Software Reference Manual

HDOS SYSTEM

Chapter 3

CONSOLE DEBUGGER DBUG

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

595-2476
Printed in the United States of America

TABLE OF CONTENTS

INTRODUCTION

The Heath Console Debugger, DBUG, allows you to enter the debug machine-language programs from a console terminal. DBUG occupies the lowest 4K of user program RAM area, starting at 042 200. A user program can be loaded into any free RAM (random access memory) location, and can be manipulated via DBUG. Note that your program MUST be ORGed above the end of DBUG. (See "Memory Layout" in Chapter 1.)

DBUG contains facilities to perform the following nine major functions:

- Display the contents of a selected memory location.
- Alter the contents of a selected memory location.
- Display the contents of any 8080 compatible register.
- Alter the contents of any 8080 compatible register.
- Execute the user program a single instruction at a time.
- Execute the program.
- Insert breakpoints and execute the user program.
- Load user programs from a device.
- Dump user programs to a device.

A number of features were designed into DBUG for your convenience. Memory locations and memory and register contents may be displayed as bytes or as words, in octal, decimal, or ASCII format. With these features, you can select the most familiar or desirable format. DBUG also contains a single-instruction facility that permits you to execute your program a single instruction at a time. And for more advanced program analysis, a breakpointing feature is included that permits you to execute several instructions in a program and then return control to DBUG for analysis and/or modification.

DBUG makes use of the console facilities of HDOS; therefore, the HDOS console control conventions, CTRL-S, CTRL-Q, CTRL-O, CTRL-P, etc., also apply to DBUG. DBUG does not respond to CTRL-Cs. CTRL-A is used to return to DBUG command mode. This is done so the program being debugged can make use of CTRL-B and CTRL-C.

When it is accepting commands from the console keyboard, DBUG uses the "command completion" technique. As each character is entered, it is checked against a list of all possible commands. If the character could not be a part of any valid command, DBUG will refuse it by echoing an ASCII BELL character. In addition, if DBUG determines that there is only one choice for the next character in the command, DBUG will type that character for you. Thus, if you strike the L key, DBUG knows that all commands that start with L start with the word LOAD, and will print the entire word LOAD on the terminal.

DISK LOADING/DUMPING

DBUG offers two commands for program loading and dumping (or saving). With these commands, described below, you can load or dump an absolute binary program.

LOAD <fspec>

The LOAD command causes the contents of the file specified by <fspec> to be loaded into memory. The file must be in absolute binary format (the format generated by the HDOS assembler). Note that the absolute binary file contains information to tell HDOS where the file must be loaded. DBUG will not allow you to load a program over the DBUG code, or over the HDOS operating system. The entry point address for the loaded program will be entered into the program counter (Pc register) automatically. For example:

```
LOAD△SY1:TEST.ABS ↵
```

DUMP <fspec> saddr-eaddr

The DUMP command causes the contents of memory from saddr to and including eaddr to be written to the file <fspec> in absolute binary format. The contents of the Pc register at the time of the DUMP are stored in the file as the program's entry point. You can use this feature to save a patched binary program without reassembling it. For example, to save the demo program used on Page 3-13, type:

```
DUMP△TEST△70000-70026 ↵
```

Executing DBUG

DBUG is called by the operating system (HDOS) as follows:

```
>DBUG ↵

HDOS DBUG # 102.00.00.

:B:
```

Note that the version number may not be the same as yours, but a number will be shown.

MEMORY COMMANDS

The memory commands permit you to display and alter the contents of indicated memory locations. The format for memory display commands is:

```
< FORMAT CONTROL >    < range >    < blank >
```

The form for the alter memory command is:

```
< FORMAT CONTROL >    < range > = < value list >
```

Format control specifies that memory display/alteration is in word or byte format, and whether octal, decimal or ASCII notation is to be used. The range specifies the memory address or addresses to be displayed or altered, and the command is executed by the typing of a blank using the space bar on the console terminal.

The Format Control

The format control consists of two characters which specify the form of the values that are to be displayed and entered. The format control field may take on a number of different forms. They are:

FORMAT CONTROL	DESCRIPTION
< null > < null >	Display/alter as octal integers, byte format.
F < null >	Display/alter as octal integers, word format.
< null > A	Display/alter as ASCII characters, byte format.
FA	Display/alter as ASCII characters, word format.
< null > D	Display/alter as decimal integers, byte format.
FD	Display/alter as decimal integers, word format.

WORD FORMAT (F)

If an F is specified as the first character of the format control field, it indicates that the values are to be displayed/altered as "full words." This is to say that memory locations are taken as two-byte pairs. The second byte is considered to be the high-order (most significant) byte and is displayed first. The first byte is considered to be the low-order (least significant) byte and is displayed last.

BYTE FORMAT (NULL)

If an F is not specified, the first character is null, indicating that the values are to be displayed/altered as single bytes. You can create a NULL by not typing any character for the format control portion of the memory command.

OCTAL FORMAT (NULL)

If no option (a NULL) is specified as the second character of the format control field, the values to be displayed/altered are taken to be octal integers. The NULL was chosen to specify both byte format and octal notation, as byte octal is the most commonly used format. A blank separates each octal integer, or octal integer pair if the F is specified.

DECIMAL FORMAT (D)

If a D is specified as the second character of the format control field, the values to be displayed/altered are taken to be decimal integers. A blank separates each decimal integer, or decimal integer pair if the F is specified.

ASCII FORMAT (A)

If an A is specified as the second character of the format control field, the values to be displayed/altered are converted from/to eight-bit representations of ASCII characters. A blank separates each character, or character pair if the F is specified.

Range

The range field consists of a beginning address and an ending address. You can specify addresses by using the appropriate offset octal integers; or you can use the NULL, #, and cnt (count) as indicated below.

RANGE FORM	DESCRIPTION
ADDR < null >	Range specifies the single memory location ADDR.
ADDR1-ADDR2	Range specifies the memory locations ADDR1 through ADDR2, inclusive.
ADDR/cnt	Range specifies cnt memory locations starting at location ADDR. NOTE: cnt is a decimal integer ≤ 255.
#-ADDR	Range specifies the memory locations starting at the beginning of the previous range and ending at ADDR.
#/cnt	Range specifies cnt memory locations starting at the beginning or the previous range. NOTE: cnt is a decimal integer ≤ 255.
< null>/cnt	Range specifies cnt memory locations starting at the address following the last address of the previous range. NOTE: cnt is a decimal integer ≤ 255.
< null > -ADDR	Range specifies memory locations starting at the address following the last address of the previous range and extending to memory location ADDR.

For example, to display memory location 000 043 through 000 047, DBUG simply requires the user to type 43-47 followed by a blank (a blank is generated by using the console terminal space bar). For example:

```
:B:43-47△100 112 107 114 100
:B:B

:B:/4△303 053 040 365
:B:
```

NOTE: In the first example, the contents of memory locations 000 043 through 000 047 are displayed on the first line in octal byte format. The next four bytes (locations 000 050 to 000 053) are displayed when the command /4 is typed. The contents of these next four bytes are displayed as soon as a blank is typed after the /4.

If the first address specified is greater than the second address specified, an error message is generated. The form of the error message is:

```
LWA<FWA
```

For example:

```
:B:47-43△LWA<FWA

:B:
```

NOTE: If you attempt to enter a numerical address which does not fit the offset (split) octal format, DBUG rejects the improper entry and sounds the console terminal bell. For example, the number 067777 does not fit the offset octal format; therefore, DBUG does not allow the second 7 to be entered.

Displaying Memory Contents

To display the values in the specified range and in the specified format, type a blank following the format and range fields. DBUG immediately executes the command. In the following examples, the contents of a number of locations, 002 143 to 002 163 in the Monitor ROM, are displayed in octal byte format, in octal word format, in decimal byte format, and in decimal word format. NOTE: When all the bytes or words in the specified range cannot be displayed on the line, a new line is started. DBUG supplied the starting address of the new line.

```
:B:2143-2163△343 353 041 011 040 256 136 167 056 033 172 206 276 302
002162  160 002
:B:F2143-2163△325343 041353 040011 136256 172033 276206 160302 303002
:B:D2143-2163△227 235 033 009 032 174 094 119 046 027 122 134 190 194
002162  112 002
:B:FD2143-2163△54755 08683 08201 24238 11895 31259 48774 28866 49922
```

Note that you may type CTRL-A to short a memory display.

For example:

```
:B:30000-60000△303 014 037 041 300 377 071 353 041 100 040 166 042 076 ^A
:B:
```

Altering Memory — Decimal or Octal

To alter memory in decimal or octal formats, type an = after the format control and range fields. DBUG will then type the value of the first byte, or double byte if an F was used in the format control and follow this with a /. You can then type a new value if you want to change the contents of this location. If the contents of the location are not to be changed, or if sufficient new digits have been entered to complete the change, type a space or a carriage return.

If you type a space, DBUG offers the next byte (if there is one in the range) for alteration. If you type a carriage return, DBUG returns to the command mode.

In the following example, memory locations 60000 through 60031 are loaded with the octal values of the ASCII characters A through Z. NOTE: On the first three lines, the initial address is followed by the = sign, the current octal value in that memory location, and then a /. The current octal value may be different from that shown in the example. The octal value for the letter is entered following the slash. On the successive lines, a range of successive locations are opened and then changed to the sequentially ascending ASCII characters.

After the letters have been entered, the 26 memory locations are examined in byte format as ASCII characters. The 26 locations are then examined in word format as ASCII characters. Note that the second byte is treated as the most significant byte. Finally, the 26 locations are opened in byte octal format, using the # as the first address of the range.

```
:B:60000=324/101 ↵
:B:60001=030/102 ↵
:B:60002=353/103 ↵
:B:60003/23=341/104△330/105△203/106△137/107△076/110△000/111△212/112△127/113△
060013  322/114△365/115△057/116△311/117△315/120△072/121△030/122△345/123△
060023  365/124△345/125△021/126△012/127△000/130△315/131△106/132△
:B:

:B:A60000/26△A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
:B:FA#/26△BA DC FE HG JI LK NM PO RQ TS VU XW ZY
:B:#-60031△101 102 103 104 105 106 107 110 111 112 113 114 115 116 117 120
060020  121 122 123 124 125 126 127 130 131 132
:B:
```

The DELETE (or RUBOUT) key is not effective when you are entering memory locations. Values placed in memory are taken as modulus 256 numbers (if they are entered in byte format) or as modulus 65,535 numbers (if they are entered in full word format). Thus, if you make a mistake, simply type the correct value with enough leading zeros to cause the bad digit to be eliminated. For example, if byte 70,000 is to be set to 123 and the mis-type 125 occurs, it may be correctly entered as:

```
:B:70000=111/1250123 ®
:B:70000=123/ ®
```

NOTE: Only the three least significant digits are accepted for this byte location.

Altering Memory — ASCII Format

To alter memory in ASCII format, type an = after the format control (A for ASCII) and range fields. The processing is similar to decimal or octal format memory alterations. The contents of the opened locations should then be followed by a /. You can then enter the replacement character (or two characters if the word format is used). However, the space and the carriage return are considered to be ASCII character values. To exit the command prematurely, use the ESCape or CTRL-A key to avoid altering a location.

```
:B:A70000= /A
:B:A70001= /B
:B:A70002= /C
:B:A70003-70031= /D /E /F /G /H /IT/JT/K /L /M /N7/O /P /Q /R /S /T /U /V
070026    /W /X /Y /Z
:B:A70000-70031△A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
:B:A70000/26△A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
:B:
```

REGISTER COMMANDS

DBUG permits you to display the contents of all registers using octal, decimal, or ASCII, or to display the contents of individual registers using octal, decimal, or ASCII. In addition to displaying the contents of these registers, you can alter the various registers in any of the three modes. NOTE: If the F command is used in the format field, a register command is rejected, as register size is predetermined.

Displaying All Registers

To display the contents of all registers, enter a command of the form.

```
<FORMAT> <CTRL-R>
```

DBUG displays the resister contents in a specified format. NOTE: An M register is displayed in the ALL REGISTERS command and can be specified in other commands. This register is the concatenation of the H and L registers. For example:

```
:B:<CTRL-R>
A=000 B=000 C=001 D=000 E=004 H=070 L=100 F=203 P=070005 M=070100 S=042200
:B:

:B:D<CTRL-R>
A=000 B=000 C=001 D=000 E=004 H=056 L=064 F=131 P=14341 M=14400 S=08832
:B:

:B:A<CTRL-R>
A=  B=  C=  D=  E=  H=8 L=@ F=  P=8  M=8@ S="
:B:
```

A Control R (CTRL-R) should be typed after each command. However, no character is actually displayed. Also note that the ASCII display is not particularly meaningful unless printing ASCII characters are contained in the desired registers.

Displaying Individual Registers

To display the contents of any single register, use a command in the following format:

```
< FORMAT > REG < REG-NAME > < blank >
```

For example, to display the contents of register A, type:

```
:B:REGA△=101
:B:DREGA△=065
:B:AREGA△=A
:B:
```

In the above example, the first line calls for the contents of register A to be displayed in octal format. In the second line, the contents of register A are displayed in the decimal format, and in the third line, the contents of register A are displayed in ASCII format. In the following example, the contents of the 16-bit register pair H and L, known as the M or memory register, are displayed in octal format.

```
:B:REGM△=041031
:B:REGH△=041
:B:REGL△=031
:B:
```

Altering Register Contents

To alter the contents of a register, use a command in the following format:

```
< FORMAT > REG < REG-NAME > =
```

DBUG will then display the previous contents of the register (in the specified format octal, decimal, or ASCII), followed by a /. It then accepts a new value if one is typed in. When you are using octal or decimal format, use a carriage return to close the entry or to skip the change. When you are using the ASCII format, type a single ASCII character to close the register. However, as the carriage return is a valid ASCII character, you must use ESCAPE or CTRL-A to skip the change. The following examples demonstrate the altering of register contents.

```
:B:REGA=102/103 ↵      (Change contents of A from 102₈ (ASCII B) to 103₈ (ASCII C).
:B:DREGA=067/066 ↵     (Change contents of A from 67₁₀ (ASCII C) to 66₁₀ (ASCII B).
:B:AREGA=B/C           (Change contents of A from ASCII B to ASCII C.)
:B:

:B:REGA=103/ ↵         (A carriage return skips the change.)
:B:AREGA=C/<CTRL-A>    (A CTRL-A skips the change.)
:B:AREGA△=C            (The location is unaltered.)
:B:
```

NOTE: The last three are examples of skipping the change (leaving the location unaltered).

EXECUTION CONTROL

One of the primary functions of DBUG is execution control. It lets you step through the program, one or more instructions at a time, while examining register and memory contents. In addition, complete breakpointing is available, permitting you to execute a number of instructions and then return to DBUG control to examine register and memory contents. You may also stop your program execution at any time by typing CTRL-A, which will cause control to return to DBUG. Execution control is divided into the areas of single stepping, breakpointing, and the GO command.

Single Stepping

The form of the single step command is:

```
STEP ADDR/CNT
```

where ADDR is an offset octal address (or a null) and "CNT" is a decimal step count, ≤ 255. If an address is not specified, DBUG starts stepping at the current PC-register address. When the instructions are completed, DBUG types the PC-register value and returns to the command mode. If an address is specified, DBUG starts stepping at the specified address and, when the instructions are completed, displays the terminating address value before returning to the command mode.

The following program increments the contents of memory location 070 100 each time the BC register pair is incremented from 000 000 to 027 000. This program is used to demonstrate a number of the execution control features of DBUG.

```
ADDRESS              LABEL  INSTRUCTION          COMMENT

070.000                     START  ORG   070000A
070.000  041 100 070       L1     LXI   H,070100A     POINT HL TO 070100
070.003  066 000                  MVI   M,000         LOAD MEMORY WITH ZERO
070.005  003               L2     INX   B             INCREMENT BC PAIR
070.006  170                     MOV   A,B           LOAD A WITH B
070.007  376 027                 CPI   027Q          IS B 027 OCTAL?
070.011  302 005 070             JNZ   L2            JUMP BACK IF NOT
070.014  064                     INR   M             INCREMENT MEMORY
070.015  176                     MOV   A,M           LOAD A WITH MEMORY
070.016  376 377                 CPI   377Q          IS MEMORY 377 OCTAL?
070.020  006 000                 MVI   B,000         LOAD B WITH ZERO
070.022  302 005 070             JNZ   L2            JUMP IF NOT ZERO
070.025  327                     RST   2
070.026  000                     END   START
```

NOTE: The RST2 instruction is used to return this program to DBUG. When the CPU encounters an RST2 instruction, it returns to DBUG.

For example, to load the above program using DBUG,

```
:B:70000-70025=101/041△102/100△103/070△104/066△105/000△106/003△107/170△110/376△
111/027△112/302△113/005△114/070△115/064△116/176△117/376△120/377△121/006△
070021  122/000△123/302△124/005△125/070△126/327△
:B:

:B:REGB=302/000 ⏎
:B:REGC=110/000 ⏎
:B:STEP 70000/6 ⏎

-P=070005-

:B:
```

DBUG returns the value of the PC once the first six steps are executed.

Breakpointing

DBUG contains several commands to set, display, and clear breakpoints in your program. Breakpointing permits you to execute portions of a program once (or a number of times if the portion of a program is in a loop). Breakpointing is especially useful in de-bugging programs which have a tendency to destroy themselves or obliterate the cause of the problem in the process of complete execution.

SETTING BREAKPOINTS

The breakpoint command is used to set a breakpoint. The form of the breakpoint command is:

```
BKPT ADDR1/CNT1, . . . . . ,ADDRn/CNTn
```

DBUG allows up to 8 breakpoints. They are entered in the breakpoint table within DBUG, replacing any previously defined breakpoints at those addresses. No more than eight breakpoints may be entered in the breakpoint table.

The CNT field may be used to specify the breakpoint repeat count. It is a decimal number in the range of 1 to 255. Using the breakpoint count means the breakpoint does not cause control to return to the monitor mode until the breakpoint is executed CNT-1 times. Thus, you may execute a loop a number of times prior to returning to the command mode via a breakpoint instruction. As noted, the Breakpoint Instruction executes CNT-1 times, without recognizing the break-

point. The last time through the loop, the instruction at the breakpoint address is not executed. The breakpoint returns control to DBUG. NOTE: If CNT is not specified, the value 1 is assumed.

For example, the program of the previous example is run with breakpoints.

```
:B:70100=000/ CR
:B:BKPT 70015/6 CR

:B:GO 70000 CR

-P=070015-

:B:70100=006/ CR
:B:
```

NOTE: 070 100 is incremented by 6.

```
:B:70100=006/000 CR
:B:BKPT 70015/6,70014/10,70022/30 CR

:B:GO 70000 CR

-P=070015-

:B:GO CR

-P=070014-

:B:GO CR

-P=070022-

:B:
```

DISPLAYING BREAKPOINTS

To display the current status of the breakpoint table, use the breakpoint display command. DBUG can display the contents of the breakpoint table. The form of the breakpoint command is:

```
BKPT DSPLY
```

DBUG provides a listing of the current breakpoints in the form:

```
BKPT DSPLY ADDR1/CNT1,ADDR2/CNT2, . . . . . ,ADDRn/CNTn
```

where ADDR is the address of the breakpoint instruction, and CNT are the loop counts remaining on the designated breakpoints. NOTE: When the breakpoint

count is exhausted, it causes control to return to DBUG. The exhausted breakpoint is removed from the breakpoint table; nonexhausted breakpoints remain. For example:

```
:B:70100=036/000 CR
:B:BKPT 70015/6,70014/10,70022/30 CR

:B:BKPT DSPLY 070015/006 070014/010 070022/030
:B:GO 70000 CR

-P=070015-

:B:BKPT DSPLY 070014/004 070022/025
:B:GO CR

-P=070014-

:B:BKPT DSPLY 070022/021
:B:GO CR

-P=070022-

:B:BKPT DSPLY
:B:
```

CLEARING INDIVIDUAL BREAKPOINTS

To clear an individual breakpoint, use the command

```
CLEAR ADDR1, . . . ,ADDRn
```

where ADDR1, . . . ,ADDRn specifies the address of the breakpoint to be removed from the table.

CLEARING ALL BREAKPOINTS

To clear all breakpoints from the breakpoint table, use the breakpoint clear command

```
CLEAR ALL
```

For example:

```
:B: BKPT 55012/10,55014/15,55020/20,55022/200 CR
:B: BKPT DSPLY 055012/010 055014/015 055020/020 055022/200
:B: CLEAR 55014,55022 CR
:B: BKPT DSPLY 055012/0/0 055020/020
:B: CLEAR ALL CR
:B: BKPT DSPLY
:B:
```

GO Command

Use the GO command to transfer control to your program. You can set breakpoints before via the BKPT command. The form of the GO command is:

```
GO [SADDR]
```

If you specify "SADDR," execution begins at this specified address. If you do not specify "SADDR," execution begins at the current value of the program counter register. For example, simple execution of the previous program is accomplished by

```
:B:GO ↵
-P=070025
:B:
```

EXEC

The EXEC (execute) command is a combination of the GO and BKPT commands. The form of the EXEC command is:

```
EXEC SADDR-ADDR1, . . . . ,ADDRn
```

where "SADDR" is the starting address for execution. If the starting address is omitted, execution starts at the current program counter register value. ADDR1 through ADDRn are the addresses of breakpoints to be set before execution. Thus, for example, to start at byte 070 000 and to execute to byte 070 015, the command is typed as:

```
:B: EXEC 70000-70015 ↵
-P=070015-
:B:


:B: GO 70210 ↵
↑A              (CTRL-A stuck)
-P=072121-
:B:
```

CTRL-A

When you are executing your program via EXEC or GO, you may return to DBUG by typing CTRL-A. This is useful when you fail to set a breakpoint, or fail to reach the ones you have set. For example:

```
:B:GO 70210 CR
^A              (CTRL-A stuck)
-P=072121-
:B:
```

CTRL-D

When you are finished with the DBUG program, you can return to HDOS by typing CTRL-D. The program will respond "ARE YOU SURE?" You must respond with a "Y" to exit. For example:

```
B:^D CR
ARE YOU SURE? Y
```

COMMAND COMPLETION

When DBUG is in the command mode, each terminal keystroke is considered for validity. If the character belongs to no possible command, it is refused and the bell code is echoed to the terminal. If the command syntax allows only one next character, DBUG supplies and prints this character for the user.

```
:B:^D CR
ARE YOU SURE? Y

>
```

APPENDIX A

Error Messages

The following error messages are generated by DBUG. In addition to these errors, other errors may be detected by the operating system itself. These errors are discussed in Appendix B of the HDOS Manual.

`<BELL>`

The console terminal's bell is sounded when you type an illegal character (for the current command). DBUG sends the ASCII bell code to the terminal instead of echoing the illegal character.

`LWA < FWA`

The second address specified in this command must be the same as or larger than the first address specified.

`FORMAT ERROR IN FILE`

The file you have attempted to load is not of the proper type. The LOAD command will only load absolute binary files.

`ATTEMPT TO LOAD OVER DBUG`

A file may not be loaded into the same memory locations occupied by DBUG.

`NO ROOM`

Attempt to specify more breakpoints than DBUG has room for. DBUG currently allows a maximum of eight breakpoints to be simultaneously specified.

APPENDIX B

DBUG Command Summary

Memory Commands

Memory display form:

```
FORMAT CONTROL    range  blank
```

Memory Alter form:

```
FORMAT CONTROL    range  =
```

FORMAT	RESULT
< null > < null >	byte octal
F < null >	word octal
< null > A	byte ASCII
FA	word ASCII
< null > D	byte decimal
FD	word decimal

Range

The range field consists of a beginning address and an ending address. You can specify addresses by using the appropriate offset octal integers; or you can use the NULL, #, and cnt (count) as indicated below.

RANGE FORM	DESCRIPTION
ADDR < null >	Range specifies the single memory location ADDR.

ADDR1-ADDR2	Range specifies the memory locations ADDR1 through ADDR2, inclusive.
ADDR/cnt	Range specifies cnt memory locations starting at location ADDR. NOTE: cnt is a decimal integer ≤ 255.
#-ADDR	Range specifies the memory locations starting at the beginning of the previous range and ending at ADDR.
#/cnt	Range specifies cnt memory locations starting at the beginning of the previous range. NOTE: cnt is a decimal integer ≤ 255.
< null >/cnt	Range specifies cnt memory locations starting at the address following the last address of the previous range. NOTE: cnt is a decimal integer ≤ 255.
< null > -ADDR	Range specifies memory locations starting at the address following the last address of the previous range and extending to memory location ADDR.

Register Commands

All registers:

```
FORMAT CTRL-R
```

Single register:

```
REG REG-NAME blank
```

Altering register:

```
REG REG-NAME=
```

Execution Control

Single stepping:

```
STEP ADDR/cnt
```

Breakpointing:

```
BKPT ADDR1/cnt1, . . . . . ,ADDRn/cntn
```

Breakpoint display:

```
BKPT DSPLY
```

Clearing breakpoints:

```
CLEAR ADDR1, . . . . . ,ADDRn
CLEAR ALL
```

GO:

```
GO (ADDR)    (Starts at PC value if ADDR is not specified)
```

Execute:

```
EXEC SADDR-ADDR1, . . . . . ,ADDRn    (combines GO AND BKPT)
```

Program Loading and Dumping

```
LOAD <fspec>
DUMP <fspec>,saddr-eaddr
```

INDEX