FLOATING POINT SYSTEMS, INC.

APDBUG Manual

by FPS Technical Publications Staff

APDBUG Manual

2nd Edition, February 1978

This edition applies to Release 2.0 of the AP simulator (APSIM) and the AP-120B hardware debugger (HWDBUG) software and all subsequent releases until indicated by a new edition. This supersedes the AP-120B debugger manual FPS-7277-02.

Publication No. FPS-7364-01.

NOTICE

THE MATERIAL IN THIS MANUAL IS FOR INFORMATIONAL PURPOSES ONLY AND IS SUBJECT TO CHANGE WITHOUT NOTICE.

FLOATING POINT SYSTEMS, INC. ASSUMES NO RESPONSIBILITY FOR ANY ERRORS WHICH MAY APPEAR IN THIS PUBLICATION.

Printed in U.S.A.

CONTENTS

CHAPTER 1
INTRODUCTION

APDBUG provides an interactive facility for checking out AP-120B programs. The user can run portions of his AP-120B program, stop and examine the results, make program patches and then continue with program execution. The process of interactive debugging greatly facilitates preparation of correctly operating AP-120B programs.

APDBUG has commands to:

1. Examine and/or change memory locations and registers inside the AP-120B

2. Type out memory contents and integers, floating point numbers or program word fields

3. Set, clear and examine breakpoints

4. Run programs, or execute them one step at a time

Two versions of APDBUG are available: one which executes programs in the AP-120B hardware (HWDBUG) and one which executes programs in the AP-120B simulator (APSIM).

The simulator APDBUG (called APSIM) is most convenient for initial program development and has the advantage that it allows debugging off-line from the AP-120B hardware. It allows access to more internal AP-120B registers than with the hardware. Simulation is limited, however, to program execution inside the AP-120B. Input/Output interaction with the host computer is not simulated. Depending upon the speed of the host computer, the simulator runs about one million times slower than real time or about six instruction cycles per second.

The hardware APDBUG (called HWDBUG) is convenient for debugging AP-120B programs which require long execution times and/or real-time interaction with the host computer.

This document describes Release 2.0 of APSIM and HWDBUG. In the description below, (cr) means carriage return or end of line, as appropriate to the particular host computer system. In the examples listed, the responses typed by the computer are underlined.

CHAPTER 2
OPERATING PROCEDURE

Debugging is the process of detecting, locating and removing mistakes from a program. When the programmer wishes to debug an AP-120B program, he loads the program into APDBUG. The user may then control program execution, causing the program to breakpoint at selected program locations so that he can examine the contents of registers or memory locations. Contents may be examined as program words, integers or floating point numbers.

APDBUG types a "*" when ready for user input. A "?" is typed when an error is detected.

2.1 MONITORING REGISTERS AND MEMORY LOCATIONS

Registers and memory locations in the AP-120B may be opened, examined and modified using one of the following commands:

E	open and examine locations in the AP-120B
+	examine the next higher location in an AP-120B memory
-	examine the next lower location in an AP-120B memory
C	change the open location
.	re-examine the currently open location
Z	zeros out all AP-120B registers and memories
O	set program source memory offset

A register in the AP-120B is opened with an "E" (exam), "+" (next), or "-" (last) command. APDBUG gets the value of the desired location in the AP-120B and types out the value on the user console. If desired, the contents of that location may be changed with a "C" (change) command. A "." (re-exam) types the contents of the open register.

2.1.1 "E", Open and Examine

To open and examine a register in the AP-120B:

```
E (cr)
name (cr)
```

where NAME is the name of the desired register.

To open and examine a memory location in the AP-120B:

```
E (cr)
name (cr)
location (cr)
```

where NAME is the memory name and LOCATION is the desired memory location.

A list of the examinable registers and memories and their memories is given in an appendix on page A-6. For the purposes of APDBUG all functional units of the AP-120B which have addresses are considered "memories". This includes the three obvious memories (Main Data Memory, Table Memory and Program Memory) plus Data Pad X, Data Pad Y and S-Pad.

Some examples:

1. Examine Main Data Memory Location 23:

```
*
E (cr)
MD (cr)
23 (cr)
-234.0000000
*
```

MD location 20 contains -234.0

2. Examine the Memory Address Register

```
*
E (cr)
MA (cr)
1376
*
```

MA contains 1376

2.1.2 "+", "-" and ".", Examine Next, Last and Re-examine

To open and examine the next higher sequential memory location above a currently open memory location:

```
+ (cr)
```

To open and examine the next lower sequential memory location below a currently open memory location:

```
- (cr)
```

To re-examine the currently open memory location:

```
. (cr)
```

Examples:

1. Examine Main Data Memory locations 23 and 24

```
*
E (cr)
MD (cr)
23 (cr)
-234.0000000
*
```

MD location 23 contains -234.0, now examine MD location 24

```
*
+ (cr)
MD   000024
789.0000000
*
```

MD location contains 789.0

2. Examine S-Pad registers 7 and 6

```
*
E (cr)
SP (cr)
7 (cr)
000027
*
```

S-Pad register 7 contains 27. Now examine register 6

```
*
- (cr)
SP   000006
-136
*
```

S-Pad register 6 contains -136

2.1.3 "C", Change

To change the contents of a currently "open" register or memory location to a specified value:

```
C (cr)
value (cr)
```

where VALUE is an integer(s) or floating point number (depending upon what register or memory is "open"). (See section 2.2.)

To change a register or MEMORY location the user must first "open" it by doing an "E", "+" or "-" command.

Examples:

1. Examine Main Data Memory Location 20 and then change its value to -97.5.

```
*
E (cr)
MD (cr)
20 (cr)
76.00000000
*
C (cr)
-97.5 (cr)
*
```

Main Data Memory location 20 contained 76.0. The user changed it to contain -97.5.

2. Now change Main Data Memory location 21 to -63.4

```
*
+ (cr)
MD   000020
- 3.000000000
*
C (cr)
-63.4 (cr)
*
```

MD location 21 contained -3.0 and was changed to contain -63.4.

3. Examine S-Pad Register 3 and change its value to 17:

```
*
E (cr)
SP (cr)
3 (cr)
56
*
C (cr)
17 (cr)
*
```

S-Pad Register 3 contained 56 and was changed to contain 17.

2.1.4 "O", Set P.S. Offset

To set the Program Source Memory addressing offset:

```
O      (cr)
value  (cr)
```

where VALUE is an integer in the current radix specifying the offset to be used when accessing Program Memory. The default setting is 0.

The offset is used when debugging a load module containing several separately assembled programs. For example, assume that programs A, B and C have been loaded together with APLINK and the following load map obtained from APLINK with the "S" command:

SYMBOL	VALUE
A	000000
B	000153
C	000247

meaning that program "A" was loaded at PS location 0, "B" at location 153 and "C" at location 247.

To examine locations 3 and 4 of program "B", type:

```
*
E    (cr)
PS   (cr)
156  (cr)
000000  000000  000000  000000
*
+  (cr)
PS    000157
000000  000000  000000  000000
*
-
```

This is confusing because the locations printed by APDBUG, 156 and 157, don't agree with the APAL listings which always start at zero. The offset simplifies matters by adjusting the base address for ALL PS related I/O. Thus, for convenience-sake, the offset should be set to the base address of the program currently being examined:

```
*
0     (cr)
153   (cr)
*
E     (cr)
PS    (cr)
3     (cr)
000000  000000  000000  000000
*
+     (cr)
PS     000004
000000  000000  000000  000000
*
```

The offset applies to Examining or Changing PS and PSA and also to Breakpoints and Running programs. It should be remembered, when setting the offset, that it is NOT relative to itself, but is an absolute address. Thus, the offset can always be reset to the default value of zero by typing:

```
0  (cr)
0  (cr)
```

2.2 CHANGING INPUT/OUTPUT FORMATS

The input and output format used when examining and changing registers and memory locations may be selected using the following commands:

N Sets the radix for integers

F Sets the format for input/output of 38-bit wide registers and memory words

V Sets the format for input/output of 64-bit wide program memory words

APDBUG selects the proper format for input/output depending upon the word size of the particular register or memory location that is open and the setting of the above three flags:

1. 16-bit words: MA, TMA, DPA, S-Pad, etc. These locations are examined or changed as integers in the radix as selected by "N"
2. 38-bit words: DPX, DPY, Main Data Memory, Table Memory, etc. These locations are examined or changed as either floating point numbers or as three integers depending upon the "F" flag
3. 64-bit words: Program memory. These locations are examined or changed as either op-code fields or as four 16-bit integers depending upon the "V" flag

The listing of accessible AP-120B register and memories on page A-6 specifies the width of each as:

```
   16-bit  (integer word)
or 38-bit  (floating point word)
or 64-bit  (program word)
```

NOTE

Integer output is always unsigned on the range 0-177777 (octal), or 0-65536 (decimal), or 0-FFFF (hex). Thus negtive twos-complement numbers will be typed out as their 16-bit unsigned equivalent. For example (in octal) -1 would be output as 177777, and -2 as 177776, and so forth.

2.2.1 "N" Set Radix

To set the radix for all integers input/output to APDBUG:

```
N (cr)
radix (cr)
```

where the Radix is either 8, 10 or 16 for octal, decimal or hexadecimal radices respectively. (Note that the radix number is always in decimal.)

The contents of 16-bit wide registers (S-Pad, MA, PSA, etc.) are examined and changed using the integer radix as set by the "N" command. In addition, memory addresses are also entered using the current radix.

On type-outs, octal numbers may be recognized as having six digits, decimal numbers as having five digits and hex numbers as having four digits.

The default radix is either octal or hex depending upon the conventions of the host computer.

Examples:

1. Examine S-Pad register 10 (decimal) in all three radices (starting in decimal)

```
*
E (cr)
SP (cr)
10 (cr)
 32768
*
N (cr)
8 (cr)
*
. (cr)
SP     000012
100000
*
N (cr)
16 (cr)
*
. (cr)
SP       000A
  8000
*
```

The value of S-Pad register 10 is 32768 (decimal) or 100000 (octal) or 8000 (hexadecimal).

2.2.2 "F" Set/Reset Floating Point I/O

To select floating point input/output of 38-bit registers and memory words:

```
F (cr)
1 (cr)
```

To select integer (in the current integer radix) input/output of 38-bit wide registers and memory locations:

```
F (cr)
0 (cr)
```

38-bit wide registers are split into three pieces: 10-bit exponent, 12-bit high mantissa (bits 0-11) and 16-bit low exponent (bits 12-27) for integer I/O.

Data Pad, Main Data Memory, Table Memory and Data Pad Bus are among the registers and memories whose I/O is governed by the "F" flag.

Both examining and changing of 38-bit registers are effected by "F". The default setting of the "F" switch is one for floating point I/O.

1. Examine command output formats:

```
F=1:  (floating point number)

F=0:  (exponent) (high mantissa) (low mantissa)
```

2. Change command input formats:

```
F=1:  C (cr)
      (floating point number)  (cr)

F=0:  C (cr)
      (exponent)        (cr)
      (high mantissa)   (cr)
      (low mantissa)    (cr)
```

Legal floating point numbers are of the form

    +or-XX.YYE+or-ZZ

where XX is the integer part
      YY is the fraction part
      ZZ is the exponent

Any of the three parts may be omitted, except in the case when an exponent is used. In this case, either an integer part or a fraction part must also be included. The signs may be omitted if "+" is used. The decimal point may be omitted if not needed. No spaces are allowed inside floating point numbers.
The following are all legal floating point inputs.

```
-2.3E6
  .7E-3
 -2
 3.65
  .7
```

Examples:

1. Examine Data Pad register six in both floating point and integer. (Assume the integer radix is 16.)

```
*
E (cr)
DPX (cr)
6 (cr)
-1.000000000
*
F (cr)
0 (cr)
*
. (cr)
DPX     0006
  0200  0400  0000
*
```

DPX register six contains -1.0. Its exponent is 200 (hexadecimal) which has an exponent value of zero (0). The fraction part is 4000000 (hexadecimal) which is a fraction of -1.0.

2. Now change the exponent to 204 and the fraction to 2000000 and set "F" to 1:

```
*
C (cr)
204 (cr)
200 (cr)
0 (cr)
*
F (cr)
1 (cr)
*
. (cr)
DPX      0006
8.000000000
*
```

DPX register now contains 8.0 which is 0.5*2**4.

2.2.3 "V" Set/Reset Program Word Field I/O

To select input/output of 64-bit wide programs words by op-code fields:

```
V (cr)
1 (cr)
```

To select input/output of program words as four 16-bit numbers:

```
V (cr)
0 (cr)
```

The four 16-bit integers represent bits 0-15, 16-31, 32-47 and 48-63 of a program word.

Both examining and changing of Program words are effected by "V".

The default for the "V" flag is 0 for integer I/O of program words.

1. Examine command output formats:

```
V=1:  (24 op code field values)
V=0:  (bits 0-15) (bits 16-31) (bits 32-47) (bits 48-63)
```

2. Change command input formats:

```
V=1:  C (cr)
      (desired op-code field to change)  (cr)
      (new value)  (cr)

V=0:  C (cr)
      (bits 0-15)   (cr)
      (bits 16-31)  (cr)
      (bits 32-47)  (cr)
      (bits 48-63)  (cr)
```

The Program word op-code fields are listed on page A-8 of the appendix. When V=1, on Examine, the 64-bits of a program word are divided into 24 fields, whose values are printed out. On Change, the user enters the name of the field he wants to change along with the new value (hence the mnemonic "V") for that field. The legal values for each field are listed in appendix B of AP-120B Processor Handbook (Form #7259).

Examples:

1. Program location 20 contains the instruction

```
LDSPI 14; DB=200
```

which sets S-Pad register 14 to 200. Patch it so that S-Pad will be set to 300 instead.

```
*
E (cr)
PS (cr)
20 (cr)
001660 000000 000000 000200
*
C (cr)
1660 (cr)
0 (cr)
0 (cr)
300 (cr)
*
```

Note that to change the "value" field (which is the fourth quarter) (bits 48-63) of the program word, all four quarters had to be typed in.

2. Now change the instruction so that S-Pad register 11 (instead of 14) will get loaded with 300.

```
*
V (cr)
1 (cr)
*
C (cr)
SPD (cr)
11 (cr)
*
```

The SPD (S-Pad Destination) field (program word bits 10-13) was changed to 11.

3. Examining Program Memory location 20 in both formats yields:

```
*
. (cr)
PS     000020
001644 000000 000000 000300
*
V (cr)
1 (cr)
*
. (cr)
PS     000020
   B     00    SOP 00    SH   00    SPS  14    SPD 11    FADD 00
   A1    00    A2  00    COND 00    DISP 00    DPX 00    DPY  00
   DPBS  00    XR  00    YR   00    XW   00    YW  00    FM   00
   M1    01    M2  02    MI   00    MA   00    DPA 00    TMA  00
```

SPS=14 means "LDSPI", SPD=11 means S-Pad destination register is 11. The YW, FM, M1, M2, M1, MA, DPA and TMA fields are meaningless since the "value" of 300 occupied these fields.

4. Program location 30 contains the instruction:

```
FADD FM, MD; FMUL TM, MD
```

Change the second argument for the "FADD" (A2) from MD to FA

```
*
E (cr)
PS (cr)
30 (cr)
000001 114000 000000 160000
*
V (cr)
1 (cr)
*
C (cr)
A2 (cr)
1 (cr)
*
```

2.3 MEMORY LOADING AND DUMPING

Blocks of AP-120B memory locations may be loaded and dumped to and from files with the following commands:

- Y Yank (load) into a memory from a file
- W Write out the contents of a memory to a file
- Z Zero all the memories (and registers) (APSIM only)

The list of memories on which the above commands may operate is different for APSIM and for HWDBUG. In APSIM, only Main Data memory (MD), Table Memory (TM) and Program Memory (PM) may be yanked into or written from. In HWDBUG, the list of memories is extended to include S-Pad (SP) and Data Pad X and Y (DPX, DPY).

A further difference lies in the area of I/O data formats. In APSIM, "Y" and "W" to/from 38-bit memories are always in the floating point format (F=1). Program memory I/O is always in integer mode (V=0). In HWDBUG, I/O to/from 38-bit memories are governed by the "F" switch. Hence, it is either in floating point or integer, as set by the user. Program Memory input is always in integer mode, whereas output may be in either integer or op-code field format, as governed by the current setting of the "V" switch.

The user should be aware that the procedure for typing in filenames varies greatly according to the respective system. In some systems the notion of user files is nonexistent. In these cases, a logical unit number referring to an I/O device, which was opened previously by JCL control statements must be entered in place of a filename. Other systems allow access to disk files, line printers and terminals by symbolic names. Thus, what must be entered for a filename depends on the convention of each respective system. The examples given below are only meant to be representative and may not be legal on a given system.

2.3.1 "Y", Yank From a File

To load a memory from a file:

```
Y                  (cr)
memory name        (cr)
starting location  (cr)
filename           (cr)
```

where MEMORY NAME is an AP-120B memory, the beginning memory address is loaded at the STARTING LOCATION. The name of the file from which the data is to be read is called FILENAME. The filename must, of course, be in the proper form as determined by the particular host operating system.

Yank is used typically to load programs into Program Memory and data into Main Data memory. Some examples:

1. Load a program into PS location 0. The program is assumed to be in a file named MYPROG which was made using the "E" command output from APLINK.

```
*
Y       (cr)
PS      (cr)
0       (cr)
MYPROG  (cr)
*
```

2. Load data into MD starting at location 20 from a file called DATA. Section 2.3.4 explains how to create data files:

```
*
Y     (cr)
MD    (cr)
20    (cr)
DATA  (cr)
*     (cr)
```

2.3.2 "W", Write To a File

To write the contents of a memory into a file:

```
W                  (cr)
memory name        (cr)
starting location  (cr)
ending location    (cr)
filename           (cr)
```

where MEMORY NAME is an AP-120B memory, STARTING LOCATION is the initial address to be written, ENDING ADDRESS is the last address to be written and FILENAME is the name of the file into which the data is to be written.

Some examples:

1. Write Main Data memory locations 20 through 40 into a file called DUMP:

```
*
W     (cr)
MD    (cr)
20    (cr)
40    (cr)
DUMP  (cr)
*
```

2. Write Data Pad X locations 3 through 6 to the line printer (first, in floating point format and second, in integer format). (Strictly as an example, the line printer is called LP:.) Note that Data Pad may be dumped only from HWDBUG.

```
*
F    (cr)
1    (cr)
*
W    (cr)
DPX  (cr)
3    (cr)
6    (cr)
LP:  (cr)
*
F    (cr)
0    (cr)
*
W    (cr)
DPY  (cr)
3    (cr)
6    (cr)
LP:  (cr)
*
```

If the user mistypes a "W" command, he has several options to abort the command. If the wrong memory name or starting address was typed, then the command may be canceled by entering an ending address (which is lower than the starting address). In HWDBUG, an unwanted dump already underway (for example, when a location 1000 was typed whereas location 100 was wanted) can be aborted by a USER BREAK. How this is accomplished varies from system to system. Typically, on single-user mini-computer systems, it is accomplished by raising the most significant bit of the host switch register.

2.3.3 "Z", Zero the AP-120B

The "Z" command is legal only in APSIM. It zeros out all the registers and memories in APSIM. It should be the first command given to APSIM. It is accomplished by:

```
Z (cr)
```

2.3.4 Preparing Data Files for Yanking

Data files may be prepared by the user for loading into MD and TM by using APSIM (typically prepared by using the host system editor). The files may be prepared for loading into MD, SP, DPX and DPY by using HWDBUG. The format of the data file is as follows:

```
data count
data item #1
data item #2
... ... ...
data item #N
```

All entries must be left justified, one entry per line.

The data count is the number of memory locations to be filled and written as a real number (with a decimal point). Thus, if there were three data items, the count would be "3.".

The format of data items depends upon which debugger is used. In APSIM, only floating point numbers may be loaded. These must appear one per line in the data file. In HWDBUG, the format is determined by the "F" switch setting for 38-bit memories. For integer formats, the radix is determined by the N (radix) setting. When floating point numbers are used they appear one per line. Also, integers must appear one per line in the file. Thus, for 38-bit memories written in integer format (F=0), three integers (exponent, high mantissa, low mantissa), written on three separate lines, must be included for each memory location.

Some examples:

1. Four element floating point data file:

```
4.
1.2
 .3
-6E7
2.3E-5
```

2. Three element integer data for a 38-bit wide memory which will load three integers into the low mantissa (HWDBUG only):

```
3.
0
0
1
0
0
2
0
0
3
```

2.4 EXECUTING PROGRAMS

AP-120B program execution may be controlled with the following commands:

- B set breakpoint
- D delete breakpoint
- L list breakpoint
- Q set breakpoint counter
- S set step mode
- I initialize the AP-120B
- R run an AP-120B program
- P proceed (continue) with an AP-120B program
- T print elapsed execution time
- M set memory speed
- X exit to the host operating system

The typical strategy when debugging a program is to set breakpoints at a strategic location in the program. Run the program. When it hits the breakpoint, examine various data locations to see what has been changed correctly or incorrectly. Thus, the user will typicaly alternate between running a program and examining the results.

2.4.1 "B", Set Breakpoint

To set a breakpoint:

```
B             (cr)
memory name   (cr)
location      (cr)
```

where MEMORY NAME is the memory on which to break execution (must be MD, TM or PS) and LOCATION is the memory address on which to stop. The AP-120B allows breakpointing on two conditions: 1) read or write of a given Main Data memory or Table Memory location, or 2) execution of a given program instruction. Contrary to typical debuggers, the program halts AFTER executing the breakpointed instruction. Only one breakpoint may be set at a time. Setting a new breakpoint clears any previously set breakpoint. Users of HWDBUG should consult section 4.2 of the AP-120B Processor Handbook (#7259) for possible interaction between the breakpoint and the program.

Some examples:

1. Set a breakpoint in order that the program will stop after executing the instruction at location 20.

```
*
B    (cr)
PS   (cr)
20    (cr)
*
```

2. Set a breakpoint so that the program will halt after reading or writing Main Data location 100.

```
*
B     (cr)
MD    (cr)
100   (cr)
*
```

2.4.2 "D", Delete Breakpoint

To delete a breakpoint:

```
D  (cr)
```

Delete clears any previously set breakpoints.

2.4.3 "L", List Breakpoint

To list on the console what breakpoint is currently set:

```
L  (cr)
```

APDBUG will type the memory name in which the breakpoint is set, followed by the location of the breakpoint. If no breakpoint is set, only an asterisk ("*") is typed.

For example, if a breakpoint is set at PS location 20, then:

```
*
L  (cr)
PS    000020
*
```

2.4.4 "Q", Set Breakpoint Counter

To set the breakpoint counter:

```
Q      (cr)
count  (cr)
```

where COUNT is the desired counter setting. The breakpoint counter is the number of times a breakpoint must be hit before the AP-120B program will halt. It is also used by the step flag. (See section 2.4.5.) For example, it is useful when a bug occurs every ten times around a loop. The count is reset to one every time a new breakpoint is set or the step flag is set or reset.

For example, set a breakpoint at program location 20 in order that the program halts only after hitting the breakpoint 10 times.

```
*
B   (cr)
PS  (cr)
20  (cr)
*
Q   (cr)
10  (cr)
*
```

2.4.5 "S", Set/Reset Step Mode

To set step mode:

```
S  (cr)
1  (cr)
```

To clear step mode:

```
S  (cr)
0  (cr)
```

In step mode, the program executes only a single instruction after being started and then halts. This is useful when sequencing step-by-step through a piece of code while watching for a data location to be destroyed or for the program to go awry. Step mode also uses the breakpoint counter which, in "step" mode, counts instructions to execute before stopping.

Some examples:

1. Set step mode so that when next started, the program will execute one instruction and then stop.

```
*
S  (cr)
1  (cr)
*
```

2. Set step mode so that when next run, the program will execute 100 instructions and then halt:

```
*
S    (cr)
1    (cr)
*
Q    (cr)
100  (cr)
*
```

2.4.6 "I", Initialize the AP-120B

To initialize (reset) the AP-120B:

    I (cr)

In APSIM, the initialize command clears the memory timing and arithmetic pipelines. In HWDBUG, an interface reset is done to the AP-120B. This is necessary to stop a program when that program has "run away."

2.4.7 "R", Run an AP-120B Program

To run an AP-120B program:

```
R         (cr)
location  (cr)
```

where LOCATION is the program location where execution starts. APDBUG starts the program at the specified location and then waits until the program hits a breakpoint. If the program "runs away" in APSIM, the user typically has no recourse. In HWDBUG, control can be regained by causing a USER BREAK. (See section 2.3.2.)

When the AP-120B program finally halts, APSIM responds by printing out the current program address (PSA), the total elapsed program execution time after the last "R" command, and the contents of the currently open register or memory location. HWDBUG merely responds with an asterisk ("*").

Some examples:

1. In APSIM, examine MA. Then, set a breakpoint at program location 16. Then start execution at location 10:

```
*
E   (cr)
MA  (cr)
123
*
B   (cr)
PS  (cr)
16  (cr)
*
R   (cr)
10  (cr)
PSA=000017       1.17 us.
MA
123
*
```

The program has executed 1.17 us and stopped with location 17 as the next instruction to be executed. MA hasn't changed. Note that the printout of the last examined location is useful for monitoring registers to see when they change.

2. In HWDBUG, set a breakpoint at program location 16, then start execution at location 10:

```
*
B   (cr)
PS  (cr)
16  (cr)
*
R   (cr)
10  (cr)
*
```

HWDBUG signals program return merely by a "*"

2.4.8 "P", Proceed with AP-120B Program Execution

To proceed with AP-120B program execution:

```
P  (cr)
```

Proceed is used to resume AP-120B program execution after hitting a breakpoint or when stepping. The program continues in the location wherever the address is currently in the Program Source Address register (PSA). When the program next encounters a breakpoint, the typeout is the same as that which follows a return from a Run.

Some examples:

1. Set a breakpoint at location 16, run at location 10, examine S-Pad 3, then continue execution.

```
*
B   (cr)
PS  (cr)
16  (cr)
*
R   (cr)
10  (cr)
PSA=000007       1.17 us.
*
E   (cr)
SP  (cr)
3   (cr)
000123
*
P   (cr)
```

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

```
*
S    (cr)
1    (cr)
*
E    (cr)
MA   (cr)
000103
*
R    (cr)
10   (cr)
PSA=000011        0.17 us.
MA
000104
*
P    (cr)
PSA=000012        0.33 us.
MA
000105
*
P    (cr)
PSA=000013        0.50 us.
MA
000106
*
```

2.4.9 "T", Execution Time

To print elapsed AP-120B program execution time up to the last Run (R) command (APSIM only):

```
T  (cr)
```

2.4.10 "M", Set Memory Speed

To set Main Data Memory speed (APSIM only):

```
M
speed (cr)
```

where SPEED is 1 for FAST memory timing and 2 for STANDARD memory timing. The default is 2 for STANDARD memory timing.

2.4.11 "X", Exit to the Host System

To complete a debugging session and exit to the host operating system:

```
X  (cr)
```

APPENDIX A
SUMMARY OF APDBUG COMMANDS

Abbreviations used in the following appendices:

Symbol	Meaning
(cr)	Carriage Return
loc	An integer location number
count	An integer count
val	An integer value
fpn	A floating-point number in form acceptable to FORTRAN
mem	The name of an AP-120B internal memory
reg	The name of an AP-120B internal register

Debug types an "*" when ready for further action. A "?" is typed when a command is not understood.

A.1 Program Execution Commands

B mem loc	(cr) (cr) (cr)	Breakpoint. Delete the last breakpoint and set a new breakpoint at location LOC of memory MEM. MEM must be PS, MD, or TM.
D	(cr)	Delete. Delete the current breakpoint.
L	(cr)	List. List the current breakpoint.
Q count	(cr) (cr)	Set the continue counter to (COUNT).
S val	(cr) (cr)	Step. If (VAL) is not zero, place the AP-120B in step mode.
I	(cr)	Initialize. Reset the AP-120B before program execution is resumed next.
R loc	(cr) (cr)	Run. Begin program execution at Program Source location LOC.
P	(cr)	Proceed. Begin instruction execution at the Program Source location pointed to by the AP-120B PSA (Program Source Address) register.
T	(cr)	Print out elapsed execution time (APSIM only).
X	(cr)	Exit to the operating system.
M	(cr)	Set memory speed. VAL is 1 for one cycle (fast) memory.
val	(cr)	2 for two cycle (standard) memory (APSIM only).

A.2 Register Examination/Modification Commands

E reg	(cr) (cr)	Examine register. Print out the contents of AP-120B register REG.
E mem loc	(cr) (cr) (cr)	Examine memory. Print out the contents of AP-120B memory MEM, location LOC.
.	(cr)	Re-examine the currently open register or memory location (the last location examined).
+	(cr)	Examine the next higher sequential memory location of the memory that is currently open.
-	(cr)	Examine the next lower sequential memory location of the memory that is currently open.
F val	(cr) (cr)	Floating Point Flag, affects the input/output of 38-bit wide registers and memory locations. VAL=0: 3 integers (Exponent, High Mantissa, Low Mantissa) VAL=1: floating-point.
V val	(cr) (cr)	Program Source field value flag, affects input/output of program source memory location. VAL=0: 4 integers (the four 16-bit quarters of PS) VAL=1: Decode into the 24 instruction word field values.
C val	(cr) (cr)	Change. Change the contents of the currently open register or memory location to VAL. The format of VAL depends on the width of the current open locations as follows: 16-bit wide registers: an integer of the current radix. 38-bit wide registers: F=0; VAL (cr) three integers in the current radix VAL (cr) which represent the exponent, high VAL (cr) mantissa, and low mantissa F=1: FPN (cr) a floating point number legal to FORTRAN

64-bit wide registers:

```
V=0:  VAL   (cr) four integers in the current radix
      VAL   (cr) which are the four quarters of an
      VAL   (cr) AP-120B program word
      VAL   (cr)

V=1:  FIELD   (cr) FIELD is the name of the
                   instruction
      VAL   (cr) field to be changes, VAL is the new
                 integer value.
```

N VAL	(cr) (cr)	Number radix. Set the radix for integer user I/O to VAL, which must be 8 (for octal), 10 (for decimal), or 16 (for hexadecimal).
O VAL	(cr) (cr)	Offset. Sets the base address to which Program Source memory addresses are relative (for user I/O).
Z	(cr)	Zero. Zero out all AP-120B memories and registers. (APSIM only)

A.3 Memory Load/Dump Commands

```
Y         (cr) Yank. Load memory MEM starting at location.
MEM       (cr) LOC from an external data FILENAME.
LOC       (cr)
filename  (cr)

W         (cr) Write. Dump memory MEM starting at location
MEM       (cr) (START) and ending at location (STOP) to
START     (cr) external data FILENAME.
STOP      (cr) MEM can be PS, MD or TM.
filename  (cr)
```

A.4 Accessible Functional Units

AP-120B Functional Units that may be examined or changed with APDBUG:

MEMORIES

			Accessible from		Can 'Y' or 'W'	
Mnemonic	Name	Width	APSIM	HWDBUG	APSIM	HWDBUG
PS	Program Source memory	64	yes	yes	yes	yes
MD	Main Data memory	38	yes	yes	yes	yes
TM	Table memory	38	yes	(read only)	yes	yes
DPX	Data Pad X	38	yes	yes	no	yes
DPY	Data Pad Y	38	yes	yes	no	yes
IO	I/O devices	38	yes	no	no	no
SP	S-Pad	16	yes	yes	no	no
SRS	Subroutine Return Stack	16	yes	no	no	no

REGISTERS

Mnemonic	Name	Width	Accessible from: APSIM	HWDBUG
MA	Main Data Address	16	yes	yes
TMA	Table memory Address	16	yes	yes
DPA	Data Pad Address	6	yes	yes
PSA	Program Source Address	12	yes	yes
SPD	S-Pad Destination Addr.	4	yes	yes
STAT	AP Status Register	16	yes	yes
DA	I/O Device Address	6	yes	yes
SPFN	S-Pad Function	16	yes	yes
SWR	Switch register	16	no	yes
FN	Function register	16	no	yes
LITE	Lites register	16	no	yes
APMA	AP DMA Memory Address	16	no	yes
HMA	Host DMA Memory Address	16	no	yes
WC	DMA Word Count	16	no	yes
CTL	DMA Control register	16	no	yes
FMTH	Formatter High	16	no	yes
FMTL	Formatter Low	16	no	yes
IFRS	Interface reset	16	no	yes
IFSTAT	Interface status	16	no	yes (when present)
MDR	Main Data Read Buffer	38	yes	no
TMR	Table Memory Read Buff.	38	yes	no
MI	Main Data Input Buff.	38	yes	no
DPBS	Data Pad Bus	38	yes	no
INBS	I/O Input Bus	38	yes	no
PNBS	Panel Bus	16	yes	no
FLAG	Program Flags	4	yes	no
SRA	Subroutine Stack Addr.	4	yes	no
A1	Floating Adder #1 input	38	yes	no
A2	Floating Adder #2 input	38	yes	no
M1	Multiplier #1 input	38	yes	no
M2	Multiplier #2 input	38	yes	no
FA	Floating Adder output	38	yes	no
FM	Floating Multiplier out.	38	yes	no

A.5 Program Word Fields

Fields within an instruction word that may be examined or changed by name:

Name	Program Word Bits
D	0
SOP	1-3
SH	4-5
SPS	6-9
SPD	10-13
FADD	14-16
A1	17-19
A2	20-22
COND	23-26
DISP	27-31
DPX	32-33
DPY	34-35
DPBS	36-38
XR	39-41
YR	42-44
XW	45-47
YW	48-50
FM	51
M1	52-53
M2	54-55
MI	56-57
MA	58-59
DPA	60-61
TMA	62-63
SOP1	6-9
SPEC	6-9
STST	10-13
HPNL	10-13
SPSA	10-13
PSEV	10-13
PSOD	10-13
PS	10-13
SEXT	10-13
FAD1	17-19
IO	17-19
LREG	20-22
RREG	20-22
IOUT	20-22
SNSE	20-22
FLAG	20-22
CONT	20-22

APPENDIX B
AN EXAMPLE DEBUGGING SESSION

In this hypothetical sequence an AP-120B program called MYPROG is being debugged. MYPROG uses the FPS supplied divide routine to divide two numbers in Main Data memory.

1. APAL SOURCE PROGRAM

```
        $TITLE MYPROG
        $ENTRY MYPROG,3
        $EXT DIV

"MYPROG DOES A SCALAR DIVIDE OF TWO NUMBERS IN MAIN DATA
"MEMORY AND RETURNS THE ANSWER BACK INTO MAIN DATA MEMORY
"C <= A / B
"
"S-PAD PARAMETER DEFINITIONS:
"
        A $EQU 0          "ADDRESS OF 'A' IN MAIN DATA MEMORY
        B $EQU 1          "ADDRESS OF 'B' IN MAIN DATA MEMORY
        C $EQU 2          "ADDRESS OF 'C' IN MAIN DATA MEMORY
"
MYPROG: MOV A,A; SETMA   "FETCH A
        MOV B,B; SETMA   "FETCH B
        NOP              "WAIT
        DPY<MD           "SAVE A IN DPY
        DPX<MD;          "SAVE B IN DPX
           JSR DIV       "   AND DIVIDE A/B
        MOV C,C; SETMA; "STORE ANSWER IN C
           MI<DPY; RETURN "AND RETURN
        $END
```

The input parameters are the addresses of scalars A, B, and C. A and B are fetched from Main Data memory, A is divided by B, and the result stored into C. The $ENTRY declaration tells APAL that the routine's name is MYPROG. The ",3" tells APAL that the routine is Fortran callable through APEX with has three S-Pad parameters (the addresses of A, B, and C). The $EXT tells APAL that the routine uses the DIV routine.

2. ASSEMBLE USING APAL

```
RUN APAL   (cr)
 SOURCE FILE=
MYPROG.AP   (cr)
 OBJECT FILE=
MYPROG.OBJ   (cr)
 LISTING AND ERROR FILE=
MYPROG.LST   (cr)
 LISTING? 1=YES, 0=NO:
1   (cr)
 LISTING RADIX: 1=HEX, 0=OCTAL:
0   (cr)
     0 ERRORS: MYPROG  APAL-REV 2
```

File MYPROG.AP is assembled with the object going into file MYPROG.OBJ, and the listing (reproduced below) going into file MYPROG.LST.

```
 APAL, REV 2
 PASS 1
 PASS 2
                          $TITLE MYPROG
                          $ENTRY MYPROG,3
                          $EXT DIV

             "MYPROG DOES A CHALAR DIVIDE OF TWO NUMBERS IN MAIN DATA
             "MEMORY AND RETURNS THE ANSWER BACK INTO MAIN DATA MEMORY
             "C <=A / B
             "
             "S-PAD PARAMETER DEFINITIONS:
             "
       000000       A $EQU 0        "ADDRESS OF 'A' IN MAIN DATA MEMORY
       000001       B $EQU 1        "ADDRESS OF 'B' IN MAIN DATA MEMORY
       000002       C $EQU 2        "ADDRESS OF 'C' IN MAIN DATA MEMORY
       000000       A $EQU 0        "ADDRESS OF 'A' IN MAIN DATA MEMORY
       000001       B $EQU 1        "ADDRESS OF 'B' IN MAIN DATA MEMORY
       000002       C $EQU 2        "ADDRESS OF 'C' IN MAIN DATA MEMORY

                 "
 000000  040000  MYPROG: MOV A,A; SETMA  "FETCH A
         000000
         000000
         000060

 000001  040104          MOV B,B; SETMA  "FETCH B
         000000
         000000
         000060

 000002  000000          NOP             "WAIT
```

```
        000000
        000000
        000000

000003  000000          DPY<MD            "SAVE A IN DPY
        000000
        015000
        100000

000004  011014          DPX<MD;           "SAVE B IN DPX
        000000             JSR DIV        "   AND DIVIDE A/B
        045004
        177777

000005  040210          MOV C,C; SETMA; "STORE ANSWER IN C
        000340             MI<DPY; RETURN "AND RETURN
        004040
        000360

                        $END


****      0 ERRORS    ****


SYMBOL  VALUE
DIV     000004 EXT
A       000000
B       000001
C       000002
MYPROG  000000 ENT
```

3. LINK THE PROGRAM USING APLINK

```
RUN APLINK   (cr)
 APLINK
 REV 2
 *
L   (cr)
MYPROG.OBJ   (cr)
 *
L   (cr)
APLIB   (cr)
 LOAD COMPLETE
 *
S   (cr)
TTY   (cr)
 HIGH=000041


 SYMBOL TABLE

 SYMBOL  VALUE
 MYPROG 000000
 DIV    000006
 *
E   (cr)
MYPROG.ABS   (cr)
 MYPROG     HIGH=000041
 *
X   (cr)
```

First MYPROG.OBJ is loaded, then the FPS supplied object library (called APLIB) is loaded in. APLINK picks out DIV from the library. When the loading is done, a symbol table (load map) is printed out on the console. It shows that MYPROG is loaded at location 0, and DIV at location 7. The high location loaded was 41 (octal), which means that the total size of the load module is 34 (decimal) program words. The E command is used to output the load module for APSIM (or HWDBUG). It is put in file MYPROG.ABS.

4. DEBUG THE PROGRAM WITH APSIM

```
RUN APSIM   (cr)
 APSIM
 REV 2.0
 *
Z   (cr)
 *
Y   (cr)
PS   (cr)
0   (cr)
MYPROG.ABS   (cr)
 *
```

Zero the simulator and Yank in the load module.

```
E   (cr)
SP   (cr)
0   (cr)
 000000
 *
C   (cr)
2   (cr)
 *
+   (cr)
 SP    000001
 000000
 *
C   (cr)
3   (cr)
 *
+   (cr)
 SP    000002
 000000
 *
C   (cr)
4   (cr)
 *
```

Set up S-Pads 0, 1, and 2 with the addresses of A, B, and C. Which are chosen here to be 2, 3, and 4, respectively.

```
E   (cr)
MD   (cr)
2   (cr)
  0.0000000000
 *
C   (cr)
10.0   (cr)
 *
+   (cr)
 MD    000003
```

```
  0.0000000000
 *
C   (cr)
2.0   (cr)
 *
```

Set A (in MD 2) to 10.0, and B (in MD 3) to 2.0. The answer should be 5.0.

```
E   (cr)
PS   (cr)
5   (cr)
 040210  000340  004040  000360
 *
B   (cr)
PS   (cr)
6   (cr)
 *
```

Examine PS location 5. Yes, it looks like the last instruction of MYPROG, so set the breakpoint there.

```
R   (cr)
0   (cr)
 PSA=000000       5.00 US.
 PS    000005
 040210  000340  004040  000360
 *
```

Run the program. It took 5.0 us to reach the breakpoint. PSA shows 0 since we stopped on a RETURN, which wants to return to location 0 since the Subroutine Return Stack was zeroed by the 'Z'. PS location 5 is printed out because it is the last location we examined.

```
E   (cr)
MD   (cr)
4   (cr)
   20.00000000
 *
```

But alas, the answer is wrong, we should have 10.0, not 20.0 in C (MD location 4).

```
E   (cr)
DPY   (cr)
0   (cr)
   20.00000000
 *
```

DPY (where we thought DIV returned the answer) also has 20.0, so where's our 10.0. Lets look in DPX, since maybe we have remembered wrongly which Data Pad DIV returns the answer in.

```
E    (cr)
DPX    (cr)
0    (cr)
   5.000000000
 *
```

Sure enough, the answer is in DPX instead of DPY. A quick check of the AP-120B Math Library Manual (#7288-03, part 2) confirms that indeed, DIV returns the result in DPX, and that DPY is used as a scratch location. We now patch the program so that in PS location 5, MI<DPX instead of MI<DPY.

```
E    (cr)
PS    (cr)
5    (cr)
 040210  000340  004040  000360
 *
V    (cr)
1    (cr)
 *
 PS     000005
    D     00    SOP  04    SH    00    SPS  02    SPD  02    FADD 00
    A1    00    A2   00    COND  07    DISP 00    DPX  00    DPY  00
    DPBS  04    XR   00    YR    04    XW   00    YW   00    FM   00
    M1    00    M2   00    MI    03    MA   03    DPA  00    TMA  00
 *
C    (cr)
DPBS    (cr)
3    (cr)
 *
C    (cr)
XR    (cr)
4    (cr)
 *
 PS     000005
    D     00    SOP  04    SH    00    SPS  02    SPD  02    FADD 00
    A1    00    A2   00    COND  07    DISP 00    DPX  00    DPY  00
    DPBS  03    XR   04    YR    04    XW   00    YW   00    FM   00
    M1    00    M2   00    MI    03    MA   03    DPA  00    TMA  00
 *
```

Examine PS 5 and switch the PS output mode (V) to 1 for PS opcode field mode. It shows that now the MI field (Memory Input) is set to 3 for Data pad Bus, and that the DPBS field is set to 4 for DPY. We then change the DPBS field to 3 for DPX, and the DPX read address (XR) to 4, which biased by 4 gives the proper DPX address of 0. Re-examining shows that the patches were made correctly. We now reinitialize APSIM and run again.

```
I    (cr)
 *
R    (cr)
0    (cr)
```

```
 PSA=000000        5.00 US.
 PS     000005
    D     00      SOP  04     SH    00     SPS  02     SPD  02     FADD 00
    A1    00      A2   00     COND  07     DISP 00     DPX  00     DPY  00
    DPBS  03      XR   04     YR    04     XW   00     YW   00     FM   00
    M1    00      M2   00     MI    03     MA   03     DPA  00     TMA  00
 *
E   (cr)
MD   (cr)
4   (cr)
   5.000000000
 *
```

And now we have the correct answer in C.

```
-   (cr)
 MD     000003
   2.000000000
 *
C   (cr)
-3.4   (cr)
 *
-   (cr)
 MD     000002
   10.00000000
 *
C   (cr)
12.5   (cr)
 *
```

We now try a different case: B = -3.4, and A = 12.5, expected answer about -3.68.

```
I   (cr)
 *
R   (cr)
0   (cr)
 PSA=000000        5.00 US.
 MD     000002
   12.50000000
 *
E   (cr)
MD   (cr)
4   (cr)
  -3.676470578
 *
```

Re-initializing and re-running, we again get the correct answer.

X (cr)

So we exit APSIM, edit the change into the source, and re-assemble, link, and simulate to make sure things are correct.

FLOATING POINT
SYSTEMS, INC.

CALL TOLL FREE 800-547-1445
P.O. Box 23489, Portland, OR 97223
(503) 641-3151, TLX: 360470 FLOATPOINT PTL