OVERLAYS

Break [/option...] ovname\\

Break
```

The option modifier describes when the breakpoint is taken. The option modifier has one of the following values:

EVERY:n	Specifies that the breakpoint is taken every n times it is encountered. EVERY:1 is the default value.
AFTER:n	Specifies that the breakpoint is taken only after it is encountered n times.
UNTIL:n	Specifies that the breakpoint is taken each time it is encountered until it has been encountered n times. The breakpoint is then cleared.
IF (relexp)	Specifies that the breakpoint is taken only if the relational expression (relexp) evaluates to true. The expression is evaluated each time the breakpoint occurs.

The modifier count n applies to each breakpoint separately (i.e., n is not a cumulative total of all breakpoints).

The ARG (argument) modifier directs the debugger to display the values of the called procedure's arguments each time the breakpoint occurs. For APAL64 routines, this modifier is only valid when setting breakpoints at procedure entry points defined by the $PENTF pseudo-op (described in the APAL64 Programmer's Reference Manual). For APFTN64 routines, this modifier can only be used with the $CALLS argument or on breakpoints which occur at or after the first executable FORTRAN statement in the routine.

The executable FORTRAN statements are:

- arithmetic, logical, and statement label (ASSIGN) assignment statements
- unconditional, computed, and assigned GO TO statements
- arithmetic and logical IF statements
- block IF, ELSE IF, ELSE, and END IF statements
- DO statement
- CONTINUE statement
- READ, WRITE, and PRINT statements
- REWIND, BACKSPACE, ENDFILE, OPEN, CLOSE, WAIT, and INQUIRE statements
- CALL and RETURN statements
- STOP and PAUSE statements
- CALL EXIT
- END statement

For an array argument, only the first element of the array is displayed. (The EXAMINE command can be used to examine the entire array.)

The addexp argument specifies the program location where the break is set. It must be a program address or program symbol (line number, statement label, or entry point).

The $CALLS and $RETURNS arguments indicate that breakpoints are set at subroutine or function calls and returns. In APAL64 mode, breakpoints are set at all entry points defined by the currently open symbol file regardless of whether any local symbol information is available. In FTN mode, breakpoints are set at the first executable statement of each APFTN64 subprogram and at the entry address for each APAL64 entry point for which local symbol information exists in the currently open symbol file.

The $OVERLAYS argument indicates that a breakpoint is taken (subject to the option modifier) when any overlay is brought into memory, before any instructions in the overlay are executed.

The ovname argument specifies that a breakpoint is taken when the named overlay is brought into memory. The overlay name must be defined in the APLINK64 overlay description file used in linking the load module. (The linking process is described in the APLINK64 Manual, listed in Table 1-1.)

Breakpoints can be set on code in an overlay even when the overlay is non-resident. The debugger automatically inserts and removes these breakpoints as an overlay's memory residency status changes.

If no arguments are specified, the debugger displays the current breakpoint setting. A maximum of two breakpoints and tracepoints can be set at a time. Breakpoint settings forced by the $CALLS and $RETURNS options count as one breakpoint. Breakpoints set by the $OVERLAYS argument or the ovname\\ parameter do not count toward the maximum of two breakpoints and tracepoints.

NOTE

In APAL64 mode, the breakpoint is taken after the specified instruction is executed, and in FTN mode, the breakpoint occurs before the source line is executed.

Examples:

- In APAL64 mode, set a breakpoint so that the program stops after executing the instruction at PS location 20.

```
*B 20
*
```

- In FTN mode, set a breakpoint so that the program stops every fifth time it reaches line number 100.

```
*B/EVERY:5 $100
*
```

5.4 SET OR LIST TRACEPOINTS (TRACE)

A tracepoint is a user-defined PS location at which APDEBUG64 halts a program, and displays specified memory and register information. A maximum of two breakpoints and tracepoints can be set at one time. To set or list a tracepoint, enter one of the following:

```
Trace [/option...] [/ARG] addexp

Trace [/option...] [/ARG] $CALLS

Trace [/option...] [/ARG] $RETURNS

Trace [/option...] $OVERLAYS

Trace [/option...] ovname\\

Trace
```

The modifiers and arguments for the TRACE command are the same as for the BREAK command (described in Section 5.2).

APDEBUG64 does not return control to the user at a tracepoint as it does at a breakpoint. Instead, APDEBUG64 continues execution of the program after displaying the appropriate information.

A maximum of two breakpoints and tracepoints can be set at a time. Tracepoint settings forced by the $CALLS and $RETURNS options count as one tracepoint. Tracepoints set as a result of the $OVERLAYS argument or the ovname\\ parameter do not count toward the limit.

Tracepoints can be set on code or data locations in an overlay even when an overlay is non-resident. The debugger automatically inserts and removes these tracepoints as an overlay's memory residency status changes.

If no argument appears with the command, the debugger displays the current tracepoint settings.

5.5 SET OR LIST A WATCHPOINT (WATCH)

A watchpoint is a user-defined variable reference at which APDEBUG64 halts a program whenever the variable is read and/or written. Only one watchpoint can be set at a time. The following command sets or lists a watchpoint:

```
Watch [/option] [addexp]
```

The option modifier can have the values shown for the BREAK command plus the following values:

RD Specifies that the watchpoint is taken whenever the location is read.

WR Specifies that the watchpoint is taken whenever the location is written. WR is the default option modifier.

RW Specifies that the watchpoint is taken whenever the location is either read or written.

NOTE

The APFTN64 compiler generates code that causes implicit reads of MD. These reads cause MD read watchpoints to occur at program locations that are not actually using the data from the MD read. This problem is unavoidable; the programmer must be aware that these spurious reads can occur when a read watchpoint (RD or RW) is requested.

The addexp argument specifies the data location to be watched. Note that addexp is a data location, not a program location. The program halts whenever the location is read or written.

The debugger can only watch one 64-bit data word at a time. A watchpoint set at a character datum cannot catch a read or write into that part of the character string that extends past the first 64-bit word. Also, the watchpoint can occur when the character datum itself is not actually being accessed, such as when the character datum starts in the middle of a word.

A watchpoint can be set on a data location in an overlay even when the overlay is nonresident. The debugger automatically inserts and removes this watchpoint as an overlay's memory residency status changes.

If no argument is used with the command, the debugger displays the current watchpoint setting.

Example:

- Set a watchpoint to halt every time VAR1 is changed.

```
*W/WR VAR1
*
```

5.6 CLEAR BREAKPOINTS, TRACEPOINTS, AND WATCHPOINT (CLEAR)

To clear a breakpoint, tracepoint, or watchpoint, enter one of the following:

```
Clear addexp

Clear $CALLS

Clear $RETURNS

Clear $OVERLAYS

Clear ovname\\

Clear $ALL
```

The addexp argument specifies the location of a breakpoint, tracepoint, or watchpoint to clear.

The $CALLS argument directs the debugger to clear all breakpoints and tracepoints that were set using the $CALLS argument with the BREAK or TRACE commands. The $RETURNS argument directs the debugger to clear all breakpoints and tracepoints that were set using the $RETURNS argument with the BREAK or TRACE commands.

The $OVERLAYS argument directs the debugger to clear the breakpoint or tracepoint that was set using the $OVERLAYS argument with the BREAK or TRACE commands.

The ovname argument specifies a named overlay breakpoint or tracepoint to clear.

The $ALL argument directs the debugger to clear all watchpoints, breakpoints, and tracepoints at all locations.

5.7 RUN USER PROGRAM (RUN)

The following command runs the user program in the AP.

```
Run [/count] [addexp]
```

The count modifier specifies the number of breakpoints or watchpoints to encounter before stopping. If it is omitted, execution stops at the first breakpoint encountered. The maximum value of count is 2**31 - 1. This number is always interpreted as a decimal number.

The addexp argument specifies the program location where execution starts. If it is omitted, program execution begins at the current program source address.

The debugger starts the program at the specified location and then waits until the program stops. After execution, the debugger displays the current program location and the total elapsed execution time since the last RUN or STEP command which specified a start address. Also, the command buffer is executed if it is not disabled by the BUFFER command (described in Section 5.16) and is not empty. Otherwise, the debugger displays the contents of the currently open register or memory location.

Example:

- Set a breakpoint at program location 16 and then start program execution at location 10.

```
*B 16
*R 10
 PROG BREAK AT PSA = 17 ;  1.167 US.
*
```

  The program executed for 1.167 us and stopped with location 17 as the next instruction to be executed.

5.8 STOP USER PROGRAM (ATTENTION KEY)

The host-dependent attention key (described in the appropriate host manual, listed in Table 1-1) interrupts the user program started by the RUN or STEP command. The user can then examine the machine state, make modifications, and continue execution (by using the RUN or STEP command).

NOTE

This feature of the attention key applies only during execution of a RUN command or a STEP command. Using this key at any other time can abort the debugger.

5.9 SET OR LIST PROGRAM TIME-OUT PERIOD (PTIMEOUT)

The PTIMEOUT command sets or lists the time-out period after which the debugger interrupts the user program running in the AP and checks whether the user attention key was pressed. If the key was pressed, control returns to the user for input. If not, the debugger restarts the user program in the AP.

To set or list the program time-out value, enter the following command:

PTImeout [=value]

The value argument must be an integer expression indicating the number of seconds in the time-out period. The smallest legal time-out period is one second. The default time-out period is 10 seconds. A value of 0 indicates that no time-out period is set.

If no argument is entered with the command, the debugger displays the current time-out value.

5.10 SINGLE-STEP THROUGH USER PROGRAM (STEP)

The STEP command is useful when proceeding step-by-step through a piece of code while watching for a particular problem to occur.

The following command single-steps through an AP program:

Step [/count] [/P] [/option] [addexp]

The count modifier specifies the number of instructions to execute in step mode. The maximum value of count is 2**31 - 1. This number is always interpreted as a decimal number.

The P (proceed) modifier specifies step-and-proceed. The program executes in step mode until a breakpoint is encountered. If both P and count are specified, the program executes step-by-step until it hits a breakpoint or until the specified number of instructions are executed, whichever comes first.

The option modifier can have the value OVER (indicating that the execution of a called procedure is considered as a single step) or INTO (indicating that the execution of each statement in the procedure is a step). The default is OVER.

The addexp argument specifies the program location where execution starts. If it is omitted, program execution begins at the current program source address.

After execution of each program step, the debugger displays the current program location and the total elapsed execution time since the last RUN or STEP command which specified a start address. With APDEBUG64 (but not APSIM64), this elapsed time is inflated by six or seven cycles because of system overhead. Also, the command buffer is executed if it is not disabled by the BUFFER command (described in Section 5.7) and is not empty. Otherwise, the debugger displays the contents of the currently open register or memory location.

In APAL64 mode, a step is a single instruction.

In FTN mode, a step is a single source statement. The user must select the LINennum compiler option (included in the DEBUG option) at compile time in order to use the STEP command while debugging in FTN mode. (The APFTN64 User's Guide (listed in Table 1-1) contains a complete discussion of the compiler options.)

Examples:

- Examine MA. Then watch it change as the program is single-stepped starting at location 10.

```
*E MA
$MA = 103
*S 10
 PROG STEP AT PSA = 11  0.167 US.
$MA = 104
*S
 PROG STEP AT PSA = 12  0.333 US.
$MA = 105
*S
 PROG STEP AT PSA = 13  0.500 US.
$MA = 106
*
```

- Set a breakpoint at PS location 12. Examine MA. Then step-and-proceed through the program until it hits the breakpoint.

```
*B 12
*E MA
$MA = 103
*S/P
 PROG STEP AT PSA = 11  0.167 US.
$MA = 104
 PROG STEP AT PSA = 12  0.333 US.
$MA = 105
 PROG BREAK AT PSA = 13 ;  0.500 US.
$MA = 106
*
```

5.11 PRINT CURRENT LOCATION (WHERE)

The following command prints the current program location:

```
WHere
```

Example:

```
*WH
AT PSA = 0
```

5.12 EXECUTE A SUBROUTINE OR FUNCTION (INVOKE)

The following command executes a subroutine or function:

```
INVoke name [(arg [,arg...])]
```

The name argument specifies the subroutine or function name.

The arg arguments are the actual arguments to the subroutine. They can be address expressions (addexp) or constants.

5.13 PRINT ELAPSED EXECUTION TIME (TIME)

The TIME command displays the elapsed program execution time (in microseconds) since the last RUN or STEP command which specified a starting location. The time is always printed in decimal notation. Note that execution times are variable for on-line debugging due to I/O interrupts and DMA (direct memory access) activity.

The following command displays elapsed AP program execution time:

```
TIme
```

Example:

- Print the elapsed program execution time.

```
*TI
 14231324.667 US.
*
```

5.14 PRINT OVERLAY STRUCTURE DIAGRAM (POVERLAY)

The POVERLAY command displays the overlay structure and memory residency status of each overlay segment. Indentation indicates the hierarchy of overlays; subordinate overlays are indented beneath their superior. The multiple overlay notation appears when the subordinate overlays can occupy the same locations in memory. In addition to the memory residency status, the POVERLAY command also indicates the memory bounds (base and size) of each overlay area.

The following command displays the current overlay structure:

```
POverlay
```

Figure 5-1 shows an overlay structure diagram.

```
OVERLAY NAME        (R=RESIDENT, M=MODIFIED)   PSBASE   PSSIZE   MDBASE   MDSIZE
OV1                                             4CEE       5F       98F       3F
OV2                                        R    4CEE       3C       98F        0
(
. OV3                                           4D2A       39       98F       1E
. OV4                                           4D2A       23       98F       1E
)
```

-3392-

Figure 5-1 Overlay Structure Diagram

5.15 PRINT A PROCEDURE CALL TRACEBACK (PTRACE)

A traceback displays the procedure calls which resulted in execution of the current routine. In FTN mode, the source line number at which the call was made is also displayed.

The following command displays a procedure call traceback:

```
Ptrace
```

Example:

```
*P
 IN   'FABS  ' LOC 1102
 FROM 'FOROTS' LOC 571
 FROM 'SUB7  ' AT LINE 13  LOC 621
 CALLED FROM HOST AT LINE 57  LOC 310
*
```

5.16 FILL COMMAND BUFFER (FBUF)

A command buffer saves the user entering individual commands to manipulate memories and registers at each breakpoint, tracepoint, and watchpoint.

The following command fills the command buffer:

FBuf [/CL] ['file']

The CL modifier directs the debugger to close the command buffer.

The file argument specifies the file to be loaded into the command buffer. If it is omitted, all subsequent commands from the terminal are entered into the command buffer until it is closed.

More than one file can be put into the command buffer before it is closed. Depending on the settings of the command buffer switches (described in Section 5.17), the command buffer is executed after each RUN or STEP command. The command buffer can also be executed with the XBUF command. If the command buffer contains a RUN or STEP command, the command buffer continues execution from the point after the embedded RUN or STEP command.

The command buffer cannot contain an FBUF command. It can contain an XBUF command only if the XBUF command executes an external command file.

Note that after the FBUF command is entered, the debugger prompts with "FBUF=" instead of an asterisk. The "FBUF=" prompt remains until the buffer is closed with the /CL modifier.

Example:

- Fill the command buffer with commands to examine the address registers.

```
*FB
FBUF=E MA
FBUF=E PSA
FBUF=E TMA
FBUF=E DPA
FBUF=E DA
FBUF=E SRA
FBUF=FB/CL
*
```

5.17 TOGGLE COMMAND BUFFER SWITCHES (BUFFER)

To toggle the command buffer switches, enter one of the following:

```
BUffer [/option] [$STEP] [$BREAK] [$TRACE] [$WATCH]

BUffer [/option] $ALL
```

The option modifier can have the value ON (indicating that the command buffer is to be executed) or OFF (indicating that it is not to be executed). The default value is ON.

The $STEP argument specifies that the command applies to each program step (when using the STEP command).

The $BREAK, $TRACE, and $WATCH arguments specify that the command applies to a breakpoint, tracepoint, or watchpoint, respectively.

The $ALL argument specifies that the command applies to all of the above cases ($STEP, $BREAK, $TRACE, and $WATCH).

If no argument appears with the command, the current settings of the command buffer switches and the current contents of the command buffer are displayed.

5.18 EXECUTE THE COMMAND BUFFER (XBUF)

The following command executes commands from the command buffer:

```
XBuf [/count] ['file']
```

The count modifier specifies the number of times to execute the file or command buffer. If the modifier is omitted, count is assumed to be 1. The count modifier is interpreted as a decimal number.

The file argument directs the debugger to execute commands from the specified file rather than from the internal command buffer.

APDEBUG64 executes all the commands in the buffer or file before returning to the terminal for input. If any fatal command errors occur during the execution of the command buffer or a command file, the debugger returns to the terminal for further input.

Example:

- Execute the command buffer filled in the example in Section 5.16.

```
*XB
$MA = 104
$PSA = 23
$TMA = 1
$DPA = 3
$DA = 0
$SRA = 1
*
```

5.19 OPEN OR CLOSE A LISTING FILE (LIST)

Whenever a listing file is open, all subsequent terminal input and output produced by the debugger is written to the listing file, providing the user with a permanent copy of a debugging session. (This command is helpful when debugging must be done as a host batch job.)

The following command opens or closes a listing file:

```
LIst [/CL] ['file']
```

The CL modifier directs the debugger to close the currently open listing file.

The file argument specifies a listing file to be opened for output. The debugger closes any previously opened listing file before the file is opened. The file name must be enclosed in apostrophes.

If the modifier and argument are omitted, the debugger displays the name of the currently open listing file.

5.20 PRINT THE DEBUGGER STATUS (STATUS)

The following command displays the debugger status:

```
STAtus
```

The status displayed contains the following information:

- current breakpoints, tracepoints, and watchpoints
- current symbol table open
- current program location
- current language mode
- current default radix (and formats if in APAL64 mode)
- current listing file
- current MASK value (refer to Section 6.6)

CHAPTER 6

ACCESSING REGISTER AND MEMORY LOCATIONS

6.1 INTRODUCTION

This chapter describes the commands that affect registers and memory locations. The following commands open, examine, and modify registers and memory locations:

Table 6-1 Register and Memory Location Commands

COMMAND	MINIMUM ABBREVIATION	FUNCTION
EXAMINE	E	open and examine register or memory location
+	+	examine succeeding memory location
-	-	examine preceding memory location
DISPLAY	DI	display the value of an expression
MASK	M	set mask to be used in SEARCH command
SEARCH	SE	search memory for a specified value
DEPOSIT	D	deposit into register or memory location
ZERO	Z	zero a specified memory range

6.2 MEMORY AND REGISTER LOCATION PARAMETERS

This section describes the parameters used with the commands described in this chapter to access AP memory and register locations.

6.2.1 AP Functional Units

The AP contains two kinds of functional units accessible to the debugger: memories and registers. The next two sections define memory and register command parameters.

6.2.1.1 Memory Specifiers (mem)

Whenever mem appears as a command argument, it is a memory specifier. A memory specifier is a mnemonic representing the desired memory. Table 6-2 lists the valid memory mnemonics.

Table 6-2 Memory Mnemonics

MNEMONIC	NAME	WIDTH	MEMORY SIZE
PS	program source memory space	64	job dependent
MD	main memory space	64	job dependent
TM	table memory	64	host dependent
DPX	data pad X	64	32
DPY	data pad Y	64	32
SP	SPAD	32	64
SRS	subroutine return stack	32	256
IO	I/O devices	64	2**16 (max.)

6.2.1.2 Register Specifiers (reg)

Whenever reg appears as a command argument, it is a register specifier. A register specifier is a mnemonic representing the desired register. Table 6-3 lists the valid register mnemonics.

Table 6-3 Register Mnemonics

MNEMONIC	NAME	WIDTH
PSA	program source address	24
MA	main memory address	24
TMA	table memory address	24
DPA	data pad address	5
SRA	subroutine stack address	8
SPFN	SPAD function	32
FA	adder output	64
FAS	adder output exceptions	6
FA1	adder next output	64
FA1S	adder next output exceptions	6
FM	multiplier output	64
FMS	multiplier output exceptions	4
FM1	multiplier next output	64
FM1S	multiplier next output exceptions	4
FM2	multiplier 2nd next output	64
FM2S	multiplier 2nd next output exceptions	4
MDR	main memory output	64
MDR1	main memory next output	64
MDR2	main memory 2nd next output	64
TMR	table memory output	64
APSTAT	user status	32
APSTAT3	state save status	32
MDBRK	MD breakpoint	32
PSBRK0	PS breakpoint 0	32
PSBRK1	PS breakpoint 1	32

6.2.2 Field Specifiers (field)

Whenever field appears as a command argument, it is a field specifier. A field specifier indicates that the command applies only to the indicated bits within the word rather than to the entire word. A field specifier can be either a register or memory field mnemonic of the form,

<mnemonic>

or an expression of the form,

<left:right>

where left and right are integers specifying the bit numbers of the left-most and right-most bits in the field. Numbers in a field specifier are decimal unless an explicit radix is specified. Bit 0 is the left-most bit. (Appendix A lists the valid register and memory field mnemonics.)

Field specifiers should not be used with address ranges involving character data.

Note that angle brackets (< >) always enclose field specifiers (e.g., <field>).

6.2.3 Address Range (addrange)

Whenever addrange appears as a command argument, it is an address range. An address range has the form:

from [:to [:by]]

The from field specifies the start of the address range (e.g., PS(1)), and the to field specifies the end (e.g., PS(10)). If omitted, the end address is assumed to be the same as the start address. The to field must be greater than the from field. The from and to fields must be address expressions (addexp) of compatible types. The address range takes the type of the from and to expressions. The by field specifies the address increment of the memory range. If omitted, it is assumed to be 1. The by field must be an integer expression. The to and by fields are not allowed if the from field is a character substring expression.

6.2.4 Value Expressions (value)

Whenever value appears as a command argument, it is a value expression.

The next two sections define APAL64 and FTN value expressions.

6.2.4.1 APAL64 Value Expressions

The value of the symbol itself is always used when evaluating a value expression in APAL64 mode. The special AP register names allowed are the same as those allowed for APAL64 address expressions (listed in Section 5.2.2). Value expressions are untyped. In APAL64 mode, a value expression is any of the following:

- any valid APAL64 expression
- a valid APAL64 floating-point constant
- an ASCII constant
- an integer constant in byte format (described in Section 4.5)

6.2.4.2 FTN Value Expressions

When evaluating a value expression in FTN mode, the value of the symbol is the contents of the location pointed to by the symbol (except untyped symbols whose immediate values are always used). In FTN mode, a value expression is any valid APFTN64 expression except that function references are not allowed within the expression.

Enter floating-point constants carefully, since some valid hexadecimal floating-point constants look like integer expressions. Therefore, the decimal point must always be used. For example, 25E+30 must be entered as 25.E+30.

6.2.5 Relational Expressions (relexp)

Whenever relexp appears as a command argument, it is a relational expression. A relational expression has the form:

value relop value

where value is a value expression and relop is the relational operator.

Table 6-4 lists the valid relational operators.

Table 6-4 Relational Operators

APAL64 MODE	FTN MODE	MEANING
=	.EQ.	equal to
<>	.NE.	not equal to
<	.LT.	less than
<=	.LE.	less than or equal to
>	.GT.	greater than
>=	.GE.	greater than or equal to

6.3 OPEN AND EXAMINE A REGISTER OR MEMORY LOCATION (EXAMINE)

To open and examine a register or memory location, enter one of the following:

Examine [/format] [/radix] [addrange] [<field>] ['file']

Examine [/format] [/radix] [mem] [<field>] ['file']

Examine [/format] [/radix] [reg] [<field>] ['file']

Examine [/format] [/radix] [value]

The addrange argument specifies the location to examine. If the address range contains character data, the range is examined one datum at a time.

The mem argument specifies that the last open location in that memory is examined.

The reg argument specifies the register to examine.

If the address range, memory, and register specifiers are all omitted, the debugger displays the last open location.

The file argument directs the debugger to write to the specified file rather than to the terminal. A file written to with the EXAMINE command can be read back into the AP register or memory with the DEPOSIT command. If no file is specified, output goes to the terminal. If the 'file' specifier is used, the first line written by the debugger is a header line describing the data. This line contains all the information needed to read the file back in with the DEPOSIT command. The format of this header line follows the format of the EXAMINE and DEPOSIT commands (without the command name itself). The header line is omitted if the output is going to the terminal.

The value argument specifies a symbol to examine. In APAL64 mode, the value of the symbol is displayed. In FTN mode, the value in the location pointed to by the symbol is displayed.

NOTE

A symbol with a constant value, defined in an APFTN64 PARAMETER statement, has no associated location and hence cannot be examined with the Examine command. To display the value of a constant symbol, use the Display command, described in Section 6.5.

Examples:

- Examine the fifth element of array AR.

```
*E AR(5)
AR(5) = 25.3
*
```

  The array element contains 25.3.

- Examine the memory address register in decimal. Note that a dollar sign ($) precedes any predefined memory and register names on output to distinguish them from any user symbol names with the same name.

```
*E/D MA
$MA = 1376
*
```

  MA contains 1376 (decimal).

- Examine data pad X registers 0, 2, 4, and 6.

```
*E DPX(0):DPX(6):2
$DPX(0) = -1.0
$DPX(2) = -1.0
$DPX(4) = -1.0
$DPX(6) = -1.0
*
```

  DPX registers 0, 2, 4, and 6 all contain -1.0.

- Examine the three MD locations beginning at MA.

```
*E @MA:  @MA+2
$MD(235) = 1.2
$MD(236) = 0.0
$MD(237) = -2.3
```

  MD location 235 contains 1.2, MD location 236 contains 0.0, and MD location 237 contains -2.3.

6.4 EXAMINE A SUCCEEDING (+) OR PRECEDING (-) MEMORY LOCATION

To open and examine a memory location before or after the currently open location, enter one of the following:

+[offset] [/format] [/radix] [mem] [<field>]

-[offset] [/format] [/radix] [mem] [<field>]

The + and - specify that the memory location to examine is after (+) or before (-) the currently open memory location.

The offset specifier indicates a value to add to (+) or subtract from (-) the address of the currently open memory location to obtain the address of the memory location to examine. A value of 1 is used if the offset specifier is omitted.

The mem argument indicates which memory to examine. The offset specifier is relative to the last opened location in the specified memory. The currently open memory is examined if no memory is specified.

Note that these commands are used with memory locations only; they are not used with register locations.

Examples:

- Examine main memory locations 23 and 24.

```
*E MD(23)
$MD(23) = -234.0
*+
$MD(24) = 789.0
*
```

  MD location 23 contains -234.0, and MD location 24 contains 789.0. The current location is MD(24).

- Examine SPAD registers 7 and 5.

```
*E SP(7)
$SP(7) = 200
*-2
$SP(5) = 100
*
```

  SPAD register 7 contains 200, and SPAD register 5 contains 100.

6.5 DISPLAY THE VALUE OF AN EXPRESSION (DISPLAY)

The following command displays the value of a value expression:

DIsplay [/format] [/radix] = value

This command evaluates and displays the value expression in the format and radix specified by format and radix.

Example:

- Display the value of the expression A+3, where A is a local variable whose address is MD(52) and whose content is 17.

```
*LANG $FTN
*DI = A+3
 = 20
*LANG $APAL
*DI = A+3
 = 55
*
```

6.6 SEARCH A MEMORY (SEARCH)

To search for all occurrences of a specified value in a group of memory locations, enter one of the following:

SEarch [/radix] [/format] mem [<field>] = value

SEarch [/radix] [/format] [addrange] [<field>] = value

The entire memory is searched if the mem argument is used.

The value argument specifies the search value.

The addrange argument specifies the location to search. If addrange is not entered, the currently open memory is searched.

If the SEARCH value is a character string and the address range specified is a character data range, then each character datum is searched for the first occurrence of the value. When the value is found, the location and its contents are displayed.

If the address range is not a character data range, then each memory word is searched for the value (truncated to eight characters). If a field specifier is used, only the specified field of each memory word is searched for the value. When the value is found, the memory location and its contents are displayed.

NOTE

The search mask (described in Section 6.7) can be used to isolate non-contiguous fields for comparison with the search value. If a mask has been used in a previous search command, be sure that the mask is reset to its default value before executing the next search command.

Example:

- Locate all occurrences of the value -1 in the SPAD registers.

```
*SE SP = -1
$SP(3) = -1
$SP(7) = -1
$SP(12) = -1
*
```

SPAD registers 3, 7, and 12 all contain -1.

6.7 SET SEARCH MASK (MASK)

The following command sets the mask used in future SEARCH commands:

Mask [= value]

The search mask is used to isolate non-contiguous fields for comparison with the search value.

The value argument specifies the value used as a mask to select fields in the memory word in any future SEARCH command. (For a description of the SEARCH command, refer to Section 6.6.) The mask value is logically ANDed with each memory location searched, to produce a value for comparison with the search value. The comparison value is the value of the fields masked with ones. Fields masked with zeros are filled with zeros in the comparison value.

The mask value is right-justified: it is not necessary to enter leading zeros.

Figure 6-1 illustrates the use of the search mask.

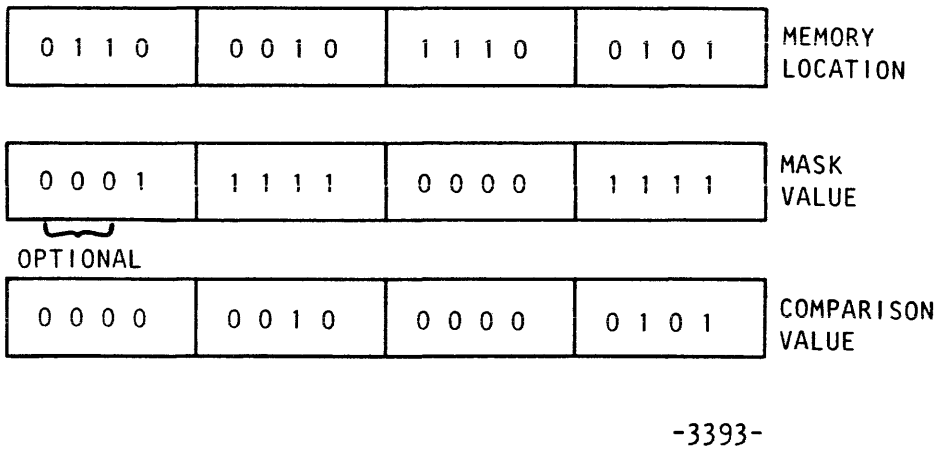

Figure 6-1 The Search Mask

The default mask is all ones, selecting the entire memory location for the comparison value. If the value argument is omitted, the current value of the mask is displayed.

Example:

- Search SPAD for even numbers by setting the mask to 1.

```
*MA = 1
*SE SP = 0
$SP(1) = 0
$SP(5) = 212
$SP(31) = 4
*
```

SPAD registers 1, 5, and 31 contain even numbers.

6.8 DEPOSIT INTO A REGISTER OR MEMORY (DEPOSIT)

To deposit (write) into a register or memory location, enter one of the following:

Deposit [/format] [/radix] [addrange] [<field>] = value

Deposit [/format] [/radix] [addrange] [<field>] ['file']

Deposit [/format] [/radix] [reg] [<field>] = value

Deposit [/format] [/radix] [reg] [<field>] ['file']

Deposit [/format] [/radix] [mem] [<field>] = value

Deposit [/format] [/radix] [mem] [<field>] ['file']

The format and radix arguments specify the format and the radix of the input value(s). If not specified, the debugger uses either the values specified in the header line (if from a file) or the global defaults. Specify the format argument only when the input value is in byte format and the format for the functional unit being deposited is not defined as a byte format (described in Section 4.5).

The addrange argument specifies the location where the file or value is deposited. If the address range is not entered, the file or value is deposited into the last open memory.

If the mem argument is used, the last open location in that memory is deposited into.

The value argument specifies the value deposited into the specified location. If the value is a character datum, the debugger truncates the value or extends the value with blanks to fit into the address range.

The file argument indicates that the values deposited are taken from the specified file. This argument makes it possible to deposit different values in each specified location in the address range. If the register or memory or any of the from or by fields from the address range are not specified in the DEPOSIT command, the appropriate information is taken from the header line of the file. The format of this header line is described in the EXAMINE command. Adding the number of locations in the range (as specified by the from and to information in the file) to the specified or default from field calculates the default to field. The radix and format information is also read from the file.

If the user is setting up the file, a non-blank header line must be included. The user can specify the radix, format, and address range of the input or specify a minimum amount of information, such as the radix, using the syntax of the DEPOSIT command without the command name.

If a minimum amount of information is specified, the address range must be specified in the DEPOSIT command.

If the value and file arguments are both omitted, the input values are read from the terminal. If the file is the terminal, the default radix and format information is taken from the current defaults. However, no defaults are assumed for the address range. No header line is read from the terminal.

Examples:

- Deposit the values contained in the file DPVAL into DPX and DPY. The locations in DPX and DPY in which to deposit the data are taken from DPVAL's header line.

```
*D DPX 'DPVAL'
*D DPY 'DPVAL'
*E DPX(1):DPX(5):2
$DPX(1) = 1.0
$DPX(3) = 2.0
$DPX(5) = 3.0
*E DPY(1):DPY(5):2
$DPY(1) = 1.0
$DPY(3) = 2.0
$DPY(5) = 3.0
*
```

  The file DPVAL contains the following:

```
DPX(1):DPX(5):2
DPX(1) = 1.0
DPX(3) = 2.0
DPX(5) = 3.0
```

- Deposit the first two values in the file DATA into SPAD(0) and SPAD(1).

```
*D SP(0):SP(1) 'DATA'
*E SP(0):SP(1)
$SP(0) = 0F21
$SP(1) = 3ECA
*
```

The file DATA contains the following:

```
/Z
0F21
3ECA
```

- Deposit -1.0 into main memory locations 1000 through 1005.

```
*D MD(1000):MD(1005) = -1.0
*E MD(1000):MD(1005)
$MD(1000) = -1.0
$MD(1001) = -1.0
$MD(1002) = -1.0
$MD(1003) = -1.0
$MD(1004) = -1.0
$MD(1005) = -1.0
*
```

MD locations 1000 through 1005 now contain -1.0.

- Deposit 0.0 into data pad Y registers 1, 4, and 7.

```
*D DPY(1):DPY(7):3 = 0.0
*E DPY(1):DPY(7):3
$DPY(1) = 0.0
$DPY(4) = 0.0
$DPY(7) = 0.0
*
```

Data pad Y registers 1, 4, and 7 now contain 0.0.

- Deposit -1 into the diagonal of a 10x10 array, MIN.

```
*D MIN(1,1):MIN(10,10):11 = -1
*
```

6.9 ZERO THE AP (ZERO)

The following commands fill a specified memory range with zeros:

Zero addrange [<field>]

Zero mem [<field>]

The addrange argument identifies the memory range to be zeroed.

The mem argument identifies an entire memory to be zeroed.

APPENDIX A

FIELD MNEMONICS

A.1 PROGRAM SOURCE FIELD MNEMONICS

The following tables list the valid program source (PS) memory field mnemonics.

Table A-1 PS SPAD Field Mnemonics

MNEMONIC	NAME	WIDTH
SOP	SPAD double-operand operation	3
SOP1	SPAD single operand operation	4
SH	SPAD shift	2
SPS	SPAD source address	4
SPSX	SPAD source address extension	2
SPD	SPAD destination address	4
SPDX	SPAD double-operand destination address extension	2
SPDX1	SPAD single operand destination address extension	2

Table A-2 PS Adder Field Mnemonics

MNEMONIC	NAME	WIDTH
FADD	adder double-operand operation	4
FADD1	adder single operand floating operation	3
IFADD1	adder single operand integer operation	3
A1	adder operand 1	3
A2	adder operand 2	3

Table A-3 PS Branch Field Mnemonics

MNEMONIC	NAME	WIDTH
COND	branch condition	4
DISP	branch displacement	5

Table A-4 PS Data Pad Field Mnemonics

MNEMONIC	NAME	WIDTH
DPX	data pad X operand	2
DPY	data pad Y operand	2
DPBS	data pad bus operand	3
XR	data pad X read index	3
YR	data pad Y read index	3
XW	data pad X write index	3
YW	data pad Y write index	3
XRXE	data pad X XW read index extension	1
YRXE	data pad Y XW read index extension	1
YWXE	data pad Y XW write index extension	1
XRYE	data pad X YW read index extension	1
YRYE	data pad Y YW read index extension	1
XWYE	data pad X YW write index extension	1

Table A-5 PS Multiplier Field Mnemonics

MNEMONIC	NAME	WIDTH
FM	multiplier double-operand operation	1
FM1	multiplier single-operand operation	2
FM0	multiplier zero-operand operation	2
M1	multiplier operand 1	2
M2	multiplier operand 2	2

Table A-6 PS Memory Field Mnemonics

MNEMONIC	NAME	WIDTH
MI	main memory input operation	2
MA	main memory address operation	2
DPA	data pad address operation	2
TMA	table memory address operation	2

Table A-7 PS Immediate Value Field Mnemonics

MNEMONIC	NAME	WIDTH
SVAL	8-bit immediate value	8
VALUE	24-bit immediate value	24
HVAL	32-bit immediate value	32
SVALNL	SVAL SPAD load inhibit	1

Table A-8 PS Special Operation Field Mnemonics

MNEMONIC	NAME	WIDTH
SPEC	special operation	4
STEST	conditional branches	4
SPECWR	partial word writes and duplicates	4
SPECFP	round and normalize inhibits	4
SPECIN	mode switching and state save	4
SETPSA	JMP's, JSR's, and read/write PS	4
STJMP	conditional JMP's	4
STJSR	conditional JSR's	4

Table A-9 PS I/O Operation Field Mnemonics

MNEMONIC	NAME	WIDTH
IO	I/O operation	3
LDREG	load address registers	3
RDREG	read address registers	3
IOWRT	partial word writes	3
IOMEM	miscellaneous	3
INOUT	I/O data transmission	3
FLAG	set/clear program flags	3
CONTRL	privileged control	3

A.2 REGISTER FIELD MNEMONICS

The following tables list the valid register field mnemonics.

Table A-10 User Status Register Field Mnemonics

MNEMONIC	NAME	WIDTH
SRAU	subroutine return stack underflow	1
FL0	program flag 0	1
FL1	program flag 1	1
FL2	program flag 2	1
FL3	program flag 3	1
NRSQRT	negative reciprocal square root	1
BITR	bit-reverse shift count	4
IOVF	SPAD integer overflow	1
F53OVF	FA/FM 53-bit integer overflow	1
F32OVF	FA/FM 32-bit integer overflow	1
OVF	FA/FM floating-point overflow	1
UNF	FA/FM floating-point underflow	1
DIVZ	reciprocal of zero	1
FZ	adder output zero	1
FN	adder output negative	1
Z	SPAD function zero	1
N	SPAD function negative	1
C	SPAD carry	1
SRAO	subroutine return stack overflow	1
IFFT	inverse FFT addressing mode	1
FFT	FFT addressing mode	1

Table A-11 State Save Status Register Field Mnemonics

MNEMONIC	NAME	WIDTH
TRAP	trap instruction indicator	1
DPM	DPBS mantissa bits 0-1	2
INTES	IOVF exception interrupt enable	1
INTE53	F53OVF exception interrupt enable	1
INTE32	F32OVF exception interrupt enable	1
INTEF	OVF/UNF exception interrupt enable	1
INTFS	SRAO/SRAU fatal interrupt enable	1
INTFD	DED fatal interrupt enable	1
INTFM	MPAR fatal interrupt enable	1
INTTB	breakpoint trap interrupt enable	1
INTTT	TRAP trap interrupt enable	1
INTTJ	JSYS trap interrupt enable	1
FMSTAT	multiplier exception status	4
FM53	multiplier F53OVF status	1
FM32	multiplier F32OVF status	1
FMOVF	multiplier OVF status	1
FMUNF	multiplier UNF status	1
FASTAT	adder exception status	6
FA53	adder F53OVF status	1
FA32	adder F32OVF status	1
FAOVF	adder OVF status	1
FAUNF	adder UNF status	1
FADIVZ	adder DIVZ status	1
FANRSQ	adder NRSQRT status	1
SRA	subroutine return stack address	8

Table A-12 Main Memory DED Error Register Field Mnemonics

MNEMONIC	NAME	WIDTH
CPUE	CPU main memory DED error	1
DMAE	DMA main memory DED error	1
PSCE	program source cache DED error	1
DEDH	DED halt enable	1
ECCDIS	error correcting disable	1
DEDADR	DED physical address	24

Table A-13 TM/PS Parity Error Register Field Mnemonics

MNEMONIC	NAME	WIDTH
TME	table memory parity error	1
PSE	program source parity error	1
TMH	TME halt enable	1
PSH	PSE halt enable	1
PERADR	parity error physical address	24

Table A-14 MD Breakpoint Register Field Mnemonics

MNEMONIC	NAME	WIDTH
MDBHIT	MD breakpoint hit	1
MDBREN	MD read breakpoint enable	1
MDBWEN	MD write breakpoint enable	1
MDBRD	MD read breakpoint	1
MDBWRT	MD write breakpoint	1
MDBADR	MD breakpoint address	24

Table A-15 PS Breakpoint Register Field Mnemonics

MNEMONIC	NAME	WIDTH
PSBHIT	PS breakpoint hit	1
PSBENB	PS breakpoint enable	1
PSBADR	PS breakpoint address	24

Table A-16 Data Word Field Mnemonics

MNEMONIC	NAME	WIDTH
LEFT	left half	32
RIGHT	right half	32
EXPON	exponent	11
MANTI	mantissa	53

APPENDIX B

APDEBUG64 INTERACTIVE COMMANDS

In Table B-1, the lowercase letters following the shortest legal abbreviation are optional.

Table B-1 APDEBUG64 Interactive Commands

COMMAND	FUNCTION	PAGE
+	examine succeeding memory location	6-8
-	examine preceding memory location	6-8
Break	set or list breakpoint	5-4
BUffer	toggle command buffer switches	5-16
Clear	clear break/trace/watch points	5-9
Deposit	deposit into register or memory location	6-12
DIsplay	display the value of an expression	6-9
Examine	open and examine register or memory location	6-6
FBuf	fill command buffer	5-15
FOrmat	set I/O format	4-5
INit	initialize the AP	3-6
INVoke	invoke a subroutine or function	5-13
LAnguage	set the language mode	4-1
LIst	open or close a listing file	5-17
LOad	load specified load module into the AP	3-6
Mask	set mask to be used in SEarch command	6-10
POverlay	print overlay structure diagram	5-14
PTImeout	set or list program timeout period	5-10
PTrace	print a procedure call traceback	5-14
Quit	exit APDEBUG64	3-6
RAdix	set integer radix	4-4
Run	run an AP program	5-9
SEarch	search memory for a specified value	6-9
STAtus	print debugger status	5-17
Step	single-step through program	5-11
SYmbol	open symbol table for symbolic input	4-3
TIme	print elapsed execution time	5-13
Trace	set or list tracepoint	5-7
Watch	set or list watchpoint	5-7
WHere	print current program location	5-13
XBuf	execute command buffer	5-16
Zero	zero specified registers and memories	6-14

APPENDIX C

ERROR MESSAGES

This appendix contains the debugger error messages along with an explanation of each error and suggested corrective action. The error messages are arranged alphabetically for quick reference.

APDEBUG64-F-ACONST, error in ASCII constant

Meaning: Incorrect syntax entered for ASCII constant.

Action: Enter correct ASCII constant syntax (refer to the APAL64 Programmer's Reference Manual).

APDEBUG64-F-ADDRANGE, negative address range

Meaning: The specified ending address is less than the starting address.

Action: Enter the command line with the correct starting and ending addresses.

APDEBUG64-F-AMBIGCMD, ambiguous command abbreviation

Meaning: More than one command begins with the specified abbreviation.

Action: Enter the command with a longer abbreviation.

APDEBUG64-F-APNUM, illegal AP number

Meaning: Only one AP number can be entered on the command line. A legal AP number is ≥ 0.

Action: Enter the command line with the correct AP number.

APDEBUG64-F-ARGOVF, table overflow - too many arguments and switches

Meaning: Command line contains too many arguments and switches.

Action: Use the LOAD, SYMBOL, and/or LIST commands to enter arguments.

APDEBUG64-F-BADFD, bad file descripter (internal error)

Meaning: Internal AP errors.

Action: Contact FPS Customer Service.

APDEBUG64-F-BADSYMFILE, bad symbol file

Meaning: The debugger found an error when reading the symbol file.

Action: Report error to FPS.

APDEBUG64-F-BRKEXISTS, breakpoint already exists as call/return

Meaning: The specified breakpoint already exists as a call/return breakpoint.

Action: Clear the breakpoint before trying to reset it (refer to Section 5.6).

APDEBUG64-F-BRKMEM, break or trace may be set at a program location only

Meaning: A breakpoint can be set at: a FORTRAN line number, FORTRAN statement label, APAL64 label, or PS location.

Action: Set the breakpoint at a valid location. Use watchpoints for data locations.

APDEBUG64-F-BYTEFMT, illegal byte format

Meaning: Byte values must be integers.

Action: Enter byte values correctly (refer to Section 4.5).

APDEBUG64-F-BYTEOVF, a maximum of 64 bytes may be specified

Meaning: Too many bytes specified in value expression on a command.

Action: Re-enter command, specifying 1 to 64 bytes.

APDEBUG64-F-BYTEVAL, byte values must be integer expressions

Meaning: Byte values must be integers.

Action: Enter byte values correctly (refer to Section 4.5).

APDEBUG64-F-CANTFINDSYM, need global symbols to output function value

Meaning: The user closed the symbol table before the program returned from the routine called by the INVOKE command. The returning function value is in DPX(0) and DPX(1).

Action: No action required.

APDEBUG64-F-CANTMIXCHAR, can't mix character and non-char values in input file

Meaning: An input data file cannot contain both character and non-character values for deposit.

Action: Separate character and non-character values into two data files.

APDEBUG64-F-CBUFFCMD, command entered not allowed in command buffer

Meaning: The specified command cannot be used in the command buffer.

Action: Do not enter the command in the command buffer.

APDEBUG64-F-CBUFFOVF, command buffer overflow, buffer closed

Meaning: The user tried to put commands into a full command buffer.

Action: Put a long series of commands into an external command buffer (refer to Section 5.16) and execute the command buffer with the XBUF command (refer to Section 5.18).

APDEBUG64-F-CHARFMT, character data may be output in ascii or integer only

Meaning: The user requested that character data be displayed in other than ASCII or integer format.

Action: Request character data in ASCII or integer format only (refer to Section 4.5).

APDEBUG64-F-CHARLENGTH, 0-length character address illegal

Meaning: The user tried to access character data of no length.

Action: Enter the corrected command line.

APDEBUG64-F-CHARMASK, field parameter not allowed with character data

Meaning: A command line cannot contain a field parameter with a character data variable.

Action: Use a substring expression (APFTN64 Reference Manual).

APDEBUG64-F-COMMAND, unrecognized command

Meaning: The debugger does not recognize the command entered.

Action: Check command syntax in this manual.

APDEBUG64-F-CONFLICT, conflicting parameters on command

Meaning: Conflicting modifiers or parameters were entered (e.g., both the /INTO and /OVER modifiers on the STEP command).

Action: Enter corrected command line.

APDEBUG64-F-COUNT, bad repetition count parameter

Meaning: The repetition count was specified as ≤ 0.

Action: Enter command line with correct repetition count (i.e., >0).

APDEBUG64-F-DIFOVERLAY, start and end addresses must be in same overlay

Meaning: The starting and ending addresses in an address range are not in the same overlay.

Action: Enter the command line with the starting and ending addresses in the same overlay.

APDEBUG64-F-DUMMYPARAM, parameter not accessible from current module

Meaning: A dummy parameter can be examined only when the program stops in the routine to which it is a parameter.

Action: No action required.

APDEBUG64-F-DUPPARAM, duplicate parameters on command

Meaning: A command parameter or modifier was entered more than once.

Action: Enter corrected command.

APDEBUG64-F-ENTRYDESC, cannot find entry descriptor

Meaning: The debugger cannot find the procedure entry descriptor (PED) corresponding to the last subroutine call (or JSR) made by the program. (Information in the PED is necessary to access an actual parameter to a subroutine or to STEP through an APFTN64 program.)

Action: For APAL64 users, the APAL64 Programmer's Guide contains subroutine linkage directives and procedure entry directives. (The compiler sets up PED's for APFTN64 programs.)

APDEBUG64-F-EOF, unexpected end of file

Meaning: The debugger came to the end of a load module, symbol file, or input file when it expected more information.

Action: Verify that the correct file was specified. If not, enter correct file name. If it was, report error to FPS Customer Service.

APDEBUG64-F-EXPACONST, ASCII constant not allowed in expressions

Meaning: An expression contained an ASCII constant.

Action: Enter an equivalent hexadecimal constant.

APDEBUG64-F-EXPEVAL, error in evaluating expression

Meaning: The debugger found an error when trying to evaluate an address or value expression.

Action: Verify that the expression is correct. If necessary, enter the command line with the correct expression syntax.

APDEBUG64-F-EXPRCONST, floating-point constant not allowed in APAL64 expressions

Meaning: An APAL64 expression contained a floating-point constant.

Action: Enter valid APAL64 expression (refer to the APAL64 Programmer's Reference Manual).

APDEBUG64-F-EXPTYPE, address expression types do not match

Meaning: An address expression is not compatible with its memory type.

Action: Remove the address expression type (using the INT operator discussed in Section 5.2.2) before using it in an expression of a different type.

APDEBUG64-F-FIELD, bad field specifier

Meaning: Field specifier formatted incorrectly.

Action: Enter field specifier correctly (refer to Section 6.2.2).

APDEBUG64-F-FIELDNAME, unrecognized field name

Meaning: The user specified an illegal field name.

Action: Enter a legal field mnemonic (Appendix A).

APDEBUG64-F-FILE, illegal file specification

Meaning: File name not enclosed in apostrophes.

Action: Enter file name with apostrophes (i.e., 'filename').

APDEBUG64-F-FILOPN, error in opening file

Meaning: The debugger could not open the specified file.

Action: Verify that the file exists. Check file protection. Enter corrected command ine.

APDEBUG64-F-FLDDELIM, '>' missing on field specifier

Meaning: Field specifiers must be enclosed in angled brackets.

Action: Enter field specifier enclosed in angled brackets (refer to Section 6.2.2).

APDEBUG64-F-FLOAT, bad format for floating point number

Meaning: Arithmetic error in evaluating an expression.

Action: Enter corrected command line.

APDEBUG64-F-ICONST, integer constant expected after command modifier

Meaning: The debugger expected an integer constant as part of a command modifier.

Action: Enter the command line with the correct command modifier.

APDEBUG64-F-IFOPTLENGTH, no room for IF expression

Meaning: The relational expressions specified on the BREAK, TRACE, or WATCH command modifiers are too long.

Action: Shorten relational expressions.

APDEBUG64-F-ILLADDRESS, illegal address expression

Meaning: The user entered an invalid address expression.

Action: Enter the correct address expression (refer to Section 5.2) on the command line.

APDEBUG64-F-ILLARG, illegal routine argument

Meaning: When calling a routine with the INVOKE command, only an address expression or value expression can be passed as an argument.

Action: Enter corrected command.

APDEBUG64-F-ILLARITH, illegal arithmetic operand

Meaning: Internal error.

Action: Report error to FPS Customer Service.

APDEBUG64-F-ILLCHAR, illegal character

Meaning: The debugger does not recognize a character on the command line.

Action: Enter command line with correct characters.

APDEBUG64-F-ILLENTRY, illegal entry point

Meaning: The debugger does not recognize the name the user entered with the INVOKE command as a valid entry point name.

Action: Examine APLINK64 map and enter corrected command line.

APDEBUG64-F-ILLHEADER, illegal file header

Meaning: The first line of a data file to be read with the DEPOSIT command is incorrect or contains insufficient information.

Action: Correct the data file header line (refer to Section 6.8).

APDEBUG64-F-ILLLOGICAL, illegal logical expression operand

Meaning: Internal error.

Action: Report error to FPS.

APDEBUG64-F-ILLPARAM, illegal parameter on command

Meaning: A parameter used on the command line is not valid for this command.

Action: Enter command line with valid parameter.

APDEBUG64-F-ILLRELATIONAL, illegal relational expression operand

Meaning: Internal error.

Action: Report to FPS.

APDEBUG64-F-ILLRELEXP, illegal relational expression

Meaning: The debugger found an error when evaluating a relational expression in a command line or assigned to a breakpoint, tracepoint, or watchpoint.

Action: Reset the breakpoint, tracepoint, or watchpoint with the corrected expression, or reenter the command.

APDEBUG64-F-ILLSTRING, illegal string

Meaning: While scanning a string constant in the input line, the debugger came to the end of the line before finding the ending apostrophe.

Action: Enter the command line with the correct syntax (APFTN64 Reference Manual).

APDEBUG64-F-ILLSYNTAX, illegal syntax

Meaning: The debugger does not recognize a command line construct.

Action: Evaluate command line and enter correctly.

APDEBUG64-F-INCOMPADD, incompatible address expressions

Meaning: The user specified starting and ending addresses that are not the same memory type.

Action: Enter compatible starting and ending addresses.

APDEBUG64-F-INCOMPATIBLE, incompatible arithmetic expression operands

Meaning: The arithmetic expression contains operands which are not compatible (e.g., a double-precision number with a complex number).

Action: Enter the command line with the corrected arithmetic expression (refer to the APFTN64 Reference Manual).

APDEBUG64-F-INCOMPVAL, value incompatible with address range

Meaning: Only integer and character values are compatible with character addresses.

Action: Use only integer or character values when depositing or searching character data.

APDEBUG64-F-INCREMENT, error in increment value

Meaning: The address increment must be a positive integer value.

Action: Enter corrected command line.

APDEBUG64-F-INPUTOVF, input source stack overflow

Meaning: The debugger external command files are nested too deeply.

Action: Do not nest external command files beyond 23 levels.

APDEBUG64-F-INPUTUNF, input source stack underflow

Meaning: The debugger found an end-of-file in the initial APDEBUG64 input source (usually the terminal).

Action: If executing the debugger from within a command file, verify that a QUIT command is entered before the end of the file.

APDEBUG64-F-INTEGER, bad format for integer number

Meaning: The user entered an invalid token where the debugger expected an integer.

Action: Enter corrected command line.

APDEBUG64-F-INTERNAL, internal error

Meaning: Internal error.

Action: Report error to FPS.

APDEBUG64-F-INVOKE, not yet returned from previously invoked routine

Meaning: The user entered an INVOKE command before the program returned from a previous INVOKE command.

Action: No action required.

APDEBUG64-F-INVOKECHAR, can't invoke character function of passed length

Meaning: The INVOKE command cannot be used to call a character function whose length is passed as a parameter.

Action: No action required.

APDEBUG64-F-LDMOVOVF, too many load modules specified on command

Meaning: Internal table overflow.

Action: Use the LOAD command to load the load modules.

APDEBUG64-F-LMBUFFER, buffer too small for load module block size

Meaning: The load module's block size is too big.

Action: Close symbol file or relink using a smaller block size.

APDEBUG64-F-LPAREN, left parenthesis missing in expression

Meaning: The left parenthesis is missing in the expression.

Action: Enter the corrected command line.

APDEBUG64-F-MAXBRK, maximum number of breakpoints exceeded

Meaning: Only two breakpoints or tracepoints are allowed at one time.

Action: Delete any unneeded breakpoints or tracepoints before setting another.

APDEBUG64-F-MISSPARAM, missing parameter on command

Meaning: The command line does not contain a necessary parameter.

Action: Enter the command line with valid command syntax.

APDEBUG64-F-MODIFIER, unrecognized command modifier

Meaning: The debugger does not recognize the command modifier following '/'.

Action: Enter the correct command modifier.

APDEBUG64-F-MODNAME, unknown module name

Meaning: The specified module name is not in the global symbol table.

Action: Verify that the correct symbol file is open and enter the command line with the correct module name.

APDEBUG64-F-NOARGS, incorrect number of arguments

Meaning: The user did not specify the correct number of arguments for the routine specified with the INVOKE command.

Action: Enter the INVOKE command (refer to Section 5.12) with the correct number of arguments.

APDEBUG64-F-NOARGSPPACE, no user space for arguments

Meaning: There is no MD space left to store argument pointers for the INVOKE command.

Action: Do not use the INVOKE command.

APDEBUG64-F-NOBRK, no breakpoint set at specified address

Meaning: The user tried to clear a breakpoint where none was set.

Action: Use the BREAK, TRACE, WATCH, and/or VFP command to list the current breakpoints, tracepoints, and watchpoints.

APDEBUG64-F-NOBYTEFMT, no byte format has been specified

Meaning: The user tried to deposit a value or search for a value entered in bytes, but a byte format is not defined for the memory being referenced.

Action: Use the FORMAT command (refer to Section 4.5) to define a byte format for the referenced memory.

APDEBUG64-F-NODEPPSROM, PSROM is read only

Meaning: The user tried to deposit into a PSROM location (PS address $\geq$ FFFC00, hex).

Action: No action required.

APDEBUG64-F-NOFD, no file descriptors available to open file

Meaning: Too many files are open. Command buffer files may be nested too deeply.

Action: Do not nest debugger command files too deeply. Close listing or symbol files.

APDEBUG64-F-NOLDMOD, no load module specified

Meaning: The user did not specify a load module with either the LOAD command or the APDEBUG64 command.

Action: Specify the desired load module.

APDEBUG64-F-NOLINENUM, can't STEP in FTN language mode if line #s not selected

Meaning The program must be compiled with the LINENUM option in order to use the STEP command.

Action: Recompile with the LINENUM option (included in the DEBUG option) or switch to $APAL language mode (refer to Section 4.2) to step through the program by machine instructions rather than FORTRAN lines.

APDEBUG64-F-NOLOCALSYM, local symbol table not open

Meaning: The user tried to close the local symbol table when it was not open.

Action: No action required.

APDEBUG64-F-NOMEMOPEN, no memory open

Meaning: The user entered an EXAMINE command without specifying the memory to examine.

Action: Specify a memory with the EXAMINE command.

APDEBUG64-F-NOOVERLAY, image has no overlays

Meaning: The user requested overlay information (using the POVERLAY command) when there are no overlays in the load module.

Action: No action required.

APDEBUG64-F-NOSPACE, symbol table is full

Meaning: The debugger internal symbol table is full.

Action: Decrease the size of the symbol file (recompile with the NOLIN or NAM REF option and relink using the SYMOUT parameter).

APDEBUG64-F-NOSTRSPACE, not enough internal space to manipulate string

Meaning: There is not enough internal buffer space to perform the requested operations on the specified string or strings.

Action: Use smaller strings or substrings.

APDEBUG64-F-NOSYMFILE, symbol file is not open

Meaning: The user tried to close the symbol file or open local symbols when no symbol file was open.

Action: Open the symbol file with the SYMBOL command (refer to Section 4.3).

APDEBUG64-F-NOVALUE, there is no value on the input line.

Meaning: The debugger expects a value on the input line, probably to be deposited to a memory.

Action: Check input file to verify that it has one value per line.

APDEBUG64-F-NUMOFFILES, illegal number of files specified

Meaning: The user specified too many files on the LOAD command.

Action: Only one file can be specified on the LOAD command (refer to Section 3.5). Enter the corrected command line.

APDEBUG64-F-ODTCORRUPT, image overlay description table invalid

Meaning: The debugger tried to read the AP's overlay description table but found it to be corrupt.

Action: The user's program may be overwriting the overlay description table.

APDEBUG64-F-OUTOFBOUNDS, address out of bounds

Meaning: The memory address referenced is either outside the memory's physical bounds or outside the user's space within the memory.

Action: Enter in-bounds memory address.

APDEBUG64-F-OVF/UNF, overflow/underflow error

Meaning: Arithmetic error in evaluating an expression.

Action: Enter corrected command line.

APDEBUG64-F-OVLNAME, unknown overlay name

Meaning: The user entered an overlay name (indicated by the '\\') that is not in the symbol file.

Action: Enter the corrected command line.

APDEBUG64-F-OVLNONRES, overlay is nonresident

Meaning: The user tried to call a routine (with the INVOKE command) not in the currently resident overlay or to access a data or program location not in the currently resident overlay.

Action: No action required.

APDEBUG64-F-QUOTE, illegal single quote

Meaning: An asterisk (') appeared with a parameter that was not a file name or a radix specifier.

Action: Re-enter line.

APDEBUG64-F-RADIX, bad radix specifier

Meaning: Incorrect radix specifier entered.

Action: Enter correct radix specifier (i.e., /B, /O, /D, or /Z).

APDEBUG64-F-RUNMEM, execution address must be a program location

Meaning: For program execution, the starting address must be a FORTRAN line number or statement label or an APAL64 label or PS location.

Action: Enter the command line with the corrected starting address.

APDEBUG64-F-SIZFILNAME, Internal - no room for file name

Meaning: The file name is too long.

Action: Shorten file name.

APDEBUG64-F-SRSTACK, subroutine return stack overflow

Meaning: The INVOKE command cannot be used because the AP subroutine return stack lacks space (as indicated by the value of the SRA register).

Action: No action required.

APDEBUG64-F-STRINGTOOBIG, string too long for address character length

Meaning: The user tried to search a character variable or array for a substring that is longer than the defined length of the variable or array elements.

Action: Shorten the string to $\leq$ the character length of the variable or array.

APDEBUG64-F-STROUTPUT, not enough output space for entire string

Meaning: String is too long for internal output buffer.

Action: Examine substrings of the string variable.

APDEBUG64-F-SUBSCRIPT, incorrect number of subscripts

Meaning: The user specified an incorrect number of array subscripts.

Action: Specify correct subscripts.

APDEBUG64-F-SUBSTRING, bad substring expression.

Meaning: The debugger found invalid syntax when evaluating a substring expression.

Action: Enter the command line with the correct syntax (APFTN64 Reference Manual).

APDEBUG64-F-SWITCH, unrecognized switch

Meaning: The switch given on the command line is not a valid debugger switch.

Action: Enter valid debugger switch.

APDEBUG64-F-SYMBOL, unrecognized symbol

Meaning: A symbol is not in the debugger symbol table or the user global or local symbol table. The user may have tried to use the line number of a non-executable APFTN64 statement as a symbol.

Action: Open the necessary user symbol table or enter the command correctly.

APDEBUG64-F-SYMCLOSE, do not specify file name when closing symbol file.

Meaning: The SYMBOL command does not accept a file name when closing a symbol file.

Action: Re-enter the command.

APDEBUG64-F-SYMNONRES, cannot evaluate symbol in nonresident overlay

Meaning: The contents of the AP at a program location (PS memory) or at a data location (MD memory) cannot be accessed because the symbol describing the location is not part of the currently resident overlay.

Action: Enter command line without the invalid symbol.

APDEBUG64-F-SYMTYPE, symbol in expression does not match expression type

Meaning: An expression is not compatible with its memory type.

Action: Remove the address expression type (using the INT operator discussed in Section 5.2.2) before using it in an expression of a different type.

APDEBUG64-F-TIMOUTCONV, error converting time-out value to real (internal)

Meaning: The user entered an incorrect value with PTIMEOUT command.

Action: Read Section 5.9 and then reenter the command with a correct value.

APDEBUG64-F-TIMOUTVAL, time-out value must be an integer no. >=0 & <=65535.

Meaning: The user entered an incorrect value with the PTIMEOUT command.

Action: Read Section 5.9 and then reenter the command.

APDEBUG64-F-TTYOPEN, error in opening terminal

Meaning: Debugger cannot open terminal.

Action: Report error to FPS.

APDEBUG64-F-VFPFAIL, VFP operation failed.

Meaning: The debugger cannot access the Virtual Front Panel, probably because the user is not privileged.

Action: No action required.

APDEBUG64-F-UNKNOWN, unknown error number

Meaning: Internal error.

Action: Report error to FPS.

APDEBUG64-F-WATCHMEM, a watchpoint may be set at a main data location only

Meaning: The user tried to set a watchpoint at other than a program variable or an array element or MD location.

Action: Correct watchpoint setting. Use breakpoint to stop on a PS location.

APDEBUG64-F-X64BUSY, requested X64 is busy, try another

Meaning: The requested AP is busy. If no AP number was specified, then all AP's are in use.

Action: Try again.

APDEBUG64-F-X64MEM, X64 memory request too large

Meaning: The user requested too much memory with the MDSIZE or PSSIZE parameter on the APDEBUG64 or INIT command.

Action: Enter a legal MDSIZE or PSSIZE memory size.

APDEBUG64-F-X64NOTAVAIL, requested X64 not available, try another

Meaning: The requested AP is off-line, or the user entered an illegal AP number.

Action: Request another AP.

APDEBUG64-F-ZERODIVIDE, division by zero

Meaning: Expression evaluation error.

Action: Check expression for division by zero.

APDEBUG64-W-BADSMB, this module is not a symbol module

Meaning: The module indicated on the error message is not recognized as part of a valid symbol file produced by APLINK64.

Action: Enter the command line with the correct symbol file name.

APDEBUG64-W-BOUNDSWARN, address out of bounds

Meaning: A PS or MD address is outside the current MDLIMIT or PSLIMIT register setting.

Action: No action required.

APDEBUG64-W-CHARLEN, data extends past character symbol length

Meaning: The debugger will display the contents of a main data memory address pointed to by a character variable symbol. However, the character symbol only occupies part of the word. The debugger will display the substring expression as if the character variable did occupy the entire word but indicate the correct character length on the error message.

Action: No action required.

APDEBUG64-W-DIMENBND, subscript outside dimension bounds

Meaning: The user specified an array element with a subscript outside dimension bounds.

Action: Specify correct subscripts.

APDEBUD64-W-DUPENTRY, duplicate entry name in symbol file

Meaning: The debugger found two entry points in the same overlay with the same name. The debugger ignores the second entry point.

Action: No action required. The user can examine the APLINK64 map to eliminate the duplicate entry name.

APDEBUG64-W-DUPMODNAME, duplicate module name in symbol file

Meaning: The symbol file contains two modules with the same name. The first occurrence of the module name is used. The second is ignored.

Action: Change the name of one of the program modules. Examine APLINK64 map to determine cause of duplicate module.

APDEBUG64-W-DUPOVERLAY, duplicate overlay name in symbol file

Meaning: The debugger found two overlays with the same name. The debugger ignores the second overlay.

Action: No action required. The user can examine the APLINK64 map to eliminate the duplicate overlay.

APDEBUG64-W-HALTED, AP is halted

Meaning: Neither the user's program nor the SUM is running on the AP.

Action: Initialize the AP (refer to Section 3.6).

APDEBUG64-W-INITSW, excess information on INIT/NOINIT switch ignored

Meaning: The INIT/NOINIT options require no additional modifiers.

Action: Reenter command line.

APDEBUG64-W-INSUFFVAL, insufficient data, excess locations not initialized

Meaning: The data file being read by the DEPOSIT command does not contain enough data to deposit into all the locations specified in the header line or DEPOSIT command.

Action: Verify that the data file is correct.

APDEBUG64-W-LISTSW, error in LIST/NOLIST switch

Meaning: Incorrect number of files specified on switch.

Action: Re-enter command line.

APDEBUG64-W-MAXCALLRETURN, max number of calls/returns exceeded

Meaning: The user exceeded the maximum number of $CALLS or RETURNS breakpoints.

Action: Include only the routines that the user is currently debugging in the symbol file by relinking with the SYMOUT parameter.

APDEBUG64-W-MEMSIZSW, error in PS or MD memory size switch

Meaning: The user entered an invalid number or more than one number on the APDEBUG64 command's PSSIZE or MDSIZE parameter.

Action: Enter the APDEBUG64 command with the correct PSSIZE or MDSIZE parameter (refer to Section 3.6).

APDEBUG64-W-NOCHARSYM, no character symbol found for output

Meaning: The debugger tried to display the contents of a character address but cannot find a local character symbol to match the address. Instead, the debugger will display the memory name and address.

Action: No action required.

APDEBUG64-W-NOFIELDFMT, no field format has been defined

Meaning: The user tried to express an output value in field (F) format, but the accessed memory or register is not defined in field format.

Action: Designate another format for output (refer to Section 4.5).

APDEBUG64-W-NOFLDFMT, field format illegal

Meaning: The debugger cannot express the output value in field (F) format because it cannot determine which memory or register to access.

Action: Report error to FPS.

APDEBUG64-W-NOLOCAL, no local symbols for this module

Meaning: The current module has no local symbols in the symbol file.

Action: Recompile or reassemble as outlined in Section 2.3.

APDEBUG64-W-NOSYMBOLS, this module not in symbol file

Meaning: The user's program stopped in a module that is not included in the symbol file.

Action: If the user wants local symbols, recompile or reassemble. Relink to get the module into the symbol file.

APDEBUG64-W-RUNNING, AP is running, enter INIT OR HALT command

Meaning: The AP is running and must be initialized.

Action: Initialize the AP (refer to Section 3.6).

APDEBUG64-W-SYMBOLSW, error in SYMBOL/NOSYMBOL switch

Meaning: Incorrect number of files specified on switch.

Action: Enter corrected command line.

APDEBUG64-W-SYMOUTPUT, no room for symbol information in output line

Meaning: The symbolic representation of the location being examined or the current program location at which the program is stopped is too long for the 80 characters allotted for it. Only 80 characters can be put into the output line, followed by any other necessary information.

Action: No action required.

APDEBUG64-W-TOOMANYBYTES, too many bytes specified, remainder ignored

Meaning: The user specified a value in byte format that has more bytes than defined by the FORMAT command. The excess bytes are ignored.

Action: Use the FORMAT command (refer to Section 4.5) to redefine the byte format.

APDEBUG64-W-TMRAMSW excess information on TMRAM/NOTMRAM switch ignored.

Meaning: The TMRAM/NOTMRAM options require no additional modifiers.

Action: Re-enter command line.

APDEBUG64-W-VALOUTPUT, not enough room for value in output line

Meaning: The user requested more bytes of data than can fit on the output line.

Action: Use the FORMAT command (refer to Section 4.5) to specify a larger byte format.

APDEB64-W-WAITSW, excess information on WAIT/NOWAIT switch, ignored.

Meaning: The WAIT/NOWAIT options require no additional modifiers.

Action: Re-enter command line.

APDEBUG64-W-X64ASSIGNED, another X64 already assigned to user

Meaning: The user is already attached to another AP.

Action: No action required; the user remains attached to the AP.

First Class
Permit No. A-737
Portland,
Oregon

BUSINESS REPLY

No postage stamp necessary if mailed in the United States

Postage will be paid by:

FLOATING POINT SYSTEMS, INC.

P.O. Box 23489

Portland, Oregon 97223

Attn: Technical Publications

READER'S COMMENT FORM

Your comments will help us improve the quality and usefulness of our publications. Please fill out and return this form to: Floating Point Systems, the mailing address is on the back.

Title of document _____

Name/Title _____ Date _____

Firm _____ Department _____

Address _____

Telephone _____

I used this manual...

- ☐ as an introduction to the subject
- ☐ as an aid for advanced training
- ☐ to instruct a class
- ☐ to learn operating procedures
- ☐ as a reference manual
- ☐ other _____

I found this material...

	Yes	No
accurate/complete	☐	☐
written clearly	☐	☐
well illustrated	☐	☐
well indexed	☐	☐

Please indicate below, listing the pages, any errors you found in the manual. Also indicate if you would have liked more information on a certain subject.

FLOATING POINT
SYSTEMS, INC.

...the world leader in array processors

CALL TOLL FREE (800) 547-1445
Ex. 4999. P.O. Box 23489 (S 500),
Portland, OR 97223 (503) 641-3151.
TLX: 360470 FLOATPOIN BEAV