

NAKED MINI® 4

DEBUGGING MONITOR

REFERENCE MANUAL

93015-90

AUGUST 1979



ComputerAutomation®

Engineering Reference Number
90-93015-00D0

NAKED MINI Division

HERTFORD HOUSE
DENHAM WAY, MAPLE CROSS
RICKMANSWORTH, HERTFORDSHIRE WD3 2XJ

Telephone Rickmansworth 71211

Telex 9226

REVISION HISTORY

<u>Revision</u>	<u>Date</u>	<u>Comments</u>
A2	Jul 1977	Original issue
A3	Sep 1977	Miscellaneous revisions
B0	Feb 1978	Adds MM, MC, ML, and T commands
B1	Dec 1978	Miscellaneous revisions
C0		Skipped for configuration control.
D0	Aug 1979	Adds Y and Q commands and iteration count option; miscellaneous revisions. The information in the manual corresponds to software released at revision level D1.

Computer Automation, Inc., reserves the right to make changes in specifications at any time and without notice. The information furnished by Computer Automation, Inc., in this publication is believed to be accurate and reliable. However, no responsibility is assumed by Computer Automation, Inc., for its use; nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of Computer Automation, Inc.

Computer Automation, Inc., welcomes comments and suggestions for improvements to our manuals. Should difficulty be encountered with the information in the manual or if you have a suggestion for manual improvement, please send the comments or suggestions to Computer Automation, Inc., Technical Publications, Department 11-0715, 18651 Von Karman, Irvine, California, 92713. Please provide as much information as possible to help evaluate the comment or suggestion.

CONTENTS

SECTION 1	INTRODUCTION	1-1
SECTION 2	COMMUNICATION WITH DEBUG4	2-1
SECTION 3	DEBUG4 COMMANDS	3-1
	3.1 DISPLAY/CHANGE USER REGISTERS (R)	3-2
	3.2 DISPLAY/CHANGE GENERAL REGISTERS (G)	3-4
	3.3 SET BREAKPOINTS (B)	3-4
	3.3.1 Temporary Breakpoints	3-4
	3.3.2 Dummy Breakpoints	3-7
	3.4 JUMP TO USER PROGRAM (J)	3-7
	3.5 INSPECT/MODIFY MEMORY LOCATIONS (I)	3-8
	3.6 LIST CONSECUTIVE MEMORY LOCATIONS (L)	3-10
	3.7 FILL MEMORY WITH A CONSTANT VALUE (F)	3-11
	3.8 COPY ONE AREA OF MEMORY TO ANOTHER (C)	3-11
	3.9 SEARCH MEMORY (S)	3-11
	3.10 PRINT CHAIN STRUCTURE (Z)	3-12
	3.11 ASSIGN DEVICES (A)	3-13
	3.12 PUNCH OR VERIFY MEMORY	3-14
	3.13 LOAD FROM PAPER TAPE (W)	3-15
	3.14 MONITOR BUS TRANSACTIONS (MM, MC, ML)	3-15
	3.14.1 Bus Monitor Mode (MM)	3-16
	3.14.2 Trace Conditions (MC)	3-17
	3.14.3 Bus Monitor Trace Display (ML)	3-18
	3.15 INTERCEPT TRAPS (T)	3-20

APPENDIX	DEBUG4 PROGRAM AND COMMAND SUMMARY	A-1
	A.1 PROGRAM SUMMARY	A-1
	A.2 COMMAND SUMMARY	A-2

FIGURES

<u>Figure</u>		<u>Page</u>
<u>3-1</u>	Processor Status Register	<u>3-3</u>

TABLES

<u>Table</u>		
<u>2-1</u>	Register Display Events	2-2

SECTION 1

INTRODUCTION

The NAKED MINI@ 4 Debugging Monitor is a program that provides for on-line checkout and testing of user software. The Debugging Monitor resides in one area of memory while a user program (or a group of programs) resides in another area. Control of the machine passes back and forth from the Debugging Monitor to the user program, as determined by commands from the operator. When the Debugging Monitor has control, the operator may issue commands to inspect, print, modify, or execute any location in memory.

The Debugging Monitor saves all of the user program's hardware registers each time it gains control and restores them when the operator directs the Monitor to return control to the user program. The operator can examine and modify the contents of each register as desired; there rarely is any need to use the console once the Debugging Monitor is available in memory.

There are three versions of the Debugging Monitor: DEBUG4, XDEBUG4, and MDBUG4. DEBUG4 is the standard debugging monitor. XDEBUG4 is the extended version of the program including the Bus Monitor (M) and Intercept Traps (T) Commands. MDBUG4 is a mini version of the program and does not include the chain command (Z), binary loader (W), dumper (D), device assignment (A), traps (T), and Bus Monitor (M). In this document, the name DEBUG4 refers to the Debugging Monitor generically, not only to the standard version. Variations between DEBUG4, XDEBUG4 and MDBUG4 are identified where applicable.

In the examples presented in this manual, information generated by DEBUG4 is underlined; information entered by the user is not. Lower case letters (e.g., "vvvv") represent hexadecimal values.

SECTION 2

COMMUNICATION WITH DEBUG4

When control is transferred to the origin address of DEBUG4, the bell rings and the program prints its program name (DEBUG4, XDEBUG4 or MDEBUG4), version number, origin address, and end address. Both addresses are in hexadecimal. DEBUG4 then generates a prompt character (an asterisk) to indicate that it is ready for a command. Each command causes a specific action to occur immediately, and each action is followed by another prompt character (*).

A command line consists of a unique Command Identifier, immediately followed by the appropriate parameters, separated by commas. A command line must be terminated with a carriage return except where noted to the contrary. Blanks are not allowed in a command line unless the first character following the prompt character (*) is itself a blank, in which case the whole line is treated as a comment only.

When the operator enters an incorrect command or an unacceptable parameter, DEBUG4 rejects the line with a back-arrow or underscore, and displays the prompt character (*) again. The operator can enter a back-arrow to cancel a line. Input is not buffered; each field of input is processed as it is entered.

The prompt character (*) usually appears in the second position on the teletypewriter (TTY) line. The first position is blank, except when a command alters the contents of memory; in this case, DEBUG4 issues a carriage return without a line feed, types an asterisk at the beginning of the line, and then issues the line feed. The result is a permanent record of the command lines that modified the user program. When the modified program is punched for future use rather than re-assembled, the DEBUG4 listing is the only documentation of the new program.

When one of the events listed in Table 2-1 occurs, DEBUG4 regains control and displays all of the user program's registers along with a character that indicates why DEBUG4 has regained control from the user program. The display is in the following form:

#	P=pppp	A=aaaa	Q=qqqq	X=xxxx	Y=yyyy
	K=kkkk	L=llll	S=ssss	D=dddd	C=cccc

The pound sign (#) in the above display represents the character that indicates the event that just occurred. The display lists the values of the P, A, Q, Y, K, L, and X registers at the time control passed to DEBUG4, and the user program values for the Processor Status Word (S), the Console Sense Switch (C), and the Console Data Register (D).

Table 2-1 shows the events that cause the above display to occur. Each of these events restore (with original contents) all of the DEBUG4 breakpoints which might have been established the last time DEBUG4 was in control.

The operator may effect register display/change via the R command (See Subsection 3-1).

Table 2-1. Register Display Events

<u>Character That Occurs in the Register Display</u>	<u>Event</u>	<u>Command Used to Return Control to DEBUG4</u>
A	Arithmetic Overflow	T
B	Temporary Breakpoint	B,J,Y or Q
C	Console Interrupt	T
I	Unimplemented Instruction	T
U	Uninstalled Memory	T
N	Character/Numeric Error	T
M	Bus Monitor	MM,MC
S	Stack Overflow	T
D	Dummy Breakpoint	
Blank	R Command	

SECTION 3

DEBUG4 COMMANDS

A DEBUG4 command usually involves the specification of a memory address and some data values. Any address or value can be entered as a signed or unsigned hexadecimal number. Leading zeroes are optional. The following are examples of address or value entries:

7FFF 0E08 +50 0

The sum or difference of any number of terms can be used. For example:

5555+4 -345-2C 3900-1+8000

Thirty-six registers, called General Registers (described in Subsection 3.2), are available for holding values. In any context where a hexadecimal number is valid, a General Register reference, which is the letter "G" followed by one digit from 0 to 9 and A to Z, is also valid. The reference can be immediately preceded by another reference, or by a hexadecimal number, as an additional value:

G4 600G2 -4G1 22G7G4G3 400G5-G6

Hardware register contents (described in Subsection 3.1) are saved whenever control is passed to DEBUG4. These user registers also may be referenced in the data parameter expression if they are preceded by the letter "R". For example,

600RA -22G7RQ 400G5- RA+6G0

A minus symbol (-) in a hex number which precedes a General Register or hardware register reference is a binary subtraction operator rather than a unary minus. For example, if G1 contains :104, the command:

*I-4G1

inspects location :FEF8 (-4-104), not location :100.

To inspect location :100 (four prior to G1), the user should enter

*IG1-4

In DEBUG4 commands that can specify more than one memory address, the default value of the first address is zero when a comma is entered immediately after the command (rather than an address). Two examples follow:

- S,1000,FFFF is equivalent to S0,1000,FFFF (i.e., search location :0 to location :1000 for all occurrences of :FFFF).
- L,100 is equivalent to L0,100 (i.e., list consecutive memory locations :0 through :100).

NOTE

1. Hexadecimal data parameters may be of any length, but if more than four characters are typed, only the last four are saved. For example, if the user types:

*I01A2BCD

DEBUG4 uses 2BCD for the Inspect (I) command. This is an aid to the user in case an erroneous hexadecimal digit is typed. In case of an error, the user does not have to restart from the beginning.

2. Unless otherwise noted, all commands must be followed by a carriage return.

3.1 DISPLAY/CHANGE USER REGISTERS (R)

When DEBUG4 gains control, the user program hardware register values are saved in DEBUG4 temporary locations. Before DEBUG4 is subsequently exited via a Jump or Breakpoint command, the values are first restored to the actual registers. These values can be displayed or altered by typing in the R command followed by the appropriate register code name. After a register value is displayed, a new value can be typed in immediately, or a carriage return can be issued to specify no change.

The following code names are used for the registers:

P	P Register	Q	Q Register
A	A Register	Y	Y Register
X	X Register	K	K Register
S	S (Status) Register	L	L Register
D	Console Data Register	C	Console Sense Switch

All values are saved in 16-bit temporary cells, including the C register. Only the setting of bit 0 in the C register's temporary cell controls the actual sense switch. Console sense switch display is 1 for "set" and 0 for "reset." RC1 sets the console sense switch and RC0 resets the console sense switch.

The following example displays the user program's A Register value, does not alter it, displays the user program's Console Data Register value, and changes it to vvvv:

```
*RA
aaaa
*RD
dddd vvvv
*
```

The value of a user register can be changed without displaying the old value. This is done by entering the R command, the user register name, followed by the new value, as follows:

*RQvvvv

The operator may redisplay user register contents divided by 2 by entering a slash (/) and a carriage return. User register contents multiplied by 2 may be redisplayed by entering an X and a carriage return:

*RA	Display contents of user program A register.
2222/	Display contents divided by two.
TTTTX	Display contents multiplied by two.
4444	Terminate register display.
*	Ready for a new command.
-	

NOTE

The (/) and (X) terminators do not alter the original contents of the register.

To display the contents of all the registers, the "R" command is entered without any parameter:

```
*R
P=xxxx A=xxxx Q=xxxx X=xxxx Y=xxxx
K=xxxx L=xxxx S=xxxx D=xxxx C=xxxx
```

The contents of the user program's Status register, which can be displayed and altered with an RS command, are summarized below and illustrated in Figure 3-1 for convenience.

Bit	Hex Value	Set	Reset
08	:100	Interrupts Enabled	Interrupts Disabled
06	:40	64K Addressing	32K Addressing
05	:20	RTC Enabled	RTC Disabled
04	:10	Console Interrupts Enabled	Console Interrupts Disabled
02	:04	Byte Mode	Word Mode
01	:02	OV Set	OV Reset
00	:01	CY Set	CY Reset

Bits 12 and 13 control interrupt levels associated with the I/O subsystem. A 1 indicates the level is disabled. A zero indicates that the level is enabled.

The bits not listed, those that are shown shaded in Figure 3-1, are reserved.

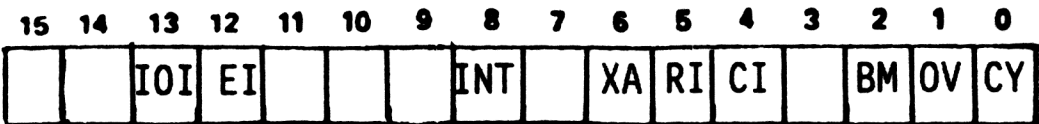


Figure 3-1. Processor Status Register
- 3-3 -

3.2 DISPLAY/CHANGE GENERAL REGISTERS (G)

User programs are normally assembled at relative location :0000, but are loaded at some higher address. To match the locations on an assembly listing with the locations in memory, the user would have to add an offset factor, or Relocation Bias, to each address used in a DEBUG4 command. To avoid this time-consuming and error-prone process, DEBUG4 provides 36 locations called General Registers, R0 - R9, RA - RZ. The value of any of the General Registers is displayed or changed in the same way that User Registers are manipulated. Once the user has established a useful value in a General Register, addresses in commands may be entered as a hex number (from the assembly listing location column) immediately suffixed by a General Register name.

The following exchange displays the value of General Register 6, does not alter it, displays the value of General Register 7, and changes it to vvvv:

```
*G6
8888
*G7
8888 vvvv
*
```

3.3 SET BREAKPOINTS (B)

A breakpoint is a designated location in the user program. If the user program is in control, and the path of execution reaches a breakpoint, control is passed to DEBUG4 for operator action. There are two kinds of breakpoints: temporary and dummy. (See Subsection 3.3.2).

DEBUG4 saves the user program instruction which ordinarily exists at each breakpoint, replaces it with a special word of machine code, and notes the location involved. When the user program control later encounters the breakpoint, DEBUG4 regains control, restores the original contents of all the temporary breakpoints (but not the dummy breakpoints), and informs the operator that a breakpoint has been reached.

The value of the P register is displayed as an absolute location and is the address of the breakpoint location just reached. From this information, the operator can decide which command to enter next.

3.3.1 Temporary Breakpoints

Temporary Breakpoints are established with a Breakpoint command. The syntax of the Breakpoint command allows specification of 1 through 4 breakpoint locations. Program control is always transferred to the current value of the saved P register. Additionally, an iteration count may be specified so that DEBUG4 command mode is not re-established until the original instruction at the breakpoint location has been executed the requested number of times (as if the breakpoint instruction was not there) and program execution reaches the breakpoint one extra time.

The following form of the Breakpoint command replaces the user's instruction at location aaaa with the DEBUG4 breakpoint instruction (JST *0) and transfers control to the location pointed by current temporary "P" register:

*Baaaa[;nnnn]

The "P" register must be loaded using an RP command if the control has to be passed to a location other than the current "P" register value. When the user program execution reaches location aaaa for the (nnnn+1) th time, DEBUG4 is re-entered. The user's original instruction is restored into location aaaa and the value of aaaa stored into the temporary P register.

NOTE

If the user's program never reaches location aaaa (because of a bug in the user's program, for example), re-entering DEBUG4 via the programmer console causes location aaaa to be restored with its original contents.

The following form replaces the instructions at location aaaa,bbbb,cccc and dddd with breakpoint instructions and passes control to the temporary "P" register value:

*Baaaa[;iiii][,bbbb[;jjjj]][,cccc[;kkkk]][,dddd[;llll]]

One to four breakpoint locations may be specified. DEBUG4 regains control when the first specified count (iiii,jjjj,kkkk or llll) condition is satisfied, at which time all four locations are restored with their usual contents.

NOTE

1. The optional iteration count parameter must be entered as a hexadecimal value.
2. The use of the optional iteration count parameter requires two consecutive instructions that are executed for each iteration of the loop. The first instruction must be a single-word instruction. The following example assumes a current P register value of :100:

0100	0001	REL	:100
00000100	0002	EQU	\$
0100 8284 0105	0003	COPY	COUNT,A
0101 0801	0004	ADD	=1,A
0102 8682 0105	0005	COPY	A,COUNT
0103 118C 0100	0006	JNE	A,LOOP
0104 0E0D	0007	HLT	
0105 0000	0008	WORD	0
0100	0009	END	LOOP

The following command does not work:

 * B103;1F

Although instruction :103 is executed for each iteration of the loop, the instruction at location :104 is not.

The following command executes the loop :1F times and returns control to DEBUG4:

 * B102;1F

3. *B101;0 and *B101 are equivalent commands.

3.3.1.1 The Y Command

The Y command sets a temporary breakpoint at the next location (current P register value plus 1). The Y command is written in the following form:

 *Y

For example, if the current value of the P register is :2000, the Y command operates the same as "B2001 (carriage return)," which sets the temporary breakpoint at location :2001. No carriage return is needed for the Y command.

NOTE

The instruction at the current location must be a single-word instruction.

3.3.1.2 The Q Command

The Q command sets a temporary breakpoint at the current location minus 1 (current P register value minus 1). The format of the Q command is the following:

 *Q

For example, if the current value of the P register is :2000, the Q command operates the same as "B1FFF (carriage return)," which sets the temporary breakpoint at location :1FFF. No carriage return is needed for the Q command.

NOTE

The instruction preceding the current location must be a single-word instruction.

3.3.2 Dummy Breakpoints

Dummy breakpoints can be coded into a source program. When control goes through a dummy breakpoint, DEBUG4 receives control and types out the same register display as for a temporary breakpoint established with a B command. DEBUG4 does not keep a record of dummy breakpoint locations, and they are never cleared or replaced -- they are a permanent part of the user program, just like any other machine code. When a B or J command with no transfer address is used to exit DEBUG4 after a dummy breakpoint, the user program's P Register value is automatically incremented past the dummy breakpoint. The same commands can be used with a real breakpoint.

To establish a dummy breakpoint in a source program, use the following code:

```
JST *0
```

Alternatively, DEBUG4 may be used to create a dummy breakpoint; simply alter the desired location to BD00--the machine language equivalent of the JST *0 instruction--using the I command.

NOTE

Absolute location 0 is used by DEBUG4 to maintain the return address to DEBUG4.

3.4 JUMP TO USER PROGRAM (J)

The Jump command (J) is used to switch control from DEBUG4 to the user program. After the J command is input, DEBUG4 reloads all hardware registers except P, with the values of their corresponding temporary cells, and performs an unconditional jump to the specified location:

J2200 Restores all user registers into real hardware registers and exits DEBUG4 by jumping to absolute location :2200.

J44G6 Exits to (General Register 6) + :44.

When no parameter is specified in the J command, the next inline instruction in the user program (the value of user P register) is used as the jump address:

J Restores all user registers and exits to P.

Temporary breakpoints can be established with the Jump command. The syntax of the Jump command allows specification of 1 through 4 breakpoint locations. Additionally, an iteration count may be specified so that DEBUG4 command mode is not re-established until the original instruction at the breakpoint location has been executed the requested number of times (as if the breakpoint instruction were not there) and program execution reaches the breakpoint one extra time.

The following form replaces the user's instruction at location aaaa with a DEBUG4 breakpoint instruction (JST *0) and transfers control to location zzzz:

```
*Jzzzz,aaaa[;cccc]
```

When the user program execution reaches location aaaa for the (cccc+1) th time, DEBUG4 is re-entered, the original user's instruction is restored into location aaaa, and the value aaaa is stored into the temporary P register.

NOTE

If the user's program never reaches location aaaa (because of a bug in the user's program, for example), reenter DEBUG4 via the programmer console, and location aaaa is restored at this time.

The following form replaces the instructions at locations aaaa, bbbb, cccc, and dddd with breakpoint instructions and transfers control to location zzzz.

```
*Jzzzz,aaaa[;iiii][,bbbb[;jjjj]][,cccc[;kkkk]][,dddd[;llll]]
```

One to four breakpoint locations may be specified.

DEBUG4 regains control when the first specified count (iiii, jjjj, kkkk, or llll) condition is satisfied, at which time all four locations are restored with their original contents. For example:

```
*J100,102;1F,108
```

Creates breakpoint instructions at :108 and :102 and transfers control to location :100. DEBUG4 regains control when location :108 is executed, or when location :102 has been reached :1F+1 times.

3.5 INSPECT/MODIFY MEMORY LOCATIONS (I)

The Inspect command (I) displays the contents of a specific word location in memory, and (if desired) specifies a new value for that location. Consecutive locations may be stepped through, either backward or forward. The user enters a starting address for the inspection, and DEBUG4 responds by printing that address and its current contents:

```
*I3000
3000 0000
```

Then, DEBUG4 waits for the user to enter a special line terminator followed by carriage return. A new value for the contents of the word just displayed can precede the plus (+) or minus (-) terminator. The line terminators are:

plus (+)	Continue inspection at location + 1
minus (-)	Continue inspection at location - 1

asterisk (*)	Continue inspection at address specified by location (indirect).
X character	Multiply contents by 2 and continue inspection at the new location.
slash(/)	Divide contents by 2 and continue inspection at the new location.

The following is an example of the Inspect command:

<pre> *I3000 3000 0000+ 3001 1111 CCCC - 3000 0000- 2FFF 0100* 0100 3000X 6000 4000/ 2000 4455 * - </pre>	<pre> Inspect location :3000 Do not alter contents of :3000. Inspect :3000 + 1. Alter contents of :3001 to :CCCC. Inspect :3001 - 1. No alteration. Inspect :3000 - 1. Inspect location :100. Multiply the contents of :0100 by 2. Look in that location. Divide :4000 by 2. Look in that location. No alteration. Terminate inspection sequence. Ready for a new command. </pre>
---	---

Consecutive memory locations may be altered by providing new values separated by commas:

<pre> *I2000 2000 1111 4444,A1A1,1000,A5A5 </pre>	<pre> Inspect location :2000 Change location :2000, :2001, :2002, and :2003 to :4444, :A1A1, :1000 and :A5A5 respectively. </pre>
---	---

If the location to be inspected is followed by * -- for example 3AAA* -- the location will be treated as an indirect address. Whenever indirect addressing is specified by *, the effective address will be calculated as follows:

<pre> User S Reg XA bit = 1 </pre>	<pre> The specified location contains the address to be inspected. (Single level indirect.) </pre>
<pre> = 0 </pre>	<pre> The specified location contains either the direct address to be inspected (bit 15 off) or an indirect pointer (bit 15 on). DEBUG4 will resolve as many levels of indirection as neces- sary to display the ultimate value. (Note that it is possible to hang up in an endless indirect loop at this time.) </pre>

If a displayed word is a Byte Address, the user can request that DEBUG4 compute the word value of the location. A slash (/) response to a query produces a display of the word value. DEBUG4 then waits for further information as with any other Inspect command:

```

*I2FFE
2FFE 4444/
2222 0000
*
-

```

Each time the I command is used to modify a memory location, DEBUG4 automatically checks whether the specified location received the specified data. If the specified data is not received, the mismatch address is printed with a percent (%) character prefix.

3.6 LIST CONSECUTIVE MEMORY LOCATIONS (L)

The List command (L) prints the contents of consecutive memory locations on the TTY/CRT/Printer. The L command requires two parameters; the starting and ending locations (inclusive) are displayed:

_*Lssss,eeee

The contents of memory locations ssss through eeee inclusive are displayed on the TTY. The display is formatted eight words per line, preceded by the address of the first word on each line.

_*Laaaa,aaaa+5

The contents of six consecutive memory locations, beginning with location aaaa, are displayed.

If the second parameter is less than first parameter (starting address), it is interpreted as an absolute hexadecimal number, specifying the number of consecutive memory locations to be listed. For example:

_*L100,A List 10 memory locations starting at :100.

If less than the first parameter, the second parameter must not be greater than :28 (40 decimal).

An additional parameter "A" available on the L command causes the equivalent ASCII representation of listed memory cells to be displayed. The example below illustrates the use of this command parameter:

_*L100,107,A

The eight hexadecimal values corresponding to the contents of locations :100 through :107 are displayed, followed by 16 equivalent ASCII characters. If a listed cell does not contain a valid ASCII value in either one of its bytes, a period (.) is output as the ASCII representation for that cell.

NOTE

The ASCII option is not implemented in MDEBUG4.

3.7 FILL MEMORY WITH A CONSTANT VALUE (F)

The Fill command (F) stores a specified 16-bit value into each word location of a selected area of memory. The F command requires three parameters: the starting and ending locations (inclusive) to fill, and the value to store into these locations; the ending address must not be less than the starting address.

_Fssss,eeee,vvvv

The value vvvv is stored into each word of memory locations ssss through eeee inclusive.

An optional fourth parameter may be specified which allows an incrementing or decrementing pattern to be stored, as follows:

_Fssss,eeee,vvvv,nnnn

Locations ssss through eeee are filled with an incrementing pattern:

LOCATION	CONTENTS
ssss	vvvv
ssss+1	vvvv+nnnn
ssss+2	vvvv+nnnn+nnnn
:	:

If nnnn is a negative value, the pattern decrements rather than increments.

3.8 COPY ONE AREA OF MEMORY TO ANOTHER (C)

The Copy command (C) copies an area of given length to another area of the same length.

_Cssss,eeee,nnnn

The contents of memory from ssss to eeee inclusive are copied to another area starting at nnnn.

After performing a copy operation, DEBUG4 automatically checks whether all locations in the copy area received the proper data. If a mismatch is encountered, the mismatch address is printed with a percent (%) character prefix. Overlapping areas may produce mismatches.

3.9 SEARCH MEMORY (S)

The S command searches memory, from a low address to a high address for a match/non-match of a desired bit value. The general form is:

S1111,hhhh,vvvv

where 1111 is the low memory address, hhhh is the high memory address and vvvv is the search value.

An example of the S command follows:

*S0FC3,1FC3,124B

0FD1 124B

1F02 124B

*
—

Searches from a low address of :0FC3 to a high address of :1FC3 and prints out each location address with a 16-bit match of the value :124B. A match was found at locations :0FD1 and :1F02.

There are two optional parameters: mask and minus (-) sign before carriage return, for non-match.

By specifying a bit-mask, the user tells DEBUG4 to compare only those bits in each word of memory which have a corresponding "1" bit in the mask. For each "0" bit in the mask, the corresponding bit in memory is ignored in finding a mask against the value. For example:

*S3700,3AFF,1234,7FFF

Searches from :3700 to :3AFF for the value :1234 and accepts a match on the value :9234.

By specifying a minus (-) sign, the search is made for a non-match of the match value. For example:

*S01A0,1000,1234,-

Searches memory from :01A0 to :1000 for location having value other than :1234.

3.10 PRINT CHAIN STRUCTURE (Z)

NOT SUPPORTED BY MDBUG4

A specialized data structure called a "chain" or a "linked list" is either created using the CHAN assembler directive or built at run time. Chains are used in RTX4 and in the software that supports NAKED MINI@ 4 FORTRAN IV.

The Z command of DEBUG4 is specifically intended to trace and print chain information. This command uses three parameters: the starting address (head) of the chain, the number of words in each node of the chain, and the relative position of the link word within each node. DEBUG4 saves the second and third parameters from one Z command to the next, so these parameters are optional after the first execution of the Z command. The initial default values for the second and third parameters are :A and :0, respectively.

The Z command outputs the contents of each word in every node in the chain, formatted as one output line per node. The final node in the chain, called the "tail", has the value :0000 in its link word.

EXAMPLE 1

*Z2345,4,0

3333 0123 4567 89AB

3999 CDEF 0123 4567

0000 89AB CDEF 0123

*
—

Trace and print a chain from its head at :2345. Each node contains four words, with the link being the first (0th) word.

First node, at :2345.

Second node, at :3333.

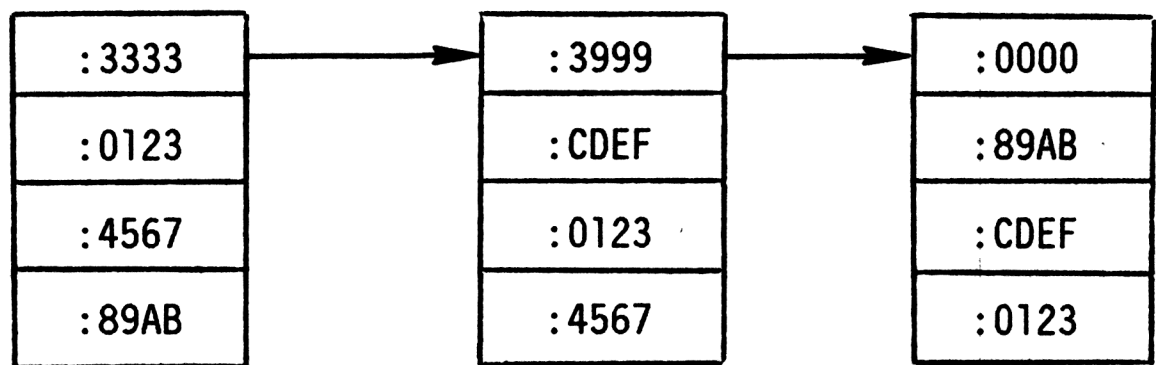
Third and last node, at :3999.

Ready for a new command.

Memory Location :2345

Location :3333

Location :3999



The link word always points to the next link word.

EXAMPLE 2

*Z2345,4,2

3333 0123 4565 89AB

3999 CDEF 0121 4567

9ABC DEFO 0000 1234

*

—

First node at :2345, link word points to location :4565.

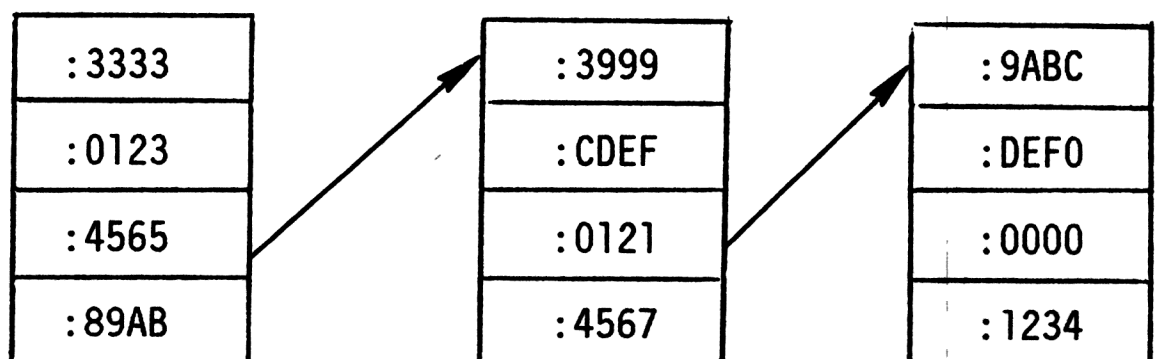
Second node, at :4565.

Third node, at :121.

Memory Location :2345

Location(:4565)

Location (:0121)



3.11 ASSIGN DEVICES (A)

NOT SUPPORTED BY MDBUG4

DEBUG4 supports two kinds of output device assignments: punch (AP) and list (AL).

For binary loader/dump (W,D), command device assignment is made as follows:

*APd where d = H (High Speed Paper Tape Reader/Punch/
T (TTY Reader/Punch)

Most DEBUG4 commands result in a display of memory or register contents. The AL command may be used to direct output to either the TTY (CRT), the printer, or both. To select the appropriate device for listing output enter the following:

*ALd where d = T (TTY or CRT),
P (line printer) or
B (both TTY/CRT and line printer)

The line printer must be a Distributed I/O device address :F and channel number 7.

Note that ALP or ALB outputs to the line printer only for the list (L), Search (S), and Chain (Z) commands. Display for all other functions is to the TTY/CRT only, regardless of the ALD specification.

3.12 PUNCH OR VERIFY MEMORY

NOT SUPPORTED BY MDBUG4

The user may punch or verify specified areas of memory using standard Computer Automation, Inc., Binary Tape format. The tape is punched by using the following command:

$DF_1, T_1 [, F_2, T_2, \dots, F_6, T_6] [, TAD]$

Parameters are processed in pairs. The first word of a pair specifies the first word of a block to be punched. The second word of a pair specifies the last word of the block to be punched. If an odd number of parameters is entered the last parameter is the transfer address (execution address). Up to six memory segments can be punched using this command. For example:

$*D100, 150, 750, 800, 1000, 107F, 102F$ Dumps memory block :100 through :150, :750 through :800, :1000 through :107F. Location :102F is the transfer address.

NOTE

Only ABS (absolute) data is generated by this command. In addition, the desired device assignment must have been made previously using the Assign command (A).

The tape is verified by using the following command:

$VF_1, T_1 [, F_2, T_2, \dots, F_6, T_6] [, TAD]$

This operation is similar to punch except that memory blocks are compared for verification with the tape. Except for the "V" character, the typed input should correspond exactly with the input used to punch the tape. During verification, a miscompared address is displayed, prefixed by a percent sign (%).

Punching and reading tape for verification begins immediately upon entry of these commands; be sure that the device is ready before typing the carriage return.

3.13 LOAD FROM PAPER TAPE (W)

NOT SUPPORTED BY MDBUG4

DEBUG4 may be used to read a paper tape in the standard Computer Automation, Inc., Binary Tape format from either the High Speed Paper Tape (HSPT) reader or the TTY reader. The tape is loaded using the following command:

W [offset] where "offset" is the optional relocation bias applied to relocatable binary tapes.

DEBUG4 only accepts the following specific loader type codes:

:18	Begin
:02/:03	End Program ABS/REL
:04/:05	Origin ABS/REL
:06/:07	Data ABS/REL
:08	Reserve and fill
:09	Byte address constant
:0D	Entry here (ignored)
:0E	External address (stored in memory as :FFFF)
:0F	Primary entry declaration (ignored)
:40	NULL
:4F	List segment names (list location counters)
:6D	NOTE RECORD
:6F	List segment attributes

The user is notified of the following errors:

ERROR TYPE	DESCRIPTION
C	Checksum error
Mxxxx	Memory overflow error at memory address xxxx
Tcccc	cccc is an illegal type code

At the end of the load, the starting load address, end address, and execution address if any, are displayed.

NOTE

The desired input device (HSPT or TTY) must be assigned, using the assign command A, prior to using this command. (The tape must be ready.)

3.14 BUS MONITOR TRANSACTIONS (MM, MC, ML)

SUPPORTED BY XDEBUG4 ONLY

The Bus Monitor hardware (part number 53718-01), if installed, provides the capability of tracing all transactions on the computer bus and of interrupting or resetting the processor when user-defined conditions occur.

To use the Bus Monitor with XDEBUG4, the device address must be set to :16, the interrupt vector address must be set to :96, and the skip count must be set to 5. The user is referred to the Bus Monitor User's Manual, 91-53718-00, for information on setting the addresses. In addition, Switch SW1 on the Bus Monitor card must be set for a Step count of 5.

Initially, the Bus Monitor is in the off state. To enable the Bus Monitor, the programmer issues either an MM or an MC command. When XDEBUG4 passes control to the user program, the Bus Monitor monitors the bus until control returns to XDEBUG4. At this point, the Bus Monitor returns to its default (off) state. The programmer must issue another MM or MC command to reenable it.

3.14.1 Bus Monitor Mode (MM)

The MM command defines the type of memory accesses to be compared with the conditions defined by the MC command. In addition, the MM command indicates the type of action to be taken if a match occurs. The command syntax is:

MM[access[,access[,access]],action]

where:

access specifies the type of access to be compared:

R - non-DMA read transactions

W - non-DMA write transactions

D - all DMA transactions

action specifies the action to be taken if a match occurs:

I - interrupt the processor and pass control to XDEBUG4

E - reset the processor

The programmer can specify one, two, or three access codes in the command, but he can specify only one action code. These specifications are effective whenever the Bus Monitor is enabled until they are changed by a subsequent MM command.

For example:

*MMR,W,E

This command enables the Bus Monitor and specifies that the processor is to be reset if the conditions defined by the MC command occur during a read or write non-DMA operation.

An MM command with no arguments clears any previous MM specifications, allowing the programmer to monitor bus transactions without interrupting or resetting the processor.

3.14.2 Trace Conditions (MC)

In the MC command, the programmer specifies a set of bus conditions; if these conditions occur, XDEBUG4 takes the action defined by the MM command. The operation causing the matching condition may not complete.

The command syntax is:

MC[expression[,expression[,expression]]]

where "expression" may be:

=location	access of the specified memory location
>location	access of a memory location equal to or greater than the specified location
<location	access of a memory location equal to or less than the specified location
location ₁ :location ₂	access of a memory location greater than location ₁ and less than location ₂

The valid combinations of expressions are:

```

=location
>location
<location
location1:location2

=location,=location
>location,=location
<location,=location
<location,>location
location1:location2,=location

=location,=location,=location
<location,>location,=location

```

The MC command is also used to display the current setting of the various parameters controlling Bus Monitor activities. If the programmer issues an MC command with no arguments, XDEBUG4 displays the contents of the Bus Monitor registers in the following format:

C1=value C2=value C3=value M=value

where:

C1,C2,C3 represent the comparison values specified in the most recent MC command

M represents the type of transactions being monitored and the action to be taken if a match is found:

<u>Bit</u>	<u>Meaning if On (=1)</u>
0	Monitor Non-DMA Write Transactions
1	Monitor Non-DMA Read Transactions
2	Not Used
3	Monitor all DMA Transactions
4	Compare Address Bus for Equal to C1
5	Compare Address Bus for Equal to C2
6	Compare Address Bus for Equal to C3
7	Compare Address for Greater than or Equal to C1 and Less than or Equal to C2
8	Compare Address for Less than C1 or Greater than C2
9	Reset processor when match is found
10	Interrupt processor when match is found
11-15	Not Used

For example, suppose that the programmer has issued the MM command illustrated on the preceding page and the MC command:

*MC>5000,<2000,=3000

If he then issues an MC command with no arguments, the following sequence occurs:

*MC

C1=5000 C2=2000 C3=3000 M=01C3

*

The M value 01C3 indicates that non-DMA read and write transactions are to be monitored for values greater than or equal to :5000 and less than or equal to :2000 or equal to :3000, and that the processor is to be reset if a match is found.

3.14.3 Bus Monitor Trace Display (ML)

NOTE

To use the ML command to list complete information concerning an instruction fetch, it is necessary to connect the IF- line on processor pin 110 to one of the Bus Monitor external inputs.

The Bus Monitor stores the most recent bus transactions, up to :50 entries. The ML command causes some or all of these transactions to be listed.

The programmer can issue the command in any of the following forms:

- ML requests that all currently stored transactions be displayed.
- ML* requests that all transactions since the most recent J or B instruction be displayed.
- MLn requests that the "n" most recent transactions be displayed, where "n" is between 1 (newest) and :50 (oldest).
- MLn,m requests that transactions "n" through "m" be displayed, where both "n" and "m" are between 1 and :50 and "m" cannot be less than "n".

Output from the ML command is in the form:

type address status mode address status mode ...

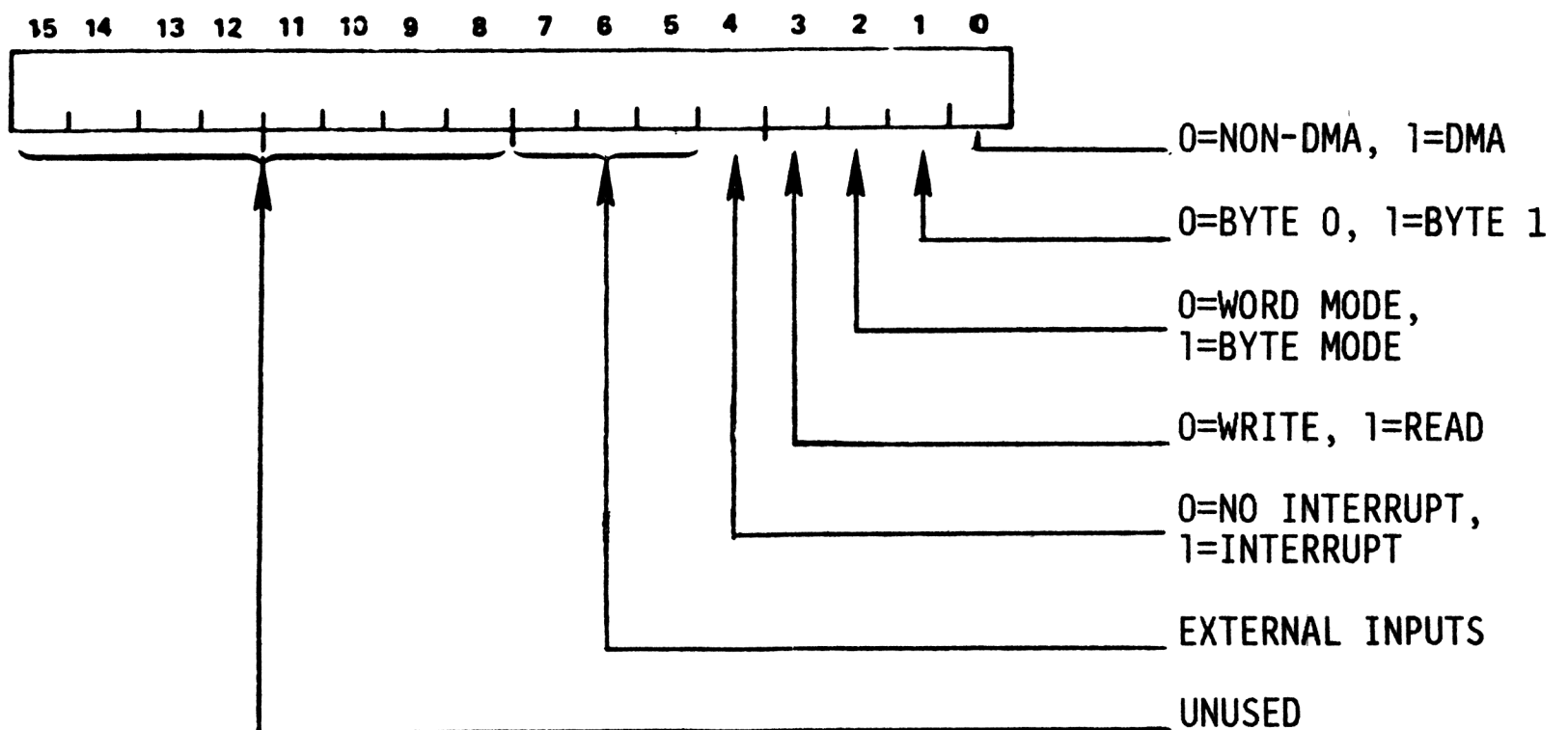
where:

type is the type of access:

- F - instruction fetch
- I - interrupt request
- D - DMA access

address is the location addressed by the memory bus lines

status is the hexadecimal equivalent of the following Bus Monitor status bits:



mode is the mode of the transaction:

R - read transaction
W - write transaction

Because entries relating to location :0 and to locations within XDEBUG are not displayed, the command may display less entries than were requested.

For example, suppose that the programmer has stored the following code at location :5000:

```
COPY    A,X
COPY    A,LOCTIN
COPY    QSEC,Q
EXCH    A,Q
```

where LOCTIN is stored at :5010 and QSEC at :5072. He issues the command:

```
*J5000,5003
```

When control returns to XDEBUG, he issues an ML command to display the transactions which have occurred since the J command:

```
*ML*
F5000 88R
F5001 88R 5010 00W
F5002 88R 5072 08R
F5003 08R
*
```

3.15 INTERCEPT TRAPS (T)

SUPPORTED BY XDEBUG ONLY

The T command allows the programmer to specify that XDEBUG is to intercept traps if they occur during the execution of his program. The command syntax is:

```
T[trap[,trap]...]
```

where:

trap identifies a type of trap to be intercepted:

C - console interrupt trap
I - unimplemented instruction trap
U - uninstalled memory trap
T - all traps
A - arithmetic exception trap
S - stack overflow trap
N - character/numeric error trap

A T command with no arguments causes XDEBUG to discontinue intercepting any traps.

When the programmer issues a T command, XDEBUG4 saves the current values in the trap vectors associated with each trap to be intercepted and sets those vectors for service by XDEBUG4. If a later T command discontinues XDEBUG4 interception of particular traps, the associated vectors are restored.

For example:

	*TC,I,U	Saves trap vector contents associated with the console interrupt, unimplemented instruction, and uninstalled memory traps and stores XDEBUG4 information in these vectors.
	.	
	.	
	*TI,U	Restores the saved trap vector contents associated with the console interrupt trap, and maintains XDEBUG4 interception of unimplemented instruction and uninstalled memory traps.
	.	
	.	
	*T	Restores the saved contents into the unimplemented instruction and uninstalled memory trap vectors.
	-	

When the programmer issues a T command to request XDEBUG4 interception of console interrupts, XDEBUG4 immediately sets (to 1) the Console Interrupt enable (CI) bit (bit 4) of the user program's S register. However, XDEBUG4 does not attempt to preserve or restore the CI bit if it is subsequently reset (to 0) by either the program (via a CBIT or COPY instruction) or the operator (via an RS command). Further, XDEBUG4 does not interfere with the EI bit of the program's S register except in response to an explicit RS command. The T command alone, then, does not guarantee that pressing the INT button on the console will always interrupt a user's program.

When a subsequent T command removes XDEBUG4 interception of console interrupts, XDEBUG4 immediately resets the CI bit in addition to restoring the associated trap vector.

APPENDIX

DEBUG4 PROGRAM AND COMMAND SUMMARY

A.1 PROGRAM SUMMARY

1. DEBUG4 and MDBUG4 are binary relocatable. XDEBUG4 is not.
2. User must turn Status Register XA bit on if loading DEBUG4 above 32K.
3. NAMs are DEBUG4, XDEBUG4, and MDBUG4.
4. Control can be transferred to DEBUG4 at origin address for standard CRT Distributed I/O address and channel number, or at origin + 1 for a desired Distributed I/O address and channel number in the Q Register. Q Register must contain the Distributed I/O address in bits 4-7, the channel number in bits 1-3 and bit 0 = 0.
5. Sign on messages are:

 DEBUG4 XX :Origin address - End address, or
 XDEBUG4 XX :Origin address - End address, or
 MDBUG4 XX :Origin address - End address
 where "XX" is the version number of the DEBUG4 program.
6. All parameters must be separated by a comma.
7. In commands which can specify more than one memory location, the default value of the first address is zero if a comma is entered immediately after the command.
8. Embedded blanks are not allowed in a command.
9. Back-arrow deletes the command and "*" is displayed on the next line. User can then reenter the command.
10. Invalid commands or parameters are rejected by a back-arrow.
11. A command must be terminated by a carriage return, unless otherwise noted.
12. Address and data parameters can be specified in an expression using any combination of G-Reg, R-Reg, + or -.
13. Altering the contents of memory inserts an asterisk (*) at the beginning of the command line.

14. General registers (G0 - G9 and GA - GZ) may be referenced by a user program linked to DEBUG4 by referring (in a NAM directive) to the register name with an imbedded colon; e.g.;

NAM G:8

causes the G8 register in DEBUG4 to be set to the location G:8 in the user program.

15. Breakpoint uses location 0 to hold return address, and uses JST *0 as breakpoint instruction.
16. The B (Breakpoint) and J (Jump) commands may optionally use iteration counts.
17. When DEBUG4 modifies a memory location, the altered location is verified for proper modification. If a miscomparison occurs, the miscomparison address is printed with a percent (%) character prefix.
18. DBG4:LP and DBG4:CNT are NAMED in XDEBUG4 and DEBUG4 (see respective load maps).

If the line printer being used is an automatic line feed printer, make the following patch:

	<u>IS</u>	<u>CHANGE TO</u>
ORIGIN ADDRESS + DBG4:LP	:BE81	:0000

To alter the number of lines printed per page on the line printer, change location ORIGIN ADDRESS + DBG4:CNT to the 2's complement representation of the desired number of lines per page. The default is 39 lines per page.

For example, if you wish to print 60 lines per page to the line printer, make the following patch:

	<u>IS</u>	<u>CHANGE TO</u>
ORIGIN ADDRESS + DBG4:CNT	:FFD9	:FFC4

A.2 COMMAND SUMMARY

ALd ¹	Assign output device. d = T (TTY/CRT), P (printer) or B (both).
APd ¹	Assign output device. d = H (HSPT reader), T (TTY reader)
J	Reenter user program according to P Register contents.
Jt	Reenter user program at location t.
Jt,a,b,c,d	Set temporary breakpoints (up to 4) at a, b, c and d. Reenter user program at location t. Each breakpoint may optionally use an iteration count.

¹These commands are not supported by MDBUG4.

Ba,b,c,d	Set temporary breakpoints (up to 4) at a, b, c, and d. Reenter user program according to P Register contents. Each breakpoint may optionally use an iteration count.
Q	Set temporary breakpoint at current P register value minus 1. No carriage return is necessary.
Y	Set temporary breakpoint at current P register value plus 1. No carriage return is necessary.
R	Display all user program registers.
Rr	Display user program r Register, where r = A, Q, X, Y, P, K, L, S, D (Console Data Register) or C (Console Sense Register).
Rrvvvv	Change user program register r to vvvv.
Gg	Display general register, where g = 0-9, A-Z.
Ggvvvv	Change general register g to vvvv.
Ia	Inspect memory at location a. After value is displayed, commands are: + Inspect next location / Continue inspection at value/2 - Inspect previous location X Continue inspection at value x 2 * Inspect indirect Consecutive memory locations can be altered by providing new values separated by commas.
Cs,e,nnnn	Copy memory from s to e inclusive, into new area starting at nnnn.
Ls,e	List memory from s to e inclusive. (s≤e)
Ls,e,A ¹	List memory from s to e inclusive with ASCII equivalents (s≤e)
Ls,n	List memory from s for :n locations. (s>n & n≤:28)
Ls,n,A ¹	List memory from s for :n locations with ASCII equivalents (s>n and n≤:28)
Fs,e,vvvv	Fill memory from s to e inclusive (s≤e) with constant value vvvv.
Fs,e,vvvv,d	Fill memory from s to e inclusive (s≤e), using d as an increment/decrement to build a pattern. Store vvvv into s, vvvv+d into s+1, vvvv+d+d into s+2, etc.
Ss,e,vvvv	Search memory from s to e inclusive (s≤e) for value vvvv.
Ss,e,vvvv,m	Search memory from s to e inclusive (s≤e) for value vvvv under mask m, where zero bits of m are don't-care.
Ss,e,vvvv,-	Search memory from s to e inclusive (s≤e) for non-match of value vvvv.
Zs,w,d ¹	Print chain. - s = Starting address - w = Number of words/node - d = Relative displacement of link word

¹These commands are not supported by MDBUG4.

DF ₁ ,T ₁ [,F ₂ ,T ₂ ,...F ₆ ,T ₆][,TAD] ¹	Dump specified memory areas F ₁ = start T ₁ = end pair TAD = execution (transfer) address
VF ₁ ,T ₁ [,F ₂ ,T ₂ ,...,F ₆ ,T ₆][,TAD] ¹	Verify specified memory areas with tape
W offset ¹	Load binary tape, offset - relocation bias
MMm,m,m,a ²	Set the Bus Monitor mode switch for: m { R - monitor non-DMA reads W - monitor non-DMA writes D - monitor all DMA accesses a { I - interrupt on match E - reset on match 0 to 4 parameters may be used.
MM ²	Clear any previous MM specifications
MCexp ₁ ,exp ₂ ,exp ₃ ²	Set or display the Bus Monitor match registers. The following combinations are allowed: =x <x <x,=y <x,>y =x,=y >x >x,=y <x,>y,=z =x,=y,=z x:y x:y,=z
MC ²	Display current Bus Monitor register settings in the form: _C1=value C2=value C3=value M=value
ML ²	Display all Bus Monitor entries.
ML*2	Display current Bus Monitor entries.
MLx ²	Display x Bus Monitor entries.
MLx,y ²	Display Bus Monitor entries x through y.
T[trap[,trap]...] ²	Intercept traps: C - console interrupt I - unimplemented instruction U - uninstalled memory T - all traps A - arithmetic exception S - stack overflow N - character/numeric error

¹These commands are not supported by MDBUG4.

²These commands are not supported by MDBUG4 or DEBUG4.