Motorola MVME 143S-2

CPU Controller

In Stock

Used and in Excellent Condition

Open Web Page

https://www.artisantg.com/97359-15

ARTISAN® TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

- Critical and expedited services
- In stock / Ready-to-ship
- We buy your excess, underutilized, and idle equipment
- Full-service, independent repair center

MVME143BUG

143Bug DEBUGGING PACKAGE

USER'S MANUAL

(MVME143BUG/D1)

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights or the rights of the others.

PREFACE

This manual provides general information and operating instructions for the 143Bug firmware provided on the MVME143 MPU Module.

This manual is intended for anyone who wants to design OEM systems, supply additional capability to an existing compatible system, or in a lab environment for experimental purposes.

The MVME143Bug package is a powerful evaluation and debugging tool for systems built around the MVME143 MPU VMEmodule. Facilities are available for loading and executing user programs under complete operator control for system evaluation. The 143Bug includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassembler useful for patching programs, and a self test on power-up feature which verifies the integrity of the system. Various 143Bug routines that handle I/O, data conversion, and string functions are available to user programs through the TRAP #15 handler. In addition, 143Bug provides as an option a "system" mode that allows autoboot on power up or reset, and a menu interface to several system commands used in VME Delta Series systems.

A basic knowledge of computers and digital logic is assumed.

To use this manual, you should be familiar with the publications listed in the *related documentation* paragraph in Chapter 1 of this manual.

Throughout this manual the paragraph headings conform to the following convention:

Entering Debugger Command Lines

(this is a main topic heading)

Syntactic Variables

(this is a subordinate topic heading under a main topic)

Address as a Parameter

(this is a subordinate topic heading under the subordinate topic)

Address Formats

(this is a subordinate topic heading under the subordinate topic)

First Edition April 1989

TABLE OF CONTENTS

CHAPTER 1 - GENERAL INFORMATION

CHAPTER 2 - USING THE 143Bug DEBUGGER

CHAPTER 3 - THE 143Bug DEBUGGER COMMAND SET

CHAPTER 5 - SYSTEM CALLS

CHAPTER 6 – 143Bug DIAGNOSTIC FIRMWARE GUIDE

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1
GENERAL INFORMATION

Description of MVME143Bug

The MVME143Bug package is a powerful evaluation and debugging tool for systems built around the MVME143 monoboard microcomputer. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 143Bug includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassembler useful for patching programs, and a self test on power-up feature which verifies the integrity of the system. Various 143Bug routines that handle I/O, data conversion, and string functions are available to user programs through the TRAP #15 handler. In addition, 143Bug provides as an option a "system" mode that allows autoboot on power up or reset, and a menu interface to several system commands used in Delta Series systems.

143Bug consists of three parts: a command-driven user-interactive software debugger, described in Chapter 2 and hereafter referred to as the debugger, a command-driven diagnostic package for the MVME143 hardware, described in Chapter 6 and hereafter referred to as the diagnostics, and a user interface which accepts commands from the system console terminal.

When using 143Bug, the user operates out of either the debugger directory or the diagnostic directory. If the user is in the debugger directory, then the debugger prompt "143-Bug>" is displayed and the user has all of the debugger commands at his disposal. If in the diagnostic directory, then the diagnostic prompt "143-Diag>" is displayed and the user has all of the diagnostic commands at his disposal as well as all of the debugger commands. The user may switch between directories by using the Switch Directories (SD) command or may examine the commands in the particular directory that the user is currently in by using the Help (HE) command (refer to Chapter 3).

Because 143Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. The flow of control in normal 143Bug operation is illustrated in Figure 1-1. The flow of control in system 143Bug operation is illustrated in Figure 1-2. When a command is entered, 143Bug executes the command and the prompt reappears. However, if a command is entered which causes execution of user target code (for example, "GO"), then control may or may not return to 143Bug, depending on the outcome of the user program.

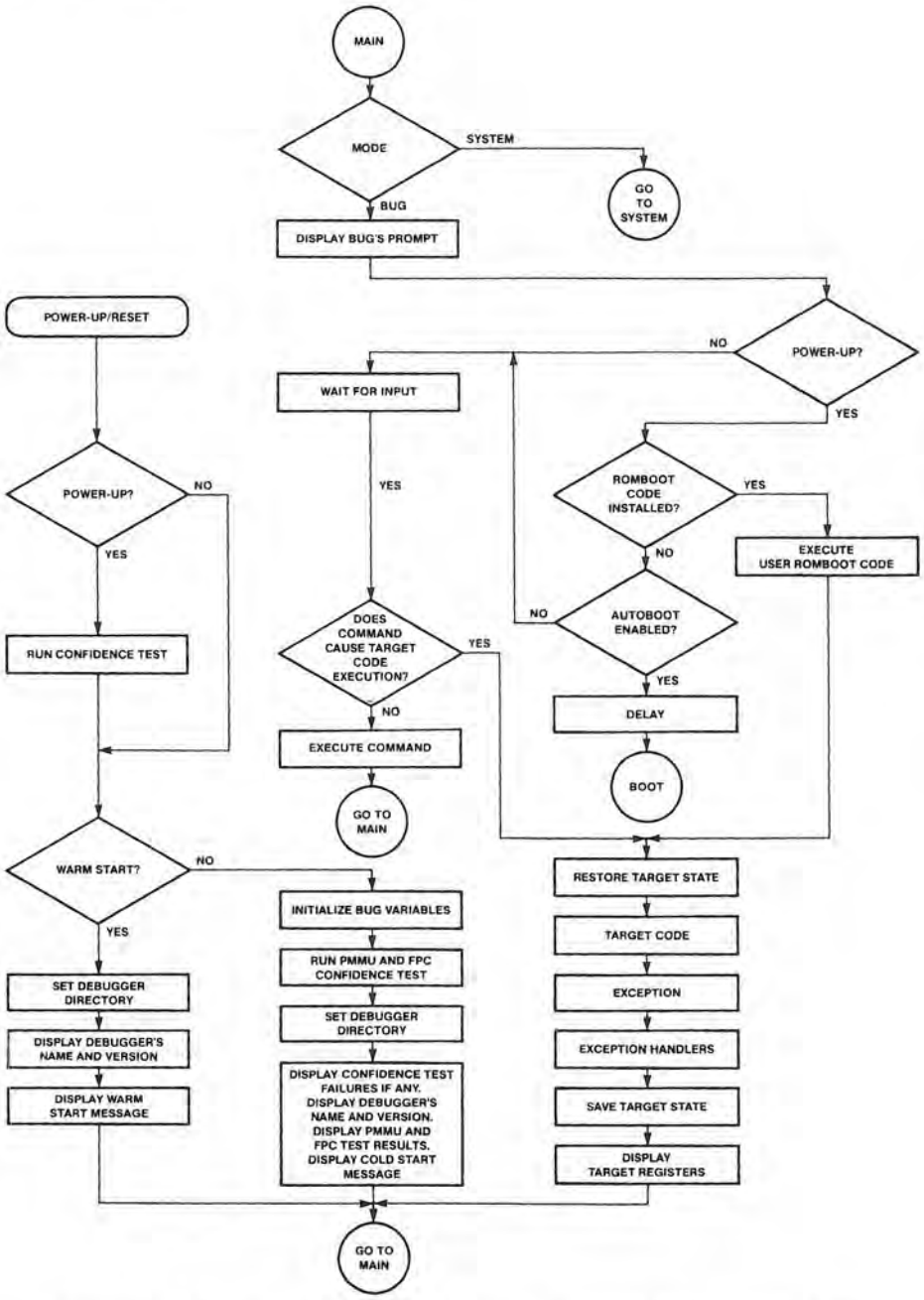


FIGURE 1-1. Flow Diagram of 143Bug (Normal) Operational Mode

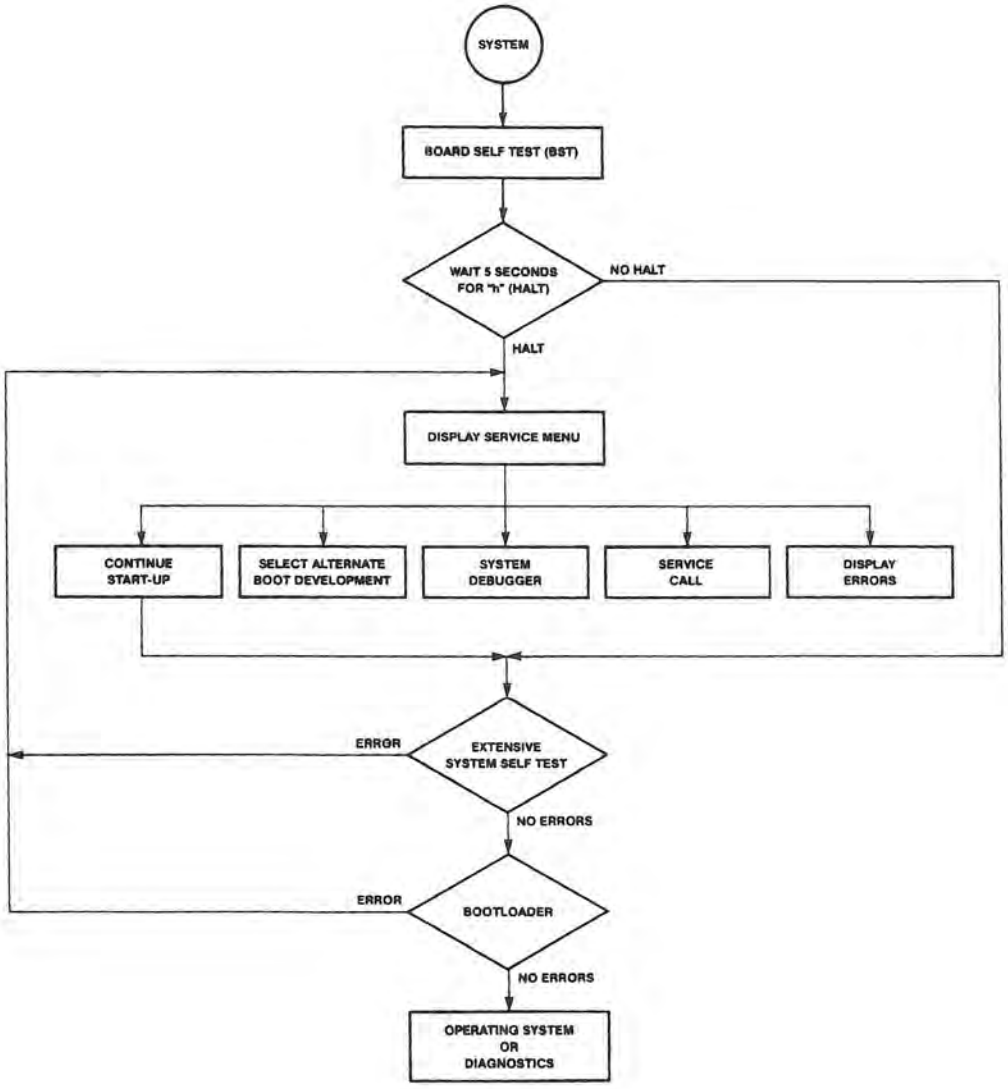


FIGURE 1-2. Flow Diagram of 143Bug (System) Operational Mode

The commands are more flexible and powerful than previous debuggers. Also, the debugger in general is more "user-friendly", with more detailed error messages (refer to Appendix B) and an expanded online help facility.

How To Use This Manual

Users who have never used a debugging package before should read all of Chapter 1 before attempting to use 143Bug. This gives an idea of 143Bug structure and capabilities.

The *Installation and Startup* paragraph in this chapter describes a step-by-step procedure to power up the module and obtain the 143Bug prompt on the terminal screen.

For a question about syntax or operation of a particular 143Bug command, the user may turn to the entry for that particular command in the chapter describing the command set (refer to Chapter 3).

Some debugger commands take advantage of the built-in one-line assembler/ disassembler. The command descriptions in Chapter 3 assume that the user already understands how the assembler/disassembler works. Refer to *Using the One-Line Assembler/Disassembler* in Chapter 4 for details on its use.

NOTE

In the examples shown, all user input is in **BOLD**. This is done for clarity in understanding the examples (to distinguish between characters input by the user and characters output by 143Bug). The symbol <CR> represents the carriage return key on the terminal keyboard. Whenever this symbol appears, it means a carriage return entered by the user.

Installation and Startup

Even though the MVME143Bug EPROMs are installed on the MVME143 module, for 143Bug to operate properly with the MVME143, follow this set-up procedure.

CAUTION

Inserting or removing modules while power is applied could damage module components.

1. Turn all equipment power OFF. Refer to the *MVME143 MPU VMEmodule User's Manual* and configure the header jumpers on the module as required for the user's particular application. The only jumper configurations specifically dictated by 143Bug are those on J5. Header J5 must be configured with jumpers positioned between pins 1–3 and 4–6. This sets EPROM sockets XU3, XU12, XU21, and XU28 for 64K x 8 devices. This is the factory configuration of the MVME143 as shipped.

2. Refer to the *MVME143 MPU VMEmodule User's Manual* and configure header J1 for the user's particular application. J1 enables or disables the system controller function of the MVME143.

3. Be sure that the two 64K x 8 143Bug EPROMs are installed in sockets XU3 (odd bytes, odd BXX label) and XU12 (even bytes, even BXX label) on the MVME143 module.

4. Refer to the set-up procedure for the user's particular chassis or system for details concerning the installation of the MVME143.

5. Connect the terminal which is to be used as the 143Bug system console to connector J9 (port 1) on the front panel. Set up the terminal as follows:

   - eight bits per character
   - one stop bit per character
   - parity disabled (no parity)
   - 9600 baud to agree with default baud rate of the MVME143 ports at power-up.

   After power-up, the baud rate of the J9 port (port 1) can be reconfigured by using the Port Format (PF) command of the 143Bug debugger.

NOTE

In order for high-baud rate serial communication between 143Bug and the terminal to work, the terminal must do some handshaking. If the terminal being used does not do hardware handshaking via the CTS line, then it must do XON/XOFF handshaking. If the user gets garbled messages and missing characters, then he should check the terminal to make sure XON/XOFF handshaking is enabled.

6. If it is desired to connect device(s) (such as a host computer system or a serial printer) to port 2 and/or port 3 on the MVME143, connect the appropriate cables and configure the port(s) as detailed in the *MVME143 MPU VMEmodule User's Manual*. After power-up, these ports can be reconfigured by using the PF command of the 143Bug debugger.

7. Power up the system. 143Bug executes self-checks and displays the debugger prompt "143-Bug>".

   If after a delay, the 143Bug begins to display test result messages on the bottom line of the screen in rapid succession, the MVME143 is in the Bug "system" mode. If this is not the desired mode of operation, then press the ABORT switch on the front panel of the MVME143. When the MENU is displayed, enter a 3 to go to the system debugger. The environment may be changed by using the set environment (ENV) command. Refer to the *Set Environment To Bug/Operating System* paragraph in Chapter 3.

   When power is applied to the MVME143, bit 2 at location $FFF80001 (Multi-Functional Peripheral (MFP) general purpose I/O register) is set to 1 indicating that power was just applied. (Refer to *MVME143 MPU VMEmodule User's Manual* for a description of the MFP.) This bit is tested within the "Reset" logic path to see if the power-up confidence test needs to be executed.

   If the power-up confidence test is successful and no failures are detected, the firmware monitor comes up normally, with the FAIL LED off.

   If the confidence test fails, the test is aborted when the first fault is encountered, and the FAIL LED remains on. If possible, one of the following messages is displayed:

```
... 'CPU Register test failed'
... 'CPU Instruction test failed'
... 'ROM test failed'
... 'RAM test failed'
... 'CPU Addressing Modes test failed'
... 'Exception Processing test failed'
... 'Battery low (data may be corrupted)'
... 'Non-volatile RAM access error'
```

   The firmware monitor comes up with the FAIL LED on.

Autoboot

Autoboot is a software routine that can be enabled by a flag in the battery backed-up RAM to provide an independent mechanism for booting an operating system. When enabled by the Autoboot (AB) command, this autoboot routine automatically starts a boot from the controller and device specified. It also passes on the specified default string. This normally occurs at power-up only, but the user may change it to boot up at any board reset. NOAB disables the routine but does not change the specified parameters. The Autoboot enable/disable command details are described in Chapter 3. The default (factory-delivered) condition is with autoboot disabled.

If, at power-up, Autoboot is enabled and the drive and controller numbers provided are valid, the following message is displayed upon the system console:

"Autoboot in progress... To Abort hit <BREAK>"

Following this message there is a delay while the debug firmware waits for the various controllers and drives to come up to speed. Then the actual I/O is begun: the program pointed to within the volume ID of the media specified is loaded into RAM and control passed to it. If, however, during this time, the user wants to gain control without Autoboot, hit the <BREAK> key.

CAUTION

This information applies to the MVME350 module but not the MVME143. Although streaming tape can be used to Autoboot, the same power supply must be connected to the streaming tape drive, controller, and the MVME143. At power-up, the tape controller positions the streaming tape to load point where the volume ID can correctly be read and used.

If, however, the MVME143 loses power but the controller does not, and the tape happen not to be at the load point, the sequences of commands required (attach and rewind) cannot be given to the controller and Autoboot is not successful.

ROMboot

This function is enabled by the ROMboot (RB) command and executed at power-up (optionally also at reset), assuming there is valid code in the ROMs (or optionally elsewhere on the module or VMEbus) to support it. If ROMboot code is installed, a user-written routine is given control (if the routine meets the format requirements). One use of ROMboot might be resetting SYSFAIL* on an unintelligent controller module. The NORB command disables the function.

For a user's module to gain control through the ROMboot linkage, four requirements must be met:

a. Power must have just been applied (but the RB command can change this to also respond to any reset).

b. The user's routine must be located within the MVME143 ROM memory map (but the RB command can change this to any other portion of the onboard memory, or even offboard VMEbus memory).

c. The ASCII string "BOOT" must be located within the specified memory range.

d. The user's routine must pass a checksum test, which ensures that this routine was really intended to receive control at power-up.

To prepare a module for ROMboot, the Checksum (CS) command must be used. When the module is ready it can be loaded into RAM, and the checksum generated, installed, and verified with the CS command. (Refer to the *CS Command* paragraph and examples in Chapter 3.)

The format of the beginning of the routine is as follows:

MODULE OFFSET	LENGTH	CONTENTS	DESCRIPTION
$00	4 bytes	BOOT	ASCII string indicating possible routine; checksum must be zero, too.
$04	4 bytes	Entry Address	Longword offset from "BOOT".
$08	4 bytes	Routine Length	Longword, includes length from "BOOT" to and including checksum.
$0C	?	Routine Name	ASCII string containing routine name.

By convention within Motorola, the last three bytes of ROM contain the firmware version number, checksum, and socket number. In this environment, the length would contain the ASCII string "BOOT", through and including the socket number; however, the user wishing to make use of ROMboot does not have to fill a complete ROM. Any partial amount is acceptable, as long as the length reflects where the checksum is correct.

ROMboot searches for possible routines starting at the start of the memory map first and checks for the "BOOT" indicator. Two events are of interest for any location being tested:

a. The map is searched for the ASCII string "BOOT".

b. If the ASCII string "BOOT" is found, it is still undetermined whether the routine is meant to gain control at power-up or reset. To verify that this is the case, the bytes starting from "BOOT" through the end of the routine (as defined by the 4-byte length at offset $8) are run through the self-test checksum routine. If both the even and odd bytes are zero, it is established that the routine was meant to be used for ROMboot.

Under control of the RB command, the sequence of searches is as follows:

a. Search direct address for "BOOT".

b. Search user non-volatile RAM (first 1Kb of battery back-up RAM).

c. Search complete ROM map.

d. Search local RAM (if RB command has selected to operate on any reset), at all 8Kb boundaries starting at $00004000.

e. Search the VMEbus map (if so selected by the RB command) on all 8Kb boundaries starting at the end of the onboard RAM.

The following example performs the following:

a. Outputs a (CR)(LF) sequence to the default output port.

b. Displays the date and time from the current cursor position.

c. Outputs two more (CR)(LF) sequences to the default output port.

d. Returns control to 143Bug.

NOTE

This example assumes that the target code is temporarily loaded into the MVME143 RAM. However, an emulator such as the Motorola HDS-300 or HDS-400 could easily be used to load and modify the target code in its actual execution location.

SAMPLE ROMboot ROUTINE – Module preparation including calculation of checksum

The target code is first assembled and linked, leaving $00 in the even and odd locations destined to contain the checksum.

Load the routine into RAM (with S-records via the LO command, or from a VERSAdos disk using IOP).

```
143-Bug>mds 6000
```

Display entire module (zero checksums at $0000602C and $0000602D).

```
00006000 424F 4F54 0000 0018  0000 002E 5465 7374   BOOT........Test
00006010 2052 4F4D 424F 4F54  4E4F 0026 4E4F 0052    ROMbootNO.&NO.R
00006020 4E4F 0026 4E4F 0026  4E4F 0063 0000 0000   NO.&NO.&NO.c....
00006030 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006040 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006050 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006060 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006070 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006080 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006090 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060A0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060B0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060C0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060D0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060E0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060F0 0000 0000 0000 0000  0000 0000 0000 0000   ................
```

```
143-Bug>md 6018;di
```

Disassemble executable instructions.

```
00006018 4E4F0026            SYSCALL     .PCRLF
0000601C 4E4F0052            SYSCALL     .RTC_DSP
00006020 4E4F0026            SYSCALL     .PCRLF
00006024 4E4F0026            SYSCALL     .PCRLF
00006028 4E4F0063            SYSCALL     .RETURN
0000602C 00000000            ORI.B       #$0,D0
00006030 00000000            ORI.B       #$0,D0
00006034 00000000            ORI.B       #$0,D0
```

```
143-Bug>CS 6000 602E
Physical Address=00006000 0000602D through 602E
 (Even Odd)=F99F
```

Perform checksum on locations 6000 (refer to the CS Command paragraph in Chapter 3).

```
143-Bug>M 602C;B
0000602C 00 ?F9
0000602D 00 ?9F.
```

Insert checksum into bytes $602C,$602D.

```
143-Bug>CS 6000 602E
Physical Address=00006000 0000602D
     (Even Odd)=0000
```

Verify that the checksum is correct.

```
143-Bug>mds 6000
```

Again display entire module (now with checksums).

```
00006000 424F 4F54 0000 0018  0000 002E 5465 7374  BOOT........Test
00006010 2052 4F4D 424F 4F54  4E4F 0026 4E4F 0052   ROMbootNO.&NO.R
00006020 4E4F 0026 4E4F 0026  4E4F 0063 F99F 0000  NO.&NO.&NO.cy...
00006030 0000 0000 0000 0000  0000 0000 0000 0000  ................
00006040 0000 0000 0000 0000  0000 0000 0000 0000  ................
00006050 0000 0000 0000 0000  0000 0000 0000 0000  ................
00006060 0000 0000 0000 0000  0000 0000 0000 0000  ................
00006070 0000 0000 0000 0000  0000 0000 0000 0000  ................
00006080 0000 0000 0000 0000  0000 0000 0000 0000  ................
00006090 0000 0000 0000 0000  0000 0000 0000 0000  ................
000060A0 0000 0000 0000 0000  0000 0000 0000 0000  ................
000060B0 0000 0000 0000 0000  0000 0000 0000 0000  ................
000060C0 0000 0000 0000 0000  0000 0000 0000 0000  ................
000060D0 0000 0000 0000 0000  0000 0000 0000 0000  ................
000060E0 0000 0000 0000 0000  0000 0000 0000 0000  ................
000060F0 0000 0000 0000 0000  0000 0000 0000 0000  ................
143-Bug>
```

The EPROMs can now be programmed and inserted.

```
COLD Start
143-Bug> Searching for ROM Boot
```

Now power can be removed, and when it is reapplied the module receives control and displays the expected message.

Restarting The System

The user can initialize the system to a known state in three different ways. Each has characteristics which make it more appropriate than the others in certain situations.

Reset

Pressing and releasing the MVME143 front panel RESET switch initiates a system reset. COLD and WARM reset modes are available. By default, 143Bug is in COLD mode (refer to the *RESET Command* paragraph in Chapter 3). During COLD reset, a total system initialization takes place, as if the MVME143 had just been powered up. The breakpoint table and offset registers are cleared. The target registers are invalidated. Input and output character queues are cleared. Onboard devices (timer, serial ports, etc.) are reset. All static variables (including disk device and controller parameters) are restored to their default states. Serial ports are reconfigured to their default state.

During WARM reset, the 143Bug variables and tables are preserved, as well as the target state registers and breakpoints. If the particular MVME143 is the system controller, then a system reset is issued to the VMEbus and other modules in the system are reset as well.

Reset must be used if the processor ever halts (as evidenced by the MVME143 illuminated STATUS LED), for example after a double bus fault; or if the 143Bug environment is ever lost (vector table is destroyed, etc.).

Abort

Abort is invoked by pressing and releasing the ABORT switch on the MVME143 front panel. Whenever abort is invoked when executing a user program (running target code), a "snapshot" of the processor state is captured and stored in the target registers. (When working in the debugger, abort captures and stores only the program counter, status register, and format/vector information.) For this reason, abort is most appropriate when terminating a user program that is being debugged. Abort should be used to regain control if the program gets caught in a loop, etc. The target PC, stack pointers, etc., help to pinpoint the malfunction.

Abort generates a level seven interrupt (non-maskable). The target registers, reflecting the machine state at the time the ABORT switch was pushed, are displayed to the screen.

Any breakpoints installed in the user code are removed and the breakpoint table remains intact. Control is returned to the debugger.

Reset and Abort - Restore Battery Backed Up RAM

Pressing both the RESET and ABORT switches at the same time and releasing the RESET switch before the ABORT switch initiates an onboard reset and a restore of Key Bug dependent BBRAM variables.

During the start of the reset sequence, if abort is invoked, then the following conditions are set in BBRAM:

- Memory sized flag is cleared (onboard memory is sized on this reset).
- AUTOBOOT (Bug "normal") is turned off.
- ROMboot (Bug "normal") is turned off.
- Environment set for Bug "normal" mode.
- Operating system set for SYSTEM V/68.

Break

A "Break" is generated by pressing and releasing the BREAK key on the terminal keyboard. Break does not generate an interrupt. The only time break is recognized is when characters are sent or received by the console port. Break removes any breakpoints in the user code and keeps the breakpoint table intact. Break does not, however, take a snapshot of the machine state nor does it display the target registers.

Many times it is desired to terminate a debugger command prior to its completion, for example, the display of a large block of memory. Break allows the user to terminate the command without overwriting the contents of the target registers, as would be done if abort were used.

Memory Requirements

The program portion of 143Bug is approximately 128Kb of code. These EPROM sockets on the MVME143 are mapped at locations $FFF00000 through $FFF1FFFF. However, 143Bug code is position-independent and executes anywhere in memory.

143Bug requires a minimum of 16Kb of read/write memory to operate. This memory is usually the MVME143 onboard read/write memory, requiring stand-alone operation of the MVME143. The user selects the address at which onboard shared DRAM appears

from the VMEbus, by programming the PI/T port A data register (refer to the *MVME143 MPU VMEmodule User's Manual*) as shown below.

The onboard shared DRAM address is controlled by U52 and by control bits SLVA3-SLVA0. U52 selects one 64Mb block within the 4Gb range for the MVME143. The default factory program for U52 puts the base address of this 64Mb block at $00000000. Control bits SLVA3-SLVA0 then select one of the 16 possible positions within this 64Mb block for the 4Mb of onboard shared DRAM. Note that SLVA3-SLVA0 also define the mailbox interrupt address.

Shared DRAM Address Map on VMEbus

SLVA3	SLVA2	SLVA1	SLVA0	OFFBOARD ADDRESS	MAILBOX ADDRESS
					Short I/O Space
0	0	0	0	$00000000	$XXXXFF00
0	0	0	1	$00400000	$XXXXFF10
0	0	1	0	$00800000	$XXXXFF20
0	0	1	1	$00C00000	$XXXXFF30
0	1	0	0	$01000000	$XXXXFF40
0	1	0	1	$01400000	$XXXXFF50
0	1	1	0	$01800000	$XXXXFF60
0	1	1	1	$01C00000	$XXXXFF70
1	0	0	0	$02000000	$XXXXFF80
1	0	0	1	$02400000	$XXXXFF90
1	0	1	0	$02800000	$XXXXFFA0
1	0	1	1	$02C00000	$XXXXFFB0
1	1	0	0	$03000000	$XXXXFFC0
1	1	0	1	$03400000	$XXXXFFD0
1	1	1	0	$03800000	$XXXXFFE0
1	1	1	1	$03C00000	$XXXXFFF0

SLVA3*- SLVA0* — SLVA3*-SLVA0* define the offboard address of the onboard shared DRAM for accesses from the VMEbus. They also define the mail-box interrupt address in the short I/O space for the MVME143. Refer to shared DRAM address map on VMEbus for more details.

After reset, SLVA3-SLVA0 = %1111. Therefore, software must initialize them if a different value is desired for proper system operation.

Moreover, the user may select the onboard DRAM to respond to either 32-bit address accesses only or 24-bit and 32-bit address accesses by the VMEbus. Control bit

VMEA24* defines the address size for the VMEbus slave interface. The MVME143 onboard DRAM responds to the VMEbus accesses only when the addresses match and the address modifiers (AM0–AM5) indicate privileged or non-privileged, data or program space. Also, an MVME143 may not access its own onboard memory via the VMEbus.

The first 16Kb is used for 143Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME143 is reset, the target PC is initialized to the address corresponding to the beginning of the user space and the target stack pointers are initialized to addresses within the user space, with the target ISP set to the top of the user space.

The following abbreviated memory map for the MVME143 highlights addresses that might be of particular interest to the firmware monitor user. Note that addresses are assumed to be hexadecimal throughout this manual. In text, numbers may be preceded with a dollar sign ($) for identification as hexadecimal.

DRAM LOCATIONS	FUNCTION
00000000-000003FF	Target vector area
00000400-000007FF	Bug vector area
00000800-00000803	Multi-Processor Control Register (MPCR)
00000804-00000807	Multi-Processor Address Register (MPAR)
00000808-000037FF	Work area and stack for MVME143 debug

EPROM LOCATIONS	FUNCTION
FF800000-FFF00003	Supervisor stack address used when RESET switch is pressed
FF800004-FFF00007	Program Counter (PC) used when RESET switch is pressed
FF800008-FFF0000B	Size of code
FF80000C-FFF0000F	Reserved
FF83FFFA-FFF1FFFB	Even/odd revision number of the two monitor EPROMs
FF83FFFC-FFF1FFFD	Even/odd socket number where monitor EPROMs reside
FF83FFFE-FFF1FFFF	Even/odd checksum of the two monitor EPROMs $FF830000 – $FFF1FFFF in sockets XU3 (odd), XU12 (even)
FFF20000-FFF3FFFF	Reserved for user $FFF20000 – $FFF3FFFF in sockets XU21 (odd), XU28 (even)

BATTERY BACKED-UP RAM LOCATIONS	FUNCTION
FFFE0000-FFFE03FF	Reserved for user
FFFE0400-FFFE05FF	Reserved for operating system use
FFFE0600-FFFE07F7	Reserved for bug use
FFFE0774-FFFE0777	End of memory + 1, set via memory sizing routine
FFFE0778-FFFE077A	Reserved
FFFE077B	Memory sizing flag
FFFE077C-FFFE07C5	Reserved
FFFE07C6	AUTOBOOT controller number, set via the AB command
FFFE07C7	AUTOBOOT device number, set via the AB command
FFFE07C8-FFFE07E3	AUTOBOOT string, set via the AB command
FFFE07E4-FFFE07E9	Offboard address, set via the OBA command
FFFE07EA-FFFE07EF	ROMboot direct address, set via the RB command
FFFE07F0	AUTOBOOT enable switch, set via the [NO] AB command (Y/N)
FFFE07F1	AUTOBOOT at power-up switch, set via the AB command (Y/N)
FFFE07F2	ROMboot enable switch, set via the [NO] RB command (Y/N)
FFFE07F3	ROMboot from VMEbus switch, set via the RB command (Y/N)
FFFE07F4	ROMboot at power-up switch, set via the RB command (Y/N)
FFFE07F5	Reserved
FFFE07F6	Bug/System switch, set via the ENV command (B/S)
FFFE07F7	SYSTEM V/68 or VERSAdos switch, set via ENV command (S/V)
FFFE07F8-FFFE07FF	Time-of-day clock

I/O HARDWARE ADDRESSES	FUNCTION
FFF80026-FFF8002F	Serial port 1
FFFA0000-FFFA0001	Serial port 2
FFFA0002-FFFA0003	Serial port 3
FFF80000-FFF8FFFF	MFP registers
FFF90000-FFF9FFFF	PI/T registers
FFFA0000-FFFBFFFF	SCC registers

Disk I/O Support

143Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 143Bug consist of command-level disk operations, disk I/O system calls (only via the TRAP #15 instruction) for use by user programs, and defined data structures for disk parameters.

Parameters such as the address where the module is mapped and the type and number of devices attached to the controller module are kept in tables by 143Bug. Default values for these parameters are assigned at power-up and cold-start reset, but may be altered as described in the *Default 143Bug Controller And Device Parameters* paragraph in this chapter.

Appendix E contains a list of the controllers presently supported, as well as a list of the default configurations for each controller.

Blocks Versus Sectors

The logical block defines the unit of information for disk devices. A disk is viewed by 143Bug as a storage area divided into logical blocks. By default, the logical block size is set to 256 bytes for every block device in the system. The block size can be changed on a per device basis with the IOT command.

The sector defines the unit of information for the media itself, as viewed by the controller. The sector size varies for different controllers, and the value for a specific device can be displayed and changed with the IOT command.

When a disk transfer is requested, the start and size of the transfer is specified in blocks. 143Bug translates this into an equivalent sector specification, which is then passed on to the controller to initiate the transfer. If the conversion from blocks to sectors yields a fractional sector count, an error is returned and no data is transferred.

Disk I/O Via 143Bug Commands

These following 143Bug commands are provided for disk I/O. Detailed instructions for their use are found in Chapter 3. When a command is issued to a particular controller LUN and device LUN, these LUNs are remembered by 143Bug so that the next disk command defaults to use the same controller and device.

IOP (Physical I/O To Disk)

IOP allows the user to read or write blocks of data, or to format the specified device in a certain way. IOP creates a command packet from the arguments specified by the user, and then invokes the proper system call function to carry out the operation.

IOT (I/O Teach)

IOT allows the user to change any configurable parameters and attributes of the device. In addition, it allows the user to see the controllers available in the system.

IOC (I/O Control)

IOC allows the user to send command packets as defined by the particular controller directly. IOC can also be used to look at the resultant device packet after using the IOP command.

BO (Bootstrap Operating System)

BO reads an operating system or control program from the specified device into memory, and then transfers control to it.

BH (Bootstrap And Halt)

BH reads an operating system or control program from a specified device into memory, and then returns control to 143Bug. It is used as a debugging tool.

Disk I/O Via 143Bug System Calls

All operations that actually access the disk are done directly or indirectly by 143Bug TRAP #15 system calls. (The command-level disk operations provide a convenient way of using these calls without writing and executing a program.)

The following system calls are provided to allow user programs to do disk I/O:

- .DSKRD – Disk read. System call to read blocks from a disk into memory.
- .DSKWR – Disk write. System call to write blocks from memory onto a disk.
- .DSKCFIG – Disk configure. This function allows the user to change the configuration of the specified device.
- .DSKFMT – Disk format. This function allows the user to send a format command to the specified device.
- .DSKCTRL – Disk control. This function is used to implement any special device control functions that cannot be accommodated easily with any of the other disk functions.

Refer to Chapter 5 for information on using these and other system calls.

To perform a disk operation, 143Bug must eventually present a particular disk controller module with a controller command packet which has been especially prepared for that type of controller module. (This is accomplished in the respective controller driver module.) A command packet for one type of controller module usually does not have the same format as a command packet for a different type of module. The system call facilities which do disk I/O accept a generalized (controller-independent) packet format as an argument, and translate it into a controller-specific packet, which is then sent to the specified device. Refer to the *Invoking System Calls Through TRAP #15* paragraph in Chapter 5 for details on the format and construction of these standardized "user" packets.

The packets which a controller module expects to be given vary from controller to controller. The disk driver module for the particular hardware module (board) must take the standardized packet given to a trap function and create a new packet which is specifically tailored for the disk drive controller it is sent to. Refer to documentation on the particular controller module for the format of its packets, and for using the IOC command.

Default 143Bug Controller And Device Parameters

143Bug initializes the parameter tables for a default configuration of controllers and devices (refer to Appendix E). If the system needs to be configured differently than this default configuration (for example, to use a 70Mb Winchester drive where the default is a 40Mb Winchester drive), then these tables must be changed.

There are two ways to change the parameter tables. If BO or BH is invoked, the configuration area of the disk is read and the parameters corresponding to that device are rewritten according to the parameter information contained in the configuration area. (Appendix D has more information on the disk con- figuration area.) This is a temporary change. If a cold-start reset occurs, then the default parameter information is written back into the tables.

Alternately, the IOT command may be used to manually reconfigure the parameter table for any controller and/or device that is different from the default. This is also a temporary change and is overwritten if a cold-start reset occurs.

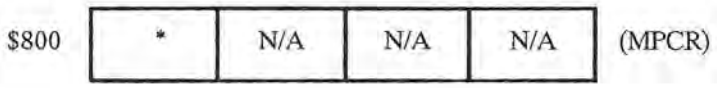
Disk I/O Error Codes

143Bug returns an error code if an attempted disk operation is unsuccessful. Refer to Appendix F for an explanation of disk I/O error codes.

Multiprocessor Support

The MVME143 dual-port RAM feature makes the shared RAM available to remote processors as well as to the local processor.

A remote processor can initiate program execution in the local MVME143 dual-port RAM by issuing a remote GO command using the Multiprocessor Control Register (MPCR). The MPCR, located at shared RAM location $800, contains one of two longwords used to control communication between processors. The MPCR contents are organized as follows:



The status codes stored in the MPCR are of two types:

- Status returned (from the monitor)
- Status set (by the bus master)

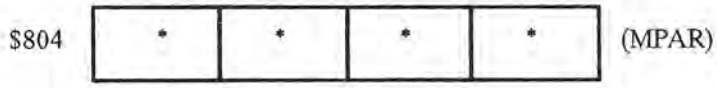
The status codes that may be returned from the monitor are:

HEX 0 (HEX 00) -- Wait. Initialization not yet complete.
ASCII R (HEX 52) -- Ready. The firmware monitor is watching for a change.
ASCII E (HEX 45) -- Code pointed to by the MPAR address is executing.

The status codes that may be set by the bus master are:

ASCII G (HEX 47) -- Use Go Direct (GD) logic specifying the MPAR address.
ASCII B (HEX 42) -- Install breakpoints using the Go (G) logic.

The Multiprocessor Address Register (MPAR), located in shared RAM location $804, contains the second of two longwords used to control communication between processors. The MPAR contents specify the address at which execution for the remote processor is to begin if the MPCR contains a G or B. The MPAR is organized as follows:



At power-up, the debug monitor self-test routines initialize RAM, including the memory locations used for multi-processor support ($800 through $807).

The MPCR contains $00 at power-up, indicating that initialization is not yet complete. As the initialization proceeds, the execution path comes to the "prompt" routine. Before sending the prompt, this routine places an R in the MPCR to indicate that initialization is complete. Then the prompt is sent.

If no terminal is connected to the port, the MPCR is still polled to see whether an external processor requires control to be passed to the dual-port RAM. If a terminal does respond, the MPCR is polled for the same purpose while the serial port is being polled for user input.

An ASCII G placed in the MPCR by a remote processor indicates that the Go Direct type of transfer is requested. An ASCII B in the MPCR indicates that breakpoints are to be armed before control is transferred (as with the Go Command).

In either sequence, an E is placed in the MPCR to indicate that execution is underway just before control is passed to RAM. (Any remote processor could examine the MPCR contents.)

If the code being executed in dual-port RAM is to reenter the debug monitor, a TRAP #15 call using function $0063 (SYSCALL .RETURN) returns control to the monitor with a new display prompt. Note that every time the debug monitor returns to the prompt, an R is moved into the MPCR to indicate that control can be transferred once again to a specified RAM location.

Diagnostic Facilities

Included in the 143Bug package is a complete set of hardware diagnostics intended for testing and troubleshooting of the MVME143 (refer to Chapter 6). In order to use the diagnostics, the user must switch directories to the diagnostic directory. If in the debugger directory, the user can switch to the diagnostic directory by entering the debugger command Switch Directories (SD). The diagnostic prompt ("143-Diag>") should appear. Refer to Chapter 6 for complete descriptions of the diagnostic routines available and instructions on how to invoke them. Note that some diagnostics depend on restart defaults that are set up only in a particular restart mode. Refer to the documentation on a particular diagnostic for the correct mode.

The following publications provide additional information. If not shipped with this product, they may be purchased from Motorola's Literature Distribution Center, 616 West 24th Street, Tempe, Arizona 85282; phone (602) 994-6561. Non-Motorola documents may be obtained from the sources listed.

DOCUMENT TITLE	MOTOROLA PUBLICATION NUMBER
MVME050 System Controller Module and MVME701/MVME701A I/O Transition Module User's Manual	MVME050
MVME143 MPU VMEmodule User's Manual	MVME143
MVME319 Intelligent Disk/Tape Controller User's Manual	MVME319
MVME320 VMEbus Disk Controller Module User's Manual	MVME320
MVME320A VMEbus Disk Controller Module User's Manual	MVME320A
MVME320B VMEbus Disk Controller Module User's Manual	MVME320B
MVME321 Intelligent Disk Controller User's Manual	MVME321
MVME321 IPC Firmware User's Guide	MVME321FW
MVME350 Streaming Tape Controller VMEmodule User's Manual	MVME350
MVME350 IPC Firmware User's Guide	MVME350FW
MVME360 SMD Disk Controller User's Manual	MVME360
VERSAdos to VME Hardware and Software Configuration User's Manual	MVMEVDOS
MC68030 32-Bit Microprocessor User's Manual	MC68030UM
MC68882 Floating-Point Coprocessor User's Manual	MC68882UM
M68000 Family VERSAdos System Facilities Reference Manual	M68KVSF

MK48T02 2K x 8 Zeropower/Timekeeper RAM Data Sheet, Thompson Components Mostek, 1310 Electronics Drive, Carrollton, TX 75606

Z8530A Serial Communications Controller data sheet; Zilog, Inc., Corporate Communications, Building A, 1315 Dell Ave, Campbell, California 95008

CHAPTER 2
USING THE 143Bug DEBUGGER

2

Entering Debugger Command Lines

143Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt ("143-Bug>") appears on the terminal screen, then the debugger is ready to accept commands.

As the command line is entered, it is stored in an internal buffer. Execution begins only after the carriage return is entered, thus allowing the user to correct entry errors.

When a command is entered, the debugger executes the command and the prompt reappears. However, if the command entered causes execution of user target code, for example "GO", then control may or may not return to the debugger, depending on what the user program does. For example, if a breakpoint has been specified, then control returns to the debugger when the breakpoint is encountered during execution of the user program. Alternately, the user program could return to the debugger by means of the TRAP #15 function ".RETURN" (described in Chapter 5). Refer to the *GD* and *GO Command* paragraphs in Chapter 3.

In general, a debugger command is made up of the following parts:

a. The command identifier (i.e., "MD" or "md" for the Memory Display command). Note that either uppercase or lowercase is allowed.

b. A port number if the command is set up to work with more than one port.

c. At least one intervening space before the first argument.

d. Any required arguments, as specified by command.

e. An option field, set off by a semicolon (;) to specify conditions other than the default conditions of the command.

When entering a command at the prompt, the following control codes may be entered for limited command line editing, if necessary, using the control characters described below.

2

NOTE

The presence of the upward caret, "^", before a character indicates that the Control, "CTRL" key must be held down while striking the character key.

^X	(cancel line)	The cursor is backspaced to the beginning of the line. If the terminal port is configured with the hardcopy or TTY option (refer to the *PF Command* paragraph in Chapter 3), then a carriage return and line feed is issued along with another prompt.
^H	(backspace)	The cursor is moved back one position. The character at the new cursor position is erased. If the hardcopy option is selected, a "/" character is typed along with the deleted character.
<DEL>	(delete or rubout)	Performs the same function as ^H.
^D	(redisplay)	The entire command line as entered so far is redisplayed on the following line.

When observing output from any 143Bug command, the XON and XOFF characters which are in effect for the terminal port may be entered to control the output, if the XON/XOFF protocol is enabled (default). These characters are initialized to ^S and ^Q respectively by 143Bug but may be changed by the user using the PF command. In the initialized (default) mode, operation is as follows:

^S	(wait)	Console output is halted.
^Q	(resume)	Console output is resumed.

The following conventions are used in the command syntax, examples, and text in this manual.

boldface strings	A boldface string is a literal such as a command or a program name, and is to be typed just as it appears.
italic strings	An italic string is a "syntactic variable" and is to be replaced by one of a class of items it represents.
|	A vertical bar separating two or more items indicates that a choice is to be made; one or more of the items separated by this symbol may be selected.

[ ]	Square brackets enclose an item that is optional. The item may appear zero or one time.
[ ]. . .	Square brackets followed by an ellipsis (three dots) enclose an item that is optional/repetitive. The item may appear zero or more times.
[ ]	Boldface brackets are required characters.

Operator inputs are to be followed by a carriage return. The carriage return is shown, as (**CR**), only if it is the only input required.

Syntactic Variables

The following syntactic variables are encountered in the command descriptions which follow. In addition, other syntactic variables may be used and are defined in the particular command description in which they occur.

del	Delimiter; either a comma or a space.
exp	Expression (described in detail below).
addr	Address (described in detail below).
count	Count; the syntax is the same as for <EXP>.
range	A range of memory addresses which may be specified either by *addr del addr* or by *addr* : *count*.
text	An ASCII string of up to 255 characters, delimited at each end by the single quote mark (').

Expression As A Parameter

An expression can be one or more numeric values separated by the arithmetic operators: plus (+), minus (-), multiplied by (*), divided by (/), logical AND (&), shift left (<<), or shift right (>>).

Numeric values may be expressed in either hexadecimal, decimal, octal, or binary by immediately preceding them with the proper base identifier.

2

BASE	IDENTIFIER	EXAMPLES
Hexadecimal	$	$FFFFFFFF
Decimal	&	&1974, &10-&4
Octal	@	@456
Binary	%	%1000110

If no base identifier is specified, then the numeric value is assumed to be hexadecimal.

A numeric value may also be expressed as a string literal of up to four characters. The string literal must begin and end with the single quote mark ('). The numeric value is interpreted as the concatenation of the ASCII values of the characters. This value is right-justified, as any other numeric value would be.

STRING LITERAL	NUMERIC VALUE (IN HEXADECIMAL)
'A'	41
'ABC'	414243
'TEST'	54455354

Evaluation of an expression is always from left to right unless parentheses are used to group part of the expression. There is no operator precedence. Subexpressions within parentheses are evaluated first. Nested parenthetical subexpressions are evaluated from the inside out.

Valid expression examples:

EXPRESSION	RESULT (IN HEX)	NOTES
FF0011	FF0011	
45+99	DE	
&45+&99	90	
@35+@67+@10	5C	
%10011110+%1001	A7	
88<<4	880	shift left
AA&F0	A0	logical AND

The total value of the expression must be between 0 and $FFFFFFFF.

Address As A Parameter

Many commands use *addr* as a parameter. The syntax accepted by 143Bug is similar to the one accepted by the MC68030 one-line assembler. All control addressing modes are allowed. An "address + offset register" mode is also provided.

Address Formats

The address formats which are acceptable for address parameters in debugger command lines are summarized in Table 2-1.

TABLE 2-1. Formats for Debugger Address Parameters

FORMAT	EXAMPLE	DESCRIPTION
N	140	Absolute address + contents of automatic offset register.
N+Rn	130+R5	Absolute address + contents of the specified offset register (not an assembler-accepted syntax).
(An)	(A1)	Address register indirect.
(d,An) or d(An)	(120,A1) 120(A1)	Address register indirect with displacement (two formats accepted).

TABLE 2-1. Formats for Debugger Address Parameters (cont'd)

FORMAT	EXAMPLE	DESCRIPTION
(d,An,Xn) or d(An,Xn)	(&120,A1,D2) &120(A1,D2)	Address register indirect with index & displacement (two formats accepted).
([bd,An,Xn],od)	([C,A2,A3],&100)	Memory indirect pre-indexed.
([bd,An],Xn,od)	([12,A3],D2,&10)	Memory indirect postindexed.

For the memory indirect modes, fields can be omitted. For example, three of many permutations are as follows:

([,An],od)	([,A1],4)
([bd])	([FC1E])
([bd,,Xn])	([8,,D2])

NOTES: N - Absolute address (any valid expression)
An - Address register n
Xn - Index register n (An or Dn)
d - Displacement (any valid expression)
bd - Base displacement (any valid expression)
od - Outer displacement (any valid expression)
n - Register number (0 through 7)
Rn - Offset register n

Offset Registers

Eight pseudo-registers (R0-R7) called offset registers are used to simplify the debugging of relocatable and position-independent modules. The listing files in these types of programs usually start at an address (normally 0) that is not the one in which they are loaded, so it is harder to correlate addresses in the listing with addresses in the loaded program. The offset registers solve this problem by taking into account this difference and forcing the display of addresses in a relative address+offset format. Offset registers have adjustable ranges and may even have overlapping ranges. The range for each offset register is set by two addresses: base and top. Specifying the base and top addresses for an offset register sets its range.

In the event that an address falls in two or more offset registers' ranges, the one that yields the least offset is chosen.

NOTE

Relative addresses are limited to 1Mb (5 digits), regardless of the range of the closest offset register.

Example: A portion of the listing file of a relocatable module assembled with the MC68030 VERSAdos Resident Assembler is shown below:

```
  1
  2                                *
  3                                * MOVE STRING SUBROUTINE
  4                                *
  5  0 00000000 48E78080           MOVESTR  MOVEM.L  D0/A0,-(A7)
  6  0 00000004 4280                        CLR.L    D0
  7  0 00000006 1018                        MOVE.B   (A0)+,D0
  8  0 00000008 5340                        SUBQ.W   #1,D0
  9  0 0000000A 12D8                LOOP    MOVE.B   (A0)+,(A1)+
 10  0 0000000C 51C8FFFC            MOVS    DBRA     D0,LOOP
 11  0 00000010 4CDF0101                    MOVEM.L  (A7)+,D0/A0
 12  0 00000014 4E75                        RTS
 13
 14                                        END
****** TOTAL ERRORS     0--
****** TOTAL WARNINGS   0--
```

The above program was loaded at address 0001327C. The disassembled code is shown next:

```
143-Bug>MD 1327C;DI
0001327C 48E78080           MOVEM.L   D0/A0,-(A7)
00013280 4280               CLR.L     D0
00013282 1018               MOVE.B    (A0)+,D0
00013284 5340               SUBQ.W    #1,D0
00013286 12D8               MOVE.B    (A0)+,(A1)+
00013288 51C8FFFC           DBF       D0,$13286
0001328C 4CDF0101           MOVEM.L   (A7)+,D0/A0
00013290 4E75               RTS
143-Bug>
```

By using one of the offset registers, the disassembled code addresses can be made to match the listing file addresses as follows:

```
143-Bug>OF R0
R0 =00000000 00000000? 1327C:16.
143-Bug>MD 0+R0;DI
00000+R0 48E78080          MOVEM.L   D0/A0,-(A7)
00004+R0 4280              CLR.L     D0
00006+R0 1018              MOVE.B    (A0)+,D0
00008+R0 5340              SUBQ.W    #1,D0
0000A+R0 12D8              MOVE.B    (A0)+,(A1)+
0000C+R0 51C8FFFC          DBF       D0,$A+R0
00010+R0 4CDF0101          MOVEM.L   (A7)+,D0/A0
00014+R0 4E75              RTS
143-Bug>
```

For additional information about the offset registers, refer to the *OF Command* paragraph in Chapter 3.

Port Numbers

Some 143Bug commands give the user the option of choosing the port which is to be used to input or output. The valid port numbers which may be used for these commands are:

0 – MVME143 RS–232C serial port 1
1 – MVME143 RS–232C serial port 2
2 – MVME143 RS–232C serial port 3

NOTE

These logical port numbers (0, 1, and 2) are referred to as "Serial Port 1", "Serial Port 2", and "Serial Port 3", respectively, by the MVME143 hardware documentation.

For example, the command DU1 (Dump S–records to Port 1) would actually output data to the device connected to the serial port labeled SERIAL PORT 2 on the P2 connector.

Entering And Debugging Programs

There are various ways to enter a user program into system memory for execution. One way is to create the program using the Memory Modify (MM) command with the assembler/disassembler option. The program is entered by the user one source line at a time. After each source line is entered, it is assembled and the object code is loaded to memory. Refer to Chapter 4 for complete details of the 143Bug Assembler/Disassembler.

Another way to enter a program is to download an object file from a host system. The program must be in S-record format (described in Appendix C) and may have been assembled or compiled on the host system. Alternately, the program may have been previously created using the 143Bug MM command as outlined above and stored to the host using the Dump (DU) command. A communication link must exist between the host system and the MVME143 port B. (Refer to hardware configuration details in the *Installation And Startup* paragraph in Chapter 1.) The file is downloaded from the host into MVME143 memory via the debugger Load (LO) command.

Another way is by reading in the program from disk, using one of the disk commands (BO, BH, IOP). Once the object code has been loaded into memory, the user can set breakpoints if desired and run the code or trace through it.

Calling System Utilities From User Programs

A convenient way of doing character input/output and many other useful operations has been provided so that the user does not have to write these routines into the target code. The user has access to various 143Bug routines via the MC68030 TRAP #15 instruction. Refer to Chapter 5 for details on the various TRAP #15 utilities available and how to invoke them from within a user program.

Preserving the Debugger Operating Environment

This paragraph explains how to avoid contaminating the operating environment of the debugger. 143Bug uses certain of the MVME143 onboard resources and may also use offboard system memory to contain temporary variables, exception vectors, etc. If the user disturbs resources upon which 143Bug depends, then the debugger may function unreliably or not at all.

2

143Bug Vector Table And Wordspace

As described in the *Memory Requirements* paragraph in Chapter 1, 143Bug needs 12Kb of read/write memory to operate and also allocates another 4Kb as user space for a total of 16Kb allocated. 143Bug reserves a 1024–byte area for a user program vector table area and then allocates another 1024–byte area and builds an exception vector table for the debugger itself to use. Next, 143Bug reserves space for static variables and initializes these static variables to predefined default values. After the static variables, 143Bug allocates space for the system stack and then initializes the system stack pointer to the top of this area.

With the exception of the first 1024–byte vector table area, the user must be extremely careful not to use the above–mentioned areas for other purposes. The user should refer to the *Memory Requirements* paragraph in Chapter 1 and to Appendix A to determine how to dictate the location of the reserved memory areas. If, for example, a user program inadvertently wrote over the static variable area containing the serial communication parameters, these parameters would be lost, resulting in a loss of communication with the system console terminal. If a user program corrupts the system stack, then an incorrect value may be loaded into the processor PC, causing a system crash.

Exception Vectors Used By 143Bug

The exception vectors used by the debugger are listed in Table 2–2. They must reside at the specified offsets in the target program vector table for the associated debugger facilities (breakpoints, trace mode, etc.) to operate.

TABLE 2–2. Exception Vectors Used by 143Bug

VECTOR OFFSET	EXCEPTION	143Bug FACILITY
$10	Illegal instruction	Breakpoints (used by BR, GO, GN, GT)
$24	Trace	T, TC, TT
$BC	TRAP #15	System calls (refer to Chapter 5)
$158	Level 7 interrupt	ABORT switch

When the debugger handles one of the exceptions listed above, the target stack pointer is left pointing past the bottom of the exception stack frame created; that is, it reflects the system stack pointer values just before the exception occurred. In this way, the operation

of the debugger facility (through an exception) is transparent to the user. Example: Trace one instruction using the debugger.

```
143-Bug>RD
PC   =00004000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =0000F830 MSP  =00005C18 ISP* =00006000 SFC  =0=FC
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=FC
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00006000
00004000 4AFC                ILLEGAL
143-Bug>T
PC   =00004000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =0000F830 MSP  =00005C18 ISP* =00006000 SFC  =0=FC
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=FC
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00006000
00004000 4AFC                ILLEGAL
143-Bug>
```

Notice that the value of the target stack pointer register (A7) has not changed even though a trace exception has taken place. The user program may either use the exception vector table provided by 143Bug or it may create a separate exception vector table of its own. The two following paragraphs detail these two methods.

Using 143Bug Target Vector Table

143Bug initializes and maintains a vector table area for target programs. A target program is any user program started by the bug, either manually with GO or Trace type commands or automatically with the BOot command. The start address of this target vector table area is the base address of the MVME143, determined as described in the *Memory Requirements* paragraph in Chapter 1. This address is loaded into the target-state Vector Base Register (VBR) at power-up and cold-start reset and can be observed by using the RD command to display the target-state registers immediately after power-up.

143Bug initializes the target vector table with the debugger vectors listed in Table 2-3 and fills the other vector locations with the address of a generalized exception handler (refer to the *143Bug Generalized Exception Handler* paragraph in this chapter). The target program may take over as many vectors as desired by simply writing its own exception vectors into the table. If the vector locations listed in Table 2-3 are overwritten, then the accompanying debugger functions are lost.

2

143Bug maintains a separate vector table for its own use in a 1Kb space elsewhere in the reserved memory space. In general, the user does not have to be aware of the existence of the debugger vector table. It is completely transparent to the user and the user should never make any modifications to the vectors contained in it.

Creating A New Vector Table

A user program may create a separate vector table in memory to contain its exception vectors. If this is done, then the user program must change the value of the VBR to point at the new vector table. In order to use the debugger facilities, the user can copy the proper vectors from the 143Bug vector table into the corresponding vector locations in the user vector table.

The vector for the 143Bug generalized exception handler (described in detail in the *143Bug Generalized Exception Handler* paragraph in this chapter) may be copied from offset $08 (Bus Error vector) in the target vector table to all locations in the user vector table where a separate exception handler is not used. This provides diagnostic support in the event that the user program is stopped by an unexpected exception. The generalized exception handler gives a formatted display of the target registers and identifies the type of exception. Example: a user routine which builds a separate vector table and then moves the VBR to point at it:

```
*
****  BUILDX - Build exception vector table ****
*
BUILDX  MOVEC.L   VBR,A0              Get copy of VBR.
        LEA       $10000,A1           New vectors at $10000.
        MOVE.L    $8(A0),D0           Get generalized exception vector.
        MOVE.W    $3FC,D1             Load count (all vectors).
LOOP    MOVE.L    D0,(A1,D1)          Store generalized exception vector.
        SUBQ.W    #4,D1
        BNE.B     LOOP                Initialize entire vector table.
        MOVE.L    $10(A0),$10(A1)     Copy breakpoints vector.
        MOVE.L    $24(A0),$24(A1)     Copy trace vector.
        MOVE.L    $BC(A0),$BC(A1)     Copy system call vector.
        LEA.L     COPROCC(PC),A2      Get user exception vector.
        MOVE.L    A2,$2C(A1)          Install as F-Line handler.
        MOVEC.L   A1,VBR              Change VBR to new table.
        RTS
        END
```

It may turn out that the user program uses one or more of the exception vectors that are required for debugger operation. Debugger facilities may still be used, however, if the user exception handler can determine when to handle the exception itself and when to pass the exception to the debugger.

When an exception occurs which the user wants to pass on to the debugger (ABORT, for example), the user exception handler must read the vector offset from the format word of the exception stack frame. This offset is added to the address of the 143Bug target program vector table (which the user program saved), yielding the address of the 143Bug exception vector. The user program then jumps to the address stored at this vector location, which is the address of the 143Bug exception handler.

The user program must make sure that there is an exception stack frame in the stack and that it is exactly the same as the processor would have created it for the particular exception before jumping to the address of the exception handler.

The following is an example of a user exception handler which can pass an exception along to the debugger:

```
*
****  EXCEPT - Exception handler ****
*
EXCEPT  SUBQ.L    #4,A7              Save space in stack for a PC value.
        LINK      A6,#0              Frame pointer for accessing PC space.
        MOVEM.L   A0-A5/D0-D7,-(SP)  Save registers.
        :
        : decide here if user code will handle exception, if so, branch...
        :
        MOVE.L    BUGVBR,A0          Pass exception to debugger; get VBR.
        MOVE.W    14(A6),D0          Get the vector offset from stack frame.
        AND.W     #$0FFF,D0          Mask off the format information.
        MOVE.L    (A0,D0.W),4(A6)    Store address of debugger exc handler.
        MOVEM.L   (SP)+,A0-A5/D0-D7  Restore registers.
        UNLK      A6
        RTS                          Put addr of exc handler into PC and go.
```

143Bug Generalized Exception Handler

143Bug has a generalized exception handler which it uses to handle all of the exceptions not listed in Table For all these exceptions, the target stack pointer is left pointing to the top of the exception stack frame created. Thus, if an unexpected exception occurs during execution of a user code segment, the user is presented with the exception stack frame to help determine the cause of the exception. The following example illustrates this:

Example: Bus error at address $F00000. It is assumed for this example that an access of memory location $F00000 will initiate Bus Error exception processing.

```
143-Bug>RD
PC   =00004000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =0000F830 MSP  =00005C18 ISP* =00006000 SFC  =0=FC
CACR =0=D:....I:...           CAAR =00000000 DFC  =0=FC
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00006000
00004000 4AFC                ILLEGAL
143-Bug>T
PC   =00004000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =0000F830 MSP  =00005C18 ISP* =00006000 SFC  =0=FC
CACR =0=D:....I:...           CAAR =00000000 DFC  =0=FC
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00006000
00004000 4AFC                ILLEGAL
143-Bug>
```

Notice that the target stack pointer is different. The target stack pointer now points to the last value of the exception stack frame that was stacked. The exception stack frame may now be examined using the MD command:

```
143-Bug>MD (A7):&44
00003FA4 A700 0000 2000 B008   3E2C 0145 0000 0027   '... .0.>,.E..,'
00003FB4 00F0 0000 00F0 0000   0000 1BCC 2039 0000   .p...p.....L 9..
00003FC4 0000 200A 0000 2008   0000 2006 0000 0000   .. ... ... ......
00003FD4 00F0 0000 100F 0487   0000 A700 4003 0000   .p........'.@...
00003FE4 0000 7FFF 0000 0000   C010 0000 0000 4000   ........@.....@.
00003FF4 0000 0000 FFF8 086C                        .....x.l
143-Bug>
```

Memory Management Unit Support

The Memory Management Unit (MMU) is supported in the 143Bug. An MMU confidence check is run at reset time to verify that registers can be accessed. It also insures that a context switch can be done successfully. Also, the commands RD, RM, MD, and MM have been extended to allow display and modification of MMU data in registers and in memory. MMU instructions can be assembled/disassembled with the DI option of the MD/MM commands. In addition, the MMU target state is saved and restored along with

the processor state as required when switching between the target program and 143Bug. Finally, there is a set of diagnostics to test functionality of the MMU.

At power-up/reset a MMU confidence check is executed. If an error is detected, the test is aborted and the message "MMU failed test" is displayed. If the test runs without errors, then the message "MMU passed test" is displayed and an internal flag is set. This flag is later checked by the bug when doing a task switch. The MMU state is saved and restored only if this flag is set.

The MMU defines the Double Longword (DL) data type, which is used when accessing the root pointers. All other registers are either byte, word, or longword registers.

The MMU registers are shown below, along with their data types in parentheses:

Address Translation Control (ATC) registers:

CRP	-	CPU Root Pointer Register	(DL)
SRP	-	Supervisor Root Pointer	(DL)
TC	-	Translation Control Register	(L)
TT0	-	Transparent Translation 0	(L)
TT1	-	Transparent Translation 1	(L)

Status Information registers:

MMUSR	-	MMU Status Register	(W)

For more information about the MMU, refer to the *MC68030 Enhanced 32-bit Microprocessor User's Manual.*

Function Code Support

The function codes identify the address space being accessed on any given bus cycle, and, in general, they are an *extension* of the address. (This becomes more obvious when using a MMU, because two identical logical addresses can be made to map to two different physical addresses. In this case, the function codes provide the additional information required to find the proper memory location.)

For this reason, the following debugger commands were changed to allow the specification of function codes:

MD	Memory display
MM	Memory modify
MS	Memory set
GO	Go to target program
GD	Go direct (no breakpoints)
GT	Go and set temporary breakpoint
GN	Go to next instruction
BR	Set breakpoint

The symbol "^" (up arrow or caret) following the address field indicates that a function code specification follows. The function code can be entered by specifying a valid function code mnemonic or by specifying a number between 0 and 7. The syntax for an address and function code specification is:

addr^fc

The valid function code mnemonics are:

FUNCTION CODE	MNEMONIC	DESCRIPTION
0	F0	Unassigned, reserved
1	UD	User Data
2	UP	User Program
3	F3	Unassigned, reserved
4	F4	Unassigned, reserved
5	SD	Supervisor Data
6	SP	Supervisor Program
7	CS	CPU Space Cycle

NOTE: Using an unassigned or reserved function code or mnemonic results in a Long Bus Error message.

Example: To change data at location $5000 in the user data space.

```
143-Bug>m 5000^ud
00005000^UD 0000 ? 1234.
143-Bug>
```

CHAPTER 3
THE 143Bug DEBUGGER COMMAND SET

Introduction

This chapter contains descriptions of each debugger command, with one or more examples of each. 143Bug debugger commands are summarized in Table 3-1.

TABLE 3-1. Debugger Commands

COMMAND MNEMONIC	TITLE
AB/NOAB	Autoboot Enable/Disable
BC	Block Compare
BF	Block of Memory Fill
BH	Bootstrap Operating System and Halt
BI	Block of Memory Initialize
BM	Block of Memory Move
BO	Bootstrap Operating System
BR/NOBR	Breakpoint Insert/Delete
BS	Block of Memory Search
BV	Block of Memory Verify
CS	Checksum
DC	Data Conversion
DU	Dump S-records
EEP	EEPROM Programming
ENV	Set Environment to Bug or Operating System
GD	Go Direct (Ignore Breakpoints)
GN	Go to Next Instruction
GO	Go Execute User Program
GT	Go to Temporary Breakpoint
HE	Help
IOC	I/O Control for Disk
IOP	I/O Physical (Direct Disk Access)
IOT	I/O "TEACH" for Configuring Disk Controller
LO	Load S-records from Host

TABLE 3-1. Debugger Commands (cont'd)

COMMAND MNEMONIC	TITLE
MA/NOMA	Macro Define/Display/Delete
MAE	Macro Edit
MAL/NOMAL	Enable/Disable Macro Expansion Listing
MAR/MAW	Save/Load Macros
MD	Memory Display
MENU	System Menu
MM	Memory Modify
MS	Memory Set
OBA	Set Memory Address from VMEbus
OF	Offset Registers Display/Modify
PA/NOPA	Printer Attach/Detach
PF/NOPF	Port Format/Detach
PS	Put RTC into Power Save Mode for Storage
RB/NORB	ROMboot Enable/Disable
RD	Register Display
REMOTE	Connect the Remote Modem to CSO
RESET	Cold/Warm Reset
RM	Register Modify
RS	Register Set
SD	Switch Directories
SET	Set Time and Date
T	Trace
TA	Terminal Attach
TC	Trace on Change of Control Flow
TIME	Display Time and Date
TM	Transparent Mode
TT	Trace to Temporary Breakpoint
VE	Verify S-records Against Memory

Each of the individual commands is described in the following pages. The command syntax is shown using the symbols explained in Chapter 2.

In the examples shown, all user input is in **bold**. This is done for clarity in understanding the examples (to distinguish between characters input by the user and characters output by 143Bug). The symbol (**CR**) represents the carriage return key on the user's terminal keyboard. The (**CR**) is shown only if the carriage return is the only user input.

Autoboot Enable/Disable

AB
NOAB

AB
NOAB

The AB command lets the user select the Logical Unit Number (LUN) for the controller and device, and the default string that may be used for an automatic boot function. (Refer to the *BO Command* in this chapter.) Appendix E lists all the possible LUNs.) The user also can select whether this occurs only at power-up, or at any board reset. These selections are stored in the battery backed-up RAM that is part of the MK48T02 RTC. The automatic boot function transfers control to the controller and device specified by the AB command.

The NOAB command disables the automatic boot function. (Refer to Chapter 1 for details on Autoboot.)

The as-delivered default condition is with the autoboot function not enabled.

Example:

```
143-Bug>ab
Controller LUN     = 00 ? (CR)
Device LUN         = 00 ? (CR)
Default string     = VME143.. ? (CR)
Boot at power-up only [Y,N] ? Y (CR)
```

Enable the autoboot function.
Select controller for boot.
Select device to boot from.
Select string to pass on.
If the user selects N, the MVME143 boots at any board reset.

```
At power-up only:
Auto Boot from Controller 0, Device 0, VME143..
143-Bug>noab
No Auto Boot from Controller 0, Device 0, VME143..
143-Bug>
```

NOAB disables the autoboot function, but does not change the parameters.

Block Of Memory Compare

BC

BC *range del addr* **[;B|W|L]**

options:

B - Byte
W - Word
L - Longword

The BC command compares the contents of the memory addresses defined by *range* to another place in memory, beginning at *addr*.

The option field is only allowed when *range* is specified using a count. In this case, the B, W, or L defines the size of data that the count is referring to. For example, a count of 4 with an option of L would mean to compare 4 long words (or 16 bytes) to the *addr* location. If the range beginning address is greater than the end address, an error results. An error also results if an option field is specified without a count in the range.

For the following examples, assume the following data is in memory.

```
143-Bug>MD 20000:20,B
00020000 54 48 49 53 20 49 53 20   41 20 54 45 53 54 21 21    THIS IS A TEST!!
00020010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................

143-Bug>MD 21000:20,B
00021000 54 48 49 53 20 49 53 20   41 20 54 45 53 54 21 21    THIS IS A TEST!!
00021010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
```

Example 1:

```
143-Bug>BC 20000 2001F 21000
Effective address: 00020000
Effective address: 0002001F
Effective address: 00021000
143-Bug>                                  (memory compares, nothing printed)
```

Example 2:

```
143-Bug>BC 20000:20 21000;B
Effective address: 00020000
Effective count  : &32
Effective address: 00021000
143-Bug>
```

(memory compares, nothing printed)

Example 3:

```
143-Bug>MM 2100F;B
0002100F 21? 0.
143-Bug>
```

(create a mismatch)

```
143-Bug>BC 20000:20 21000;B
Effective address: 00020000
Effective count  : &32
Effective address: 00021000
0002000F: 21   0002100F: 00
143-Bug>
```

(mismatches are printed out)

Block Of Memory Fill

BF *range del data* [*increment*] [;**B**|**W**|**L**]

where:

data and *increment* are both expression parameters

options (length of data field):

B – Byte
W – Word
L – Longword

The BF command fills the specified range of memory with a data pattern. If an increment is specified, then *data* is incremented by this value following each write, otherwise *data* remains a constant value. A decrementing pattern may be accomplished by entering a negative increment. The data entered by the user is right–justified in either a byte, word, or longword field (as specified by the option selected). The default field length is W (word).

If the user–entered data does not fit into the data field size, then leading bits are truncated to make it fit. If truncation occurs, then a message is printed stating the data pattern which was actually written (or initially written if an increment was specified).

If the user–entered increment does not fit into the data field size, then leading bits are truncated to make it fit. If truncation occurs, then a message is printed stating the increment which was actually used.

If the upper address of the range is not on the correct boundary for an integer multiple of the data to be stored, then data is stored to the last boundary before the upper address. No address outside of the specified range is ever disturbed in any case. The "Effective address" messages displayed by the command show exactly where data was stored.

Example 1: (Assume memory from $20000 through $2002F is clear.)

```
143-Bug>BF 20000,2001F 4E71
Effective address: 00020000
Effective address: 0002001F
143-Bug>MD 20000:30;B
00020000 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71   NqNqNqNqNqNqNqNq
00020010 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71   NqNqNqNqNqNqNqNq
00020020 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
```

Because no option was specified, the length of the data field defaulted to word.

Example 2: (Assume memory from $20000 through $2002F is clear.)

```
143-Bug>BF 20000:10 4E71 ;B
Effective address: 00020000
Effective count  : &16
Data = $71
143-Bug>MD 20000:30;B
00020000 71 71 71 71 71 71 71 71   71 71 71 71 71 71 71 71   qqqqqqqqqqqqqqqq
00020010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00020020 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
```

The specified data did not fit into the specified data field size. The data was truncated and the "Data = " message was output.

Example 3: (Assume memory from $20000 through $2002F is clear.)

```
143-Bug>BF 20000,20006 12345678 ;L
Effective address: 00020000
Effective address: 00020003
143-Bug>MD 20000:30;B
00020000 12 34 56 78 00 00 00 00   00 00 00 00 00 00 00 00   .4Vx............
00020010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00020020 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
```

The longword pattern would not fit evenly in the given range. Only one longword was written and the "Effective address" messages reflect the fact that data was not written all the way up to the specified address.

Example 4: (Assume memory from $20000 through $2002F is clear.)

```
143-Bug>BF 20000:18 0 1                                    (default size is word)
Effective address: 00020000
Effective count  : &24
143-Bug>MD 20000:18
00020000 0000 0001 0002 0003   0004 0005 0006 0007    ................
00020010 0008 0009 000A 000B   000C 000D 000E 000F    ................
00020020 0010 0011 0012 0013   0014 0015 0016 0017    ................
```

Bootstrap Operating System And Halt

BH

BH [*controller LUN*][*del device LUN*][*del string*]

where:

controller LUN – is the LUN of the controller to which the above device is attached. Defaults to LUN 0.

device LUN – is the LUN of the device to boot from. Defaults to LUN 0.

del – is a field delimiter: comma (,) or spaces ( ).

string – is a string that is passed to the operating system or control program loaded. Its syntax and use is completely defined by the loaded program.

BH is used to load an operating system or control program from disk into memory. This command works in exactly the same way as the BO command, except that control is not given to the loaded program. After the registers are initialized, control is returned to the 143Bug debugger and the prompt reappears on the terminal screen. Because control is retained by 143Bug, all the 143Bug facilities are available for debugging the loaded program if necessary.

Examples:

```
143-Bug>bh 0,1
143-Bug>
```

Boot and halt from controller 0, device LUN 1.

```
143-Bug>bh 3,a,test2;d
143-Bug>
```

Boot and halt from controller 3, device LUN $A, and pass the string "test2;d" to the loaded program.

Refer to the *BO Command* paragraph in the chapter for more detailed information about what happens during bootstrap loading.

Block Of Memory Initialize

BI *range* [;B|W|L]

options:

B – Byte
W – Word
L – Longword

The BI command may be used to initialize parity for a block of memory. The BI command is non-destructive; if the parity is correct for a memory location, then the contents of that memory location are not altered.

The limits of the block of memory to be initialized may be specified using a range. The length option is valid only when a count is entered.

BI works through the memory block by reading from locations and checking parity. If the parity is not correct, then the data read is written back to the memory location in an attempt to correct the parity. If the parity is not correct after the write, then the message "RAM FAIL" is output and the address is given.

This command may take several seconds to initialize a large block of memory.

Example 1:

```
143-Bug>BI 0 : 10000 ;B
Effective address: 00000000
Effective count  : &65536
143-Bug>
```

Example 2: (Assume system memory from $0 to $000FFFFF.)

```
143-Bug>BI 0,1FFFFF
Effective address: 00000000
Effective address: 001FFFFF
RAM FAIL AT $00100000
143-Bug>
```

Block Of Memory Move

BM

BM *range del addr* [; **B|W|L**]

3

options:

B – Byte
W – Word
L – Longword

The BM command copies the contents of the memory addresses defined by *range* to another place in memory, beginning at *addr*.

The option field is only allowed when *range* was specified using a count. In this case, the B, W, or L defines the size of data that the count is referring to. For example, a count of 4 with an option of L would mean to move 4 longwords (or 16 bytes) to the new location. If an option field is specified without a count in the range, an error results.

Example 1: (Assume memory from 20000 to 2000F is clear.)

```
143-Bug>MD 21000:20;B
00021000 54 48 49 53 20 49 53 20   41 20 54 45 53 54 21 21   THIS IS A TEST!!
00021010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................

143-Bug>BM 21000 2100F 20000
Effective address: 00021000
Effective address: 0002100F
Effective address: 00020000

143-Bug>MD 20000:20;B
00020000 54 48 49 53 20 49 53 20   41 20 54 45 53 54 21 21   THIS IS A TEST!!
00020010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
143-Bug>
```

Example 2: This utility is very useful for patching assembly code in memory. Suppose the user had a short program in memory at address 20000...

```
143-Bug>MD 20000 2000A;DI
00020000 D480                ADD.L     D0,D2
00020002 E2A2                ASR.L     D1,D2
00020004 2602                MOVE.L    D2,D3
00020006 4E4F                TRAP      #15
00020008 0021                DC.W      $21
0002000A 4E71                NOP
```

Now suppose the user would like to insert a NOP between the ADD.L instruction and the ASR.L instruction. The user should Block Move the object code down two bytes to make room for the NOP.

```
143-Bug>BM 20002 2000B 20004
Effective address: 00020002
Effective address: 0002000B
Effective address: 00020004

143-Bug>MD 20000 2000C;DI
00020000 D480                    ADD.L     D0,D2
00020002 E2A2                    ASR.L     D1,D2
00020004 E2A2                    ASR.L     D1,D2
00020006 2602                    MOVE.L    D2,D3
00020008 4E4F                    TRAP      #15
0002000A 0021                    DC.W      $21
0002000C 4E71                    NOP
```

Now the user needs simply to enter the NOP at address 20002.

```
143-Bug>MM 20002;DI
00020002 E2A2                    ASR.L     D1,D2 ? NOP
00020002 4E71                    NOP
00020004 E2A2                    ASR.L     D1,D2 ? .
143-Bug>

143-Bug>MD 20000 2000C;DI
00020000 D480                    ADD.L     D0,D2
00020002 4E71                    NOP
00020004 E2A2                    ASR.L     D1,D2
00020006 2602                    MOVE.L    D2,D3
00020008 4E4F                    TRAP      #15
0002000A 0021                    DC.W      $21
0002000C 4E71                    NOP
143-Bug>
```

Bootstrap Operating System

BO

BO [*controller LUN*][*del device LUN*][*del string*]

where:

controller LUN – is the LUN of the controller to which the above device is attached. Defaults to LUN 0.

device LUN – is the LUN of the device to boot from. Defaults to LUN 0.

del – is a field delimiter: comma (,) or spaces ( ).

string – is a string that is passed to the operating system or control program loaded. Its syntax and use is completely defined by the loaded program.

BO is used to load an operating system or control program from disk into memory and give control to it. Where to find the program and where in memory to load it is contained in block 0 of the device LUN specified. (Refer to Appendix D.) The device configuration information is located in block 1 (Appendix D). The controller and device configurations used when BO is initiated can be examined and changed via the I/O Teach (IOT) command.

The following sequence of events occurs when BO is invoked:

1. Block 0 of the controller LUN and device LUN specified is read into memory.
2. Locations $F8 (248) through $FF (255) of block 0 are checked to contain the string "MOTOROLA" or "EXORMACS".
3. The following information is extracted from block 0:

   $90 (144) – $93 (147) : Configuration area starting block.
   $94 (148) : Configuration area length in blocks.

   If any of the above two fields is zero, the present controller configuration is retained; otherwise the first block of the configuration area is read and the controller reconfigured.

4. The program is read from disk into memory. The following locations from block 0 contain the necessary information to initiate this transfer:

   $14 (20) - $17 (23) : Block number of first sector to load from disk.
   $18 (24) - $19 (25) : Number of blocks to load from disk.
   $1E (30) - $21 (33) : Starting memory location to load.

5. The first eight locations of the loaded program must contain a "pseudo reset vector", which is loaded into the target registers:

   0–3: Initial value for target system stack pointer.

   4–7: Initial value for target PC. If less than load address+8, then it represents a displacement that, when added to the starting load address, yields the initial value for the target PC.

6. Other target registers are initialized with certain arguments. The resultant target state is shown below:

   PC = Entry point of loaded program (loaded from "pseudo reset vector").
   SR = $2700.
   D0 = Device LUN.
   D1 = Controller LUN.
   D4 = Flags for IPL; 'IPLx', with x = bits

	7654 3210
Reserved	00
Firmware support for TRAP #15	1
Firmware support IPL Disk I/O	1
Firmware support for SCSI streaming tape	0
Firmware support for TRAP #15 ID Packet	1
Unused (Reserved)	00

   A0 = Address of disk controller.
   A1 = Entry point of loaded program.
   A2 = Address of media configuration block. Zero if no configuration loaded.
   A5 = Start of string (after command parameters).
   A6 = End of string + 1 (if no string was entered A5 = A6).
   A7 = Initial stack pointer (loaded from "pseudo reset vector").

7. Control is given to the loaded program. Note that the arguments passed to the target program, as for example, the string pointers, may be used or ignored by the target program.

Examples:

`143-Bug>`**BO**	Boot from default controller LUN and device LUN as defined by AB command.
`143-Bug>`**BO,3**	Boot from default controller LUN, device LUN 3.
`143-Bug>`**bo3**	Boot from controller LUN 3, default device LUN.
`143-Bug>`**bo 0,8,test**	Boot from controller LUN 0, device LUN 8, and pass the string "test" to the booted program.

Breakpoint Insert/Delete

BR
NOBR

BR [*addr*[:*count*]]
NOBR [*addr*]

The BR command allows the user to set a target code instruction address as a "breakpoint address" for debugging purposes. If, during target code execution, a breakpoint with 0 count is found, the target code state is saved in the target registers and control is returned back to 143Bug. This allows the user to see the actual state of the processor at selected instructions in the code.

Up to eight breakpoints can be defined. The breakpoints are kept in a table which is displayed each time either BR or NOBR is used. If an address is specified with the BR command, that address is added to the breakpoint table. The count field specifies how many times the instruction at the breakpoint address must be fetched before a breakpoint is taken. The count, if greater than zero, is decremented with each fetch. Every time that a breakpoint with zero count is found, a breakpoint handler routine prints the CPU state on the screen and control is returned to 143Bug.

Refer to Chapter 2 for use of a function code as part of the *addr* field.

NOBR is used for deleting breakpoints from the breakpoint table. If an address is specified, then that address is removed from the breakpoint table. If NOBR (CR) is entered, then all entries are deleted from the breakpoint table and the empty table is displayed.

Example:

```
143-Bug>BR 14000,14200  14700:&12          (Set some breakpoints.)
BREAKPOINTS
00014000              00014200
00014700:C
143-Bug>NOBR 14200                         (Delete one breakpoint.)
BREAKPOINTS
00014000              00014700:C
143-Bug>NOBR                               (Delete all breakpoints.)
BREAKPOINTS
143-Bug>
```

Block Of Memory Search

BS

BS *range del 'text'* **[;B|W|L]**

or

BS *range del data del [mask]* **[;B|W|L,N,V]**

The BS command searches the specified range of memory for a match with a user-entered data pattern. This command has three modes, as described below.

Mode 1 - LITERAL STRING SEARCH -- In this mode, a search is carried out for the ASCII equivalent of the literal string entered by the user. This mode is assumed if the single quote (') indicating the beginning of a *text* field is encountered following *range*. The size as specified in the option field tells whether the count field of *range* refers to bytes, words, or longwords. If *range* is not specified using a count, then no options are allowed. If a match is found, then the address of the first byte of the match is output.

Mode 2 - DATA SEARCH -- In this mode, a data pattern is entered by the user as part of the command line and a size is either entered by the user in the option field or is assumed (the assumption is word). The size entered in the option field also dictates whether the count field in *range* refers to bytes, words, or longwords. The following actions occur during a data search:

a. The user-entered data pattern is right-justified and leading bits are truncated or leading zeros are added as necessary to make the data pattern the specified size.

b. A compare is made with successive bytes, words, or longwords (depending on the size in effect) within the range for a match with the user-entered data. Comparison is made only on those bits at bit positions corresponding to a "1" in the mask. If no mask is specified, then a default mask of all ones is used (all bits are compared). The size of the mask is taken to be the same size as the data.

c. If the **N** (non-aligned) option has been selected, then the data is searched for on a byte-by-byte basis, rather than by words or longwords, regardless of the size of *data*. This is useful if a word (or longword) pattern is being searched for, but is not expected to lie on a word (or longword) boundary.

d. If a match is found, then the address of the first byte of the match is output along with the memory contents. If a mask was in use, then the actual data at the memory location is displayed, rather than the data with the mask applied.

Mode 3 - DATA VERIFICATION -- If the **V** (verify) option has been selected, then displaying of addresses and data is done only when the memory contents do NOT match the user-specified pattern. Otherwise this mode is identical to Mode 2.

For all three modes, information on matches is output to the screen in a four-column format. If more than 24 lines of matches are found, then output is inhibited to prevent the first match from rolling off the screen. A message is printed at the bottom of the screen indicating that there is more to display. To resume output, the user should simply press any character key. To cancel the output and exit the command, the user should press the BREAK key.

If a match is found (or, in the case of Mode 3, a mismatch) with a series of bytes of memory whose beginning is within the range but whose end is outside of the range, then that match is output and a message is output stating that the last match does not lie entirely within the range. The user may search non-contiguous memory with this command without causing a Bus Error.

Examples: (Assume the following data is in memory.)

```
00030000 0000 0045 7272 6F72   2053 7461 7475 733D   ...Error Status=
00030010 3446 2F2F 436F 6E66   6967 5461 626C 6553   4F//ConfigTableS
00030020 7461 7274 3A00 0000   0000 0000 0000 0000   tart:...........
```

```
143-Bug>BS 30000 3002F 'Task Status'
Effective address: 00030000

Effective address: 0003002F
-not found-
```

Mode 1: the string is not found, so a message is output.

```
143-Bug>BS 30000 3002F 'Error Status'
Effective address: 00030000
Effective address: 0003002F
00030003
```

Mode 1: the string is found, and the address of its first byte is output.

```
143-Bug>BS 30000 3001F 'ConfigTableStart'
Effective address: 00030000
Effective address: 0003001F
00030014
-last match extends over range boundary-
```

Mode 1: the string is found, but it ends outside of the range, so the address of its first byte and a message are output.

```
143-Bug>BS 30000:30 't' ;B
Effective address: 00030000
Effective count  : &48
0003000A  0003000C  00030020  00030023
```

Mode 1, using *range* with count and size option: count is displayed in decimal, and address of each occurrence of the string is output.

```
143-Bug>BS 30000:18,2F2F
Effective address: 00030000
Effective count  : &24
00030012|2F2F
```

Mode 2, using *range* with count: count is displayed in decimal, and the data pattern is found and displayed.

```
143-Bug>bs 30000,3002F 3d34
Effective address: 00030000
Effective address: 0003002F
-not found-
```

Mode 2: the default size is word and the data pattern is not found, so a message is output.

```
143-Bug>bs 30000,3002F 3d34 ;n
Effective address: 00030000
Effective address: 0003002F
0003000F|3D34
```

Mode 2: the default size is word and non-aligned option is used, so the data pattern is found and displayed.

```
143-Bug>BS 30000:30 60,F0 ;B
Effective address: 00030000
Effective count  : &48
00030006|6F  0003000B|61  00030015|6F  00030016|6E

00030017|66  00030018|69  00030019|67  0003001B|61

0003001C|62  0003001D|6C  0003001E|65  00030021|61
```

Mode 2, using *range* with count, mask option, and size option: count is displayed in decimal, and the actual unmasked data patterns found are displayed.

```
143-Bug>BS 3000 1FFFF 0000 000F;V
Effective address: 00003000
Effective address: 0001FFFE
0000C000|E501   0001E224|A30E
143-Bug>
```

Mode 3, on a different block of memory, mask option, scan for words with low nibble nonzero: two locations failed to verify.

Block Of Memory Verify

BV

BV *range del data* [*increment*] [;**B**|**W**|**L**]

where:

data and *increment* are both expression parameters

options:

B – Byte
W – Word
L – Longword

The BV command compares the specified range of memory against a data pattern. If an increment is specified, then *data* is incremented by this value following each comparison, otherwise *data* remains a constant value. A decrementing pattern may be accomplished by entering a negative increment. The data entered by the user is right-justified in either a byte, word, or longword field (as specified by the option selected). The default field length is W (word).

If the user-entered data or increment (if specified) do not fit into the data field size, then leading bits are truncated to make them fit. If truncation occurs, then a message is printed stating the data pattern and, if applicable, the increment value actually used.

If the range is specified using a count, then the count is assumed to be in terms of the data size.

If the upper address of the range is not on the correct boundary for an integer multiple of the data to be stored, then data is stored to the last boundary before the upper address. No address outside of the specified range is read from in any case. The "Effective address" messages displayed by the command show exactly the extent of the area read from.

Example 1: (Assume memory from $20000 to $2002F is as indicated.)

```
143-Bug>MD 20000:30;B
00020000 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71    NqNqNqNqNqNqNqNq
00020010 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71    NqNqNqNqNqNqNqNq
00020020 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71    NqNqNqNqNqNqNqNq
143-Bug>BV 20000 2001F 4E71                  (default size is word)
Effective address: 00020000
Effective address: 0002001F
143-Bug>                                     (verify successful, nothing printed)
```

Example 2: (Assume memory from $20000 to $2002F is as indicated.)

```
143-Bug>MD 20000:30;B
00020000 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
00020010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
00020020 00 00 00 00 00 00 00 00   00 00 4A FB 4A FB 4A FB    ..........J{J{J{
143-Bug>BV 20000:30 0;B
Effective address: 00020000
Effective count  : &48
0002002A|4A  0002002B|FB  0002002C|4A  0002002D|FB          (mismatches are
0002002E|4A  0002002F|FB                                     printed out)
143-Bug>
```

Example 3: (Assume memory from $20000 to $2002F is as indicated.)

```
143-Bug>MD 20000:18
00020000 0000 0001 0002 0003   0004 0005 0006 0007    ................
00020010 0008 FFFF 000A 000B   000C 000D 000E 000F    ................
00020020 0010 0011 0012 0013   0014 0015 0016 0017    ................
143-Bug>BV 20000:18 0 1                      (default size is word)
Effective address: 00020000
Effective count  : &24
00020012|FFFF                                (mismatches are printed out)
143-Bug>
```

Checksum

CS

CS *address1 address2*

The CS command provides access to the same checksum routine used by the power-up self-test firmware. This routine is used in two ways within the firmware monitor.

a. At power-up, the power-up confidence test is executed. One of the items verified is the checksum contained in the firmware monitor EPROM. If for any reason the contents of the EPROM were to change from the factory version, the checksum test is designed to detect the change and inform the user of the failure.

b. Following a valid power-up test, 143Bug examines the ROM map space for code that needs to be executed. This feature (ROMboot) makes use of the checksum routine to verify that a routine in memory is really there to be executed at power-up. For more information, refer to the *ROMboot* paragraph in Chapter 1 which describes the format of the routine to be executed and the interface provided upon entry.

This command is provided as an aid in preparing routines for the ROMboot feature. Because ROMboot does checksum validation as part of its screening process, the user needs access to the same routine in the preparation of EPROM/ROM routines.

The *address* parameters can be provided in two forms:

a. An absolute address (32-bit maximum).

b. An expression using a displacement + relative offset register.

When the CS command is used to calculate/verify the content and location of the new checksum, the operands need to be entered. The even and odd byte result should be 0000, verifying that the checksum bytes were calculated correctly and placed in the proper locations.

The algorithm used to calculate the checksum is as follows:

a. $FF is placed in each of two bytes within a register. These bytes represent the even and odd bytes as the checksum is calculated.

b. Starting with *address1* the even and odd bytes are extracted from memory and XORed with the bytes in the register.

c. This process is repeated, word by word, until *address2* is reached. This technique allows use of even ending addresses ($D40000 as opposed to $D3FFFF).

Examples:

```
143-Bug>MD 20000:3F;B
```

Display routine requiring a checksum. Start at $20000; last byte is at $20027. Checksum will be placed in bytes at $20026 and $20027, so they are zero while calculating the checksum.

```
00020000  42 4F 4F 54 00 00 00 14  00 00 00 A6 54 65 73 74   BOOT........&Test
00020010  41 F9 00 01 F0 00 20 3C  00 00 EF FF 11 00 51 C8   Ay..p. <..o...QH
00020020  FF FC 4E 75 01 01 00 00  FF FF FF FF FF FF FF FF   .|Nu............
00020030  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF   ................
```

```
143-Bug>M 20010;DI
```

Display executable code plus revision number, checksum, socket ID, and a few unused bytes following the routine.

```
00020010   41F90001 F000      LEA.L   ($1F000).L,A0 ?(CR)
00020016   203C0000 EFFF      MOVE.L  #$EFFF,D0 ?(CR)
0002001C   1100               MOVE.B  D0,-(A0) ?(CR)
0002001E   51C8FFFC           DBF.W   D0,$2001C ?(CR)
00020022   4E75               RTS ?   (CR)
00020024   0101               BTST.L  D0,D1 ?(CR)
                                       0101 is revision.

00020026   0000               DC.W    $0 ?(CR)
                                       0000 is where checksum is to be placed.

00020028   FFFF               DC.W    $FFFF ?(CR)
                                       FFFF is unused memory.

0002002A   FFFF               DC.W    $FFFF ?(CR)
                                       FFFF is unused memory.

0002002C   FFFF               DC.W    $FFFF ?(CR)
                                       FFFF is unused memory.

0002002E   FFFF               DC.W    $FFFF ?(CR)
                                       FFFF is unused memory.

00020030   FFFF               DC.W    $FFFF ?.
                                       FFFF is unused memory.
```

EXAMPLE (Using Absolute Addresses)	COMMENT
143-Bug>**CS 20000 20028**	Request checksum of area using absolute addresses.
Effective address: 00020000 Effective address: 00020027 Even/Odd = $5B3C	Checksum of even bytes is $5B. Checksum of odd bytes is $3C.
143-Bug>**M 20026;W**	Place these bytes in zeroed area used while calculating checksum.
00020026 0000 ?**5B3C.**	
143-Bug>**CS 20000 20028**	Verify checksum.
Effective address: 00020000 Effective address: 00020027 Even/Odd = $0000	Result is 0000, good checksum.

EXAMPLE (Using Relative Offset)	COMMENT
143-Bug>**OF R3** R3 =00000000 00000000? **20000** .	Define value of relative offset register 3.
143-Bug>**CS 0+R3 28+R3**	Request checksum of area using relative offset.
Effective address: 00000+R3 Effective address: 00027+R3 Even/Odd = $5B3C	Checksum of even bytes is $5B. Checksum of odd bytes is $3C.
143-Bug>**M 26+R3;W**	Place these bytes in zeroed area used while checksum was calculated.
00000026+R3 0000 ?**5B3C.**	
143-Bug>**CS 0+R3 28+R3** Effective address: 00000+R3 Effective address: 00027+R3 Even/Odd = $0000	Verify checksum.
143-Bug>	

Data Conversion

DC

DC *exp | addr*

3

The DC command is used to simplify an expression into a single numeric value. This equivalent value is displayed in its hexadecimal and decimal representation. If the numeric value could be interpreted as a signed negative number (i.e., if the most significant bit of the 32-bit internal representation of the number is set), then both the signed and unsigned interpretations are displayed.

DC can also be used to obtain the equivalent effective address of an MC68030 addressing mode.

Examples:

```
143-Bug>DC 10
          00000010 = $10 = &16

143-Bug>DC &10-&20
SIGNED   : FFFFFFF6 = -$A = -&10
UNSIGNED: FFFFFFF6 = $FFFFFFF6 = &4294967286

143-Bug>DC 123+&345+@67+%1100001
          00000314 = $314 = &788

143-Bug>DC (2*3*8) /4
          0000000C = $C = &12

143-Bug>DC 55&F
          00000005 = $5 = &5

143-Bug>DC 55>>1
          0000002A = $2A = &42
```

The subsequent examples assume A0=00030000 and the following data resides in memory:

```
00030000 11111111   22222222   33333333   44444444          ....""""3333DDDD

143-Bug>DC (A0)
          00030000 = $30000 = &196608

143-Bug>DC ([A0])
          11111111 = $11111111 = &286331153

143-Bug>DC (4,A0)
          00030004 = $30004 = &196612

143-Bug>DC ([4,A0])
          22222222 = $22222222 = &572662306
```

Dump S-Records

DU

DU [*port*]*del range del*[*text del*][*addr*][*offset*][;B|W|L]

The DU command outputs data from memory in the form of Motorola S-records to a port specified by the user. If port is not specified, then the S-records are sent to the default host port (port 1).

The option field is allowed only if a count was entered as part of the range, and defines the units of the count (bytes, words, or longwords).

The optional *text* field is for text that will be incorporated into the header (S0) record of the block of records that will be dumped.

The optional *addr* field is to allow the user to enter an entry address for code contained in the block of records. This address is incorporated into the address field of the block termination record. If no entry address is entered, then the address field of the termination record will consist of zeros. The termination record will be an S7, S8, or S9 record, depending on the address entered. Refer to Appendix C for additional information on S-records.

An optional offset may also be specified by the user in the *offset* field. The offset value is added to the addresses of the memory locations being dumped, to come up with the address which is written to the address field of the S-records. This allows the user to create an S-record file which will load back into memory at a different location than the location from which it was dumped. The default offset is zero.

NOTE

If an offset is to be specified but no entry address is to be specified, then two commas (indicating a missing field) must precede the offset to keep it from being interpreted as an entry address.

Example 1: Dump memory from $20000 to $2002F to port 1.

```
143-Bug>DU 20000 2002F
Effective address: 00020000
Effective address: 0002002F
143-Bug>
```

Example 2: Dump 10 bytes of memory beginning at $30000 to the terminal screen (port 0).

```
143-Bug>DU 0 30000:&10
Effective address: 00030000
Effective count  : &10
S0030000FC
S20E0300002602544553546608 4E4F7B
S9030000FC
```

Example 3: Dump memory from $20000 to $2002F to host (port 1). Specify a filename of "TEST" in the header record and specify an entry point of $2000A.

```
143-Bug>DU 20000 2002F 'TEST' 2000A
Effective address: 00020000
Effective address: 0002002F
143-Bug>
```

The following example shows how to upload S-records to a host computer (in this case a system running the VERSAdos operating system), storing them in the file "FILE1.MX" which the user creates with the VERSAdos utility UPLOADS.

`143-Bug>TM` `Escape character: $01=^A`	(Go into transparent mode to establish communication with the system.)
BREAK	(Press BREAK key to get VERSAdos login prompt.)
" **login** " "	(User must log onto VERSAdos and enter the catalog where FILE1.MX will reside.)
=UPLOADS FILE1	(At VERSAdos prompt, invoke the UPLOADS utility and tell it to create a file named "FILE1" for the S-records that will be uploaded.)

The UPLOADS utility at this point displays some messages like the following:

```
                    UPLOAD "S" RECORDS
                       Version x.y
                Copyrighted by MOTOROLA, INC.

volume=xxxx
 catlg=xxxx
  file=FILE1
   ext=MX

UPLOADS Allocating new file

Ready for "S" records,...
```

```
=^A                                  (When the VERSAdos prompt returns, enter
143-Bug>                             the escape character to return to 143Bug).
```

Now enter the command for 143Bug to dump the S-records to the port.

```
143-Bug>DU 20000 2000F 'FILE1'
Effective address: 00020000
Effective address: 0002000F
143-Bug>
```

```
143-Bug>TM                           (Go into transparent mode again.)
Escape character: $01=^A
```

```
QUIT                                 (Tell UPLOADS to quit looking for records.)
```

The UPLOADS utility now displays some more messages like this:

```
                    UPLOAD "S" RECORDS
                       Version x.y
                Copyrighted by MOTOROLA, INC.

volume=xxxx
 catlg=xxxx
  file=FILE1
   ext=MX

*STATUS*  No error since start of program

Upload of S-Records complete.
```

```
=OFF
```

(The VERSAdos prompt should return. Log off of the system.)

```
^A
143-Bug
```

(Enter the escape character to return to 143Bug.)

EEPROM Programming

EEP

EEP *range del addr* [;W]

options:

W - Word (default)

The EEP command is similar to the BM command in that it copies the contents of the memory addresses defined by *range* to EEPROM or another place in memory, beginning at *addr*. However, the EEP command moves the data a word at a time with a 15 millisecond delay between each data move. Also, *addr* must be a word-aligned address.

Example 1: (Assumes EEPROMs installed in XU21 and XU28 (bank 2), and J6 configured for the right size EEPROMs. Refer to the *MVME143 MPU VMEmodule User's Manual* for jumper details. XU21 and XU28 are at addresses starting at $FFF20000 and ending at or below $FFF3FFFF in the main memory map, with the odd-byte chip in XU21 and the even-byte chip in XU28. Note that 143Bug is in the EPROMs in XU3 and XU12 (bank 1), at $FFF00000 through $FFF1FFFF, with odd bytes in XU3 and even bytes in XU12.)

```
143-Bug>MD 21000:20;B
00021000 54 48 49 53 20 49 53 20   41 20 54 45 53 54 21 21    THIS IS A TEST!!
00021010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................

143-Bug>EEP 21000 2101F FFA00000
Effective address: 00021000
Effective address: 0002101F
Effective address: FFA00000
Programming EEPROM - Done.

143-Bug>MD F20000:10;W
00F20000 54 48 49 53 20 49 53 20   41 20 54 45 53 54 21 21    THIS IS A TEST!!
00F20010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
143-Bug>
```

Example 2:

```
143-Bug>EEP 21000:8 F20000;W
Effective address: 00021000
Effective count   : &8
Effective address: 00F20000
Programming EEPROM - Done.

143-Bug>MD F20000:10;W
00F20000 54 48 49 53 20 49 53 20   41 20 54 45 53 54 21 21    THIS IS A TEST!!
00F20010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
143-Bug>
```

Set Environment To Bug/Operating System

ENV

The ENV command allows the user to select the environment that the Bug is to execute in. When specified, the Bug remains in that environment until the ENV command is invoked again to change it. The selections are saved in BBRAM and used whenever power is lost.

Two Bug modes are available:

Bug	This is the standard mode of operation, and is the one defaulted to if BBRAM should fail.
System	This is the mode for system operation and is defined in Appendix A.

Two operating system modes are available:

SYSTEM V/68	This is the standard system mode, and is the one defaulted to if BBRAM should fail. In this mode the MVME143 disk controller default configurations are for 512b sectors.
VERSAdos	In this mode, the MVME143 disk controller default configurations are for 256b sectors.

Example 1:

```
143-Bug>env
Bug or System environment [B,S] = S? (CR)                    (no change)
SYSTEM V/68 or VERSAdos operating system [S,V] = S? v        (change to VERSAdos
                                                              operating system)

143-Bug>
```

Example 2:

```
143-Bug>ENV
Bug or System environment [B,S] = B? S                    (change to system
                                                           mode of operation)
SYSTEM V/68 or VERSAdos operating system [S,V] = V? s     (change to SYSTEM
                                                           V/68 operating system)
```

Firmware now takes the reset path and initializes the MVME143 for the system mode (refer to Appendix A for system mode operation details).

Example 3:

```
143-Bug>ENV
Bug or System environment [B,S] = S? B                    (change to Bug mode)

SYSTEM V/68 or VERSAdos operating system [S,V] = S? V     (change to VERSAdos
                                                           operating system)

Copyright Motorola Inc. 1988, All Rights Reserved

VME143 Monitor/Debugger Release 1.0 - 4/8/88

FPC passed test
MMU passed test

Cold Start
143-Bug>
```

Go Direct (Ignore Breakpoints)

GD

GD [*addr*]

3

GD command is used to start target code execution. If an address is specified, it is placed in the target PC. Execution starts at the target PC address. As opposed to GO, breakpoints are not inserted.

Refer to Chapter 2 for use of a function code as part of the *addr* field.

Once execution of the target code has begun, control may be returned to 143Bug by various conditions:

a. User pressed the ABORT or RESET switches on the MVME143 front panel.

b. An unexpected exception occurred.

c. By execution of the .RETURN TRAP #15 function.

Example: (The following program resides at $10000.)

```
143-Bug>MD 10000;DI
00010000 2200                    MOVE.L  D0,D1
00010002 4282                    CLR.L   D2
00010004 D401                    ADD.B   D1,D2
00010006 E289                    LSR.L   #$1,D1
00010008 66FA                    BNE.B   $10004
0001000A E20A                    LSR.B   #$1,D2
0001000C 55C2                    SCS.B   D2
0001000E 60FE                    BRA.B   $1000E
143-Bug>RM D0
```

Initialize D0 and start target program:

```
D0  =00000000 ? 52A9C.
143-Bug>GD 10000
Effective address: 00010000
```

To exit target code, press ABORT switch.

```
Exception: Abort
Format Vector = 007C
PC  =0001000E  SR  =2711=TR:OFF_S._7_X...C
USP =0000F830  MSP =0000FC18  ISP*=00010000 VBR =00000000
SFC =0=XX      DFC =0=XX      CACR=0=..     CAAR=00000000
D0  =00052A9C  D1  =00000000  D2  =000000FF D3  =00000000
D4  =00000000  D5  =00000000  D6  =00000000 D7  =00000000
A0  =00000000  A1  =00000000  A2  =00000000 A3  =00000000
A4  =00000000  A5  =00000000  A6  =00000000 A7  =00010000
0001000E 60FE                          BRA.B   $1000E
143-Bug>
```

Set PC to start of program and restart target code:

```
143-Bug>RM PC
PC  =0001000E ? 10000.
143-Bug>GD
Effective address: 00010000
```

Go To Next Instruction

GN

GN command sets a temporary breakpoint at the address of the next instruction, that is, the one following the current instruction, and then starts target code execution. After setting the temporary breakpoint, the sequence of events is similar to that of the GO command.

Refer to Chapter 2 for use of a function code as part of the *addr* field.

GN is especially helpful when debugging modular code because it allows the user to "trace" through a subroutine call as if it were a single instruction.

Example: The following section of code resides at address $6000.

```
143-Bug>MD 6000:4;DI
00006000 7003                    MOVEQ.L #$3,D0
00006002 7201                    MOVEQ.L #$1,D1
00006004 61000FFA                BSR.W   $7000
00006008 2600                    MOVE.L  D0,D3
143-Bug>
```

The following simple routine resides at address $7000.

```
143-Bug>MD 7000:2;DI
00007000 D081                    ADD.L   D1,D0
00007002 4E75                    RTS
143-Bug>
```

Execute up to the BSR instruction.

```
143-Bug>RM PC
PC  =00000000 ? 6000.
```

```
143-Bug>GT 6004
Effective address: 00006004
Effective address: 00006000
At Breakpoint
PC  =00006004  SR  =2700=TR:OFF_S._7_.....
USP =00003830  MSP =00003C18  ISP*=00004000 VBR =00000000
SFC =0=XX      DFC =0=XX      CACR=0=..     CAAR=00000000
D0  =00000003  D1  =00000001  D2  =00000000 D3  =00000000
D4  =00000000  D5  =00000000  D6  =00000000 D7  =00000000
A0  =00000000  A1  =00000000  A2  =00000000 A3  =00000000
A4  =00000000  A5  =00000000  A6  =00000000 A7  =00004000
00006004 61000FFA          BSR.W       $7000
143-Bug>
```

Use the GN command to "trace" through the subroutine call and display the results.

```
143-Bug>GN
Effective address: 00006004
At Breakpoint
PC  =00006008  SR  =2700=TR:OFF_S._7_.....
USP =00003830  MSP =00003C18  ISP*=00004000 VBR =00000000
SFC =0=XX      DFC =0=XX      CACR=0=..     CAAR=00000000
D0  =00000004  D1  =00000001  D2  =00000000 D3  =00000000
D4  =00000000  D5  =00000000  D6  =00000000 D7  =00000000
A0  =00000000  A1  =00000000  A2  =00000000 A3  =00000000
A4  =00000000  A5  =00000000  A6  =00000000 A7  =00004000
00006008 2600              MOVE.L      D0,D3
143-Bug>
```

Go Execute User Program

GO

GO [*addr*]

3

The GO command (alternate form "G") is used to initiate target code execution. All previously set breakpoints are enabled. If an address is specified, it is placed in the target PC. Execution starts at the target PC address. Refer to Chapter 2 for use of a function code as part of the *addr* field.

The sequence of events is as follows:

a. First, if an address is specified, it is loaded in the target PC.

b. Then, if a breakpoint is set at the target PC address, the instruction at the target PC is traced (executed in trace mode).

c. Next, all breakpoints are inserted in the target code.

d. Finally, target code execution resumes at the target PC address.

At this point control may be returned to 143Bug by various conditions:

a. A breakpoint with 0 count was found.

b. User pressed the ABORT or RESET switches on the MVME143 front panel.

c. An unexpected exception occurred.

d. By execution of the .RETURN TRAP #15 function.

Example: (The following program resides at $10000.)

```
143-Bug>MD 10000;DI
00010000 2200                MOVE.L  D0,D1
00010002 4282                CLR.L   D2
00010004 D401                ADD.B   D1,D2
00010006 E289                LSR.L   #$1,D1
00010008 66FA                BNE.B   $10004
0001000A E20A                LSR.B   #$1,D2
0001000C 55C2                SCS.B   D2
0001000E 60FE                BRA.B   $1000E
143-Bug>RM D0
```

Initialize D0, set some breakpoints, and start target program:

```
D0  =00000000 ? 52A9C.
143-Bug>BR 10000,1000E
BREAKPOINTS
00010000              0001000E

143-Bug>GO 10000
Effective address: 00010000
At Breakpoint
PC  =0001000E  SR  =2011=TR:OFF_S._0_X...C
USP =0000F830  MSP =0000FC18  ISP*=00010000 VBR =00000000
SFC =0=XX      DFC =0=XX      CACR=0=..     CAAR=00000000
D0  =00052A9C  D1  =00000000  D2  =000000FF D3  =00000000
D4  =00000000  D5  =00000000  D6  =00000000 D7  =00000000
A0  =00000000  A1  =00000000  A2  =00000000 A3  =00000000
A4  =00000000  A5  =00000000  A6  =00000000 A7  =00010000
0001000E 60FE              BRA.B       $1000E
```

Note that in this case breakpoints are inserted after tracing the first instruction, therefore the first breakpoint is not taken.

Continue target program execution.

```
143-Bug>G
Effective address: 0001000E
At Breakpoint
PC  =0001000E  SR  =2011=TR:OFF_S._0_X...C
USP =0000F830  MSP =0000FC18  ISP*=00010000 VBR =00000000
SFC =0=XX      DFC =0=XX      CACR=0=..     CAAR=00000000
D0  =00052A9C  D1  =00000000  D2  =000000FF D3  =00000000
D4  =00000000  D5  =00000000  D6  =00000000 D7  =00000000
A0  =00000000  A1  =00000000  A2  =00000000 A3  =00000000
A4  =00000000  A5  =00000000  A6  =00000000 A7  =00010000
0001000E 60FE              BRA.B       $1000E
```

Remove breakpoints and restart target code.

```
143-Bug>NOBR
BREAKPOINTS
143-Bug>GO 10000
Effective address: 00010000
```

To exit target code, press the ABORT switch.


```
Exception: Abort
Format Vector = 007C
PC  =0001000E  SR  =2011=TR:OFF_S._0_X...C
USP =0000F830  MSP =0000FC18  ISP*=00010000 VBR =00000000
SFC =0=XX      DFC =0=XX      CACR=0=..     CAAR=00000000
D0  =00052A9C  D1  =00000000  D2  =000000FF D3  =00000000
D4  =00000000  D5  =00000000  D6  =00000000 D7  =00000000
A0  =00000000  A1  =00000000  A2  =00000000 A3  =00000000
A4  =00000000  A5  =00000000  A6  =00000000 A7  =00010000
0001000E 60FE               BRA.B       $1000E
```

Go To Temporary Breakpoint

GT

GT *addr*

GT command allows the user to set a temporary breakpoint and then start target code execution. A count may be specified with the temporary breakpoint. Control is given at the target PC address. All previously set breakpoints are enabled. The temporary breakpoint is removed when any breakpoint with 0 count is encountered.

Refer to Chapter 2 for use of a function code as part of the *addr* field.

After setting the temporary breakpoint, the sequence of events is similar to that of the GO command. At this point control may be returned to 143Bug by various conditions:

a. A breakpoint with count 0 was found.

b. User pressed the ABORT or RESET switches on the MVME143 front panel.

c. An unexpected exception occurred.

d. By execution of the .RETURN TRAP #15 function.

Example: (The following program resides at $10000.)

```
143-Bug>MD 10000;DI
00010000 2200                    MOVE.L  D0,D1
00010002 4282                    CLR.L   D2
00010004 D401                    ADD.B   D1,D2
00010006 E289                    LSR.L   #$1,D1
00010008 66FA                    BNE.B   $10004
0001000A E20A                    LSR.B   #$1,D2
0001000C 55C2                    SCS.B   D2
0001000E 60FE                    BRA.B   $1000E
143-Bug>RM D0
```

Initialize D0 and set a breakpoint:

```
D0  =00000000 ? 52A9C.

143-Bug>BR 1000E
BREAKPOINTS
0001000E
143-Bug>
```

Set PC to start of program, set temporary breakpoint, and start target code:

```
143-Bug>RM PC
PC  =0001000E ? 10000.
143-Bug>

143-Bug>GT 10006
Effective address: 00010006
Effective address: 00010000
At Breakpoint
PC  =00010006  SR  =2711=TR:OFF_S._7_X...C
USP =00003830  MSP =00003C18  ISP*=00004000 VBR =00000000
SFC =0=XX      DFC =0=XX      CACR=0=..     CAAR=00000000
D0  =00052A9C  D1  =00000029  D2  =00000009 D3  =00000000
D4  =00000000  D5  =00000000  D6  =00000000 D7  =00000000
A0  =00000000  A1  =00000000  A2  =00000000 A3  =00000000
A4  =00000000  A5  =00000000  A6  =00000000 A7  =00004000
00010006 E289              LSR.L       #$1,D1
143-Bug>
```

Set another temporary breakpoint at $10002 and continue the target program execution.

```
143-Bug>GT 10002
Effective address: 00010006
At Breakpoint
PC  =0001000E  SR  =2711=TR:OFF_S._7_X...C
USP =00003830  MSP =00003C18  ISP*=00004000 VBR =00000000
SFC =0=XX      DFC =0=XX      CACR=0=..     CAAR=00000000
D0  =00052A9C  D1  =00000000  D2  =000000FF D3  =00000000
D4  =00000000  D5  =00000000  D6  =00000000 D7  =00000000
A0  =00000000  A1  =00000000  A2  =00000000 A3  =00000000
A4  =00000000  A5  =00000000  A6  =00000000 A7  =00004000
0001000E 60FE              BRA.B       $1000E
143-Bug>
```

Note that a breakpoint from the breakpoint table was encountered before the temporary breakpoint.

Help

HE

HE [*command*]

HE command is the 143Bug help facility. HE (CR) displays the command names of all available commands along with their appropriate titles. HE *command* displays only the command name and title for that particular command.

I/O Control For Disk

IOC

The IOC command allows a user to send command packets directly to a disk controller. The packet to be sent must already reside in memory and must follow the packet protocol of the particular disk controller. This packet protocol is outlined in the user's manual for the disk controller module. (Refer to Chapter 1.)

This command may be used as a debugging tool to issue commands to the disk controller to locate problems with either drives, media, or the controller itself.

When invoked, this command prompts for the controller and drive required. The default controller LUN and device LUN when IOC is invoked are those most recently specified for IOP, IOT, or a previous invocation of IOC. An address where the controller command is located is also prompted for. The same special characters used by the Memory Modify (MM) command to access a previous field (^), reopen the same location (=), or exit (.), can be used with IOC. The power-up default for the packet address is the area which is also used by the BO and IOP commands for building packets. IOC displays the command packet and, if instructed by the user, sends the packet to the disk controller, following the proper protocol required by the particular controller.

Example: Send the packet at $10000 to an MVME319 controller module configured as CLUN #0. Specify an operation to the hard disk which is at DLUN #1.

```
143-Bug>IOC
Controller LUN    =00? (CR)
Device LUN        =00? 1
Packet address    =000012BC? 10000
00010000 0219 1500 1001 0002   0100 3D00 3000 0000    .........=.0...
00010010 0000 0000 0300 0000   0000 0200 03          ................
Send Packet (Y/N)? Y
143-Bug>
```

I/O Physical (Direct Disk Access)

IOP

The IOP command allows the user to read, write, or format any of the supported disk or tape devices. When invoked, this command goes into an interactive mode, prompting the user for all the parameters necessary to carry out the command. The user may change the displayed value by typing a new value followed by a carriage return (CR); or may simply enter (CR), which leaves the field unchanged.

The same special characters used by the Memory Modify (MM) command to access a previous field (^), reopen the same location (=), or exit (.), can be used with IOP. After IOP has prompted the user for the last parameter, the selected function is executed. The disk SYSCALL functions (trap routines), as described in Chapter 5, are used by IOP to access the specified disk or tape.

Initially (after a cold reset), all the parameters used by IOP are set to certain default values. However, any new values entered are saved and are displayed the next time that the IOP command is invoked.

The information that the user is prompted for is as follows:

a. Controller LUN =00?

   The Logical Unit Number (LUN) of the controller to access is specified in this field.

b. Device LUN =00?

   The LUN of the device to access is specified in this field.

c. Read/Write/Format =R?

   In this field the user specifies the desired function by entering a one-character mnemonic as follows:

   1. R for read. This reads blocks of data from the selected device into memory.
   2. W for write. This writes blocks of data from memory to the selected device.
   3. F for format. This formats the selected device. For disk devices, either a track or the whole disk can be selected by a subsequent field. For tape devices, either Retension or Erase can be selected by a subsequent field.

d. Memory Address =00003000?

This field selects the starting address for the block to be accessed. For disk read operations, data is written starting at this location. For disk write operations, data is read starting at this location.

e. Starting Block =00000000?

This parameter specifies the starting disk block number to access. For disk read operations, data is read starting at this block. For disk write operations, data is written starting at this block. For disk track format operations, the track that contains this block is formatted.

f. Number of Blocks =0002?

This field specifies the number of data blocks (logical) to be transferred on a read or write operation.

g. Address Modifier =00?

This field contains the VMEbus address modifier to use for Direct Memory Access (DMA) data transfers by the selected controller. If zero is specified, a valid default value is selected by the driver. If a nonzero value is specified, then it is used by the driver for data transfers.

h. Track/Disk =T (T/D)?

This field specifies whether a disk track or the entire disk is formatted when the format operation is selected.

i. File Number =0000?

For streaming tape devices, this field specifies the starting file number to access.

j. Flag Byte =00?

The flag byte is used to specify variations of the same command, and to receive special status information. Bits 0 through 3 are used as command bits; bits 4 through 7 are used as status bits. At the present, only streaming tape devices use this field. The following bits are defined for streaming tape read and write operations.

Bit 7 File Mark flag. If 1, a file mark was detected at the end of the last operation.

The second argument would be used whenever the sequence "\1" occurred. Entering ARGUE 3000 1 ;B on the debugger command line would invoke the macro named ARGUE with the text strings 3000, 1, and ;B replacing "\0", "\1", and "\2", respectively, within the body of the macro.

To delete a macro, invoke NOMA followed by the name of the macro. Invoking NOMA without specifying a macro name deletes all macros. If NOMA is invoked with a valid macro name that does not have a definition, an error message is printed.

Examples:

```
143-Bug>MA ABC                          Define macro ABC.
M=MD 3000
M=GO \0
M= (CR)
143-Bug>

143-Bug>MA DIS                          Define macro DIS.
M=MD \0:17;DI
M= (CR)
143-Bug>

143-Bug>MA                              List macro definitions.
MACRO ABC
010 MD 3000
020 GO \0
MACRO DIS
010 MD \0:17;DI
143-Bug>

143-Bug>MA ABC                          List definitions of macro ABC.
MACRO ABC
010 MD 3000
020 GO \0
143-Bug>

143-Bug>NOMA DIS                        Delete macro DIS.
143-Bug>

143-Bug>MA ASM                          Define macro ASM.
M=MM \0;DI
M= (CR)
143-Bug>
```

MA
NOMA


```
143-Bug>MA                                  List all macros.
MACRO ABC
010 MD 3000
020 GO \0
MACRO ASM
010 MD \0;DI
143-Bug>

143-Bug>NOMA                                Delete all macros.
143-Bug>

143-Bug>MA                                  List all macros.
NO MACROS DEFINED
143-Bug>
```

characters, that the first record transferred by the host system be a header record. The header record is not used but the LF after the header record serves to break LO out of the loop so that data records are processed.

The other options have the following effects:

-C option - Ignore checksum. A checksum for the data contained within an S-record is calculated as the S-record is read in at the port. Normally, this calculated checksum is compared to the checksum contained within the S-record and if the compare fails, an error message is sent to the screen on completion of the download. If this option is selected, then the comparison is not made.

X option - Echo. This option echoes the S-records to the user's terminal as they are read in at the host port.\

T option - TRAP #15 code. This option causes LO to set the target register D4 = 'LO '*x*, with *x* = $0C ($4C4F200C). The ASCII string 'LO ' indicates that this is the LO command; the code $0C indicates TRAP #15 support with stack parameter/result passing and TRAP #15 disk support. This code can be used by the downloaded program to select the appropriate calling convention when invoking debugger functions, because some Motorola debuggers use conventions different from 143Bug, and they set a different code in D4.

The S-record format (refer to Appendix C) allows an entry point to be specified in the address field of the termination record of an S-record block. The contents of the address field of the termination record (plus the offset address, if any) are put into the target PC. Thus, after a download, the user need only enter G *addr* or GO *addr* to execute the code that was downloaded.

If a non-hex character is encountered within the data field of a data record, then the part of the record which had been received up to that time is printed to the screen and the 143Bug error handler is invoked to point to the faulty character.

As mentioned, if the embedded checksum of a record does not agree with the checksum calculated by 143Bug AND if the checksum comparison has not been disabled via the "-C" option, then an error condition exists. A message is output stating the address of the record (as obtained from the address field of the record), the calculated checksum, and the checksum read with the record. A copy of the record is also output. This is a fatal error and causes the command to abort.

When a load is in progress, each data byte is written to memory and then the contents of this memory location are compared to the data to determine if the data stored properly. If for some reason the compare fails, then a message is output stating the address where the data was to be stored, the data written, and the data read back during the compare. This is also a fatal error and causes the command to abort.

Because processing of the S-records is done character-by-character, any data that was deemed good will have already been stored to memory if the command aborts due to an error.

Examples: Suppose a host system (using VERSAdos in this case) was used to create a program that looks like this:

```
1                              *  Test Program.
2                              *
3                 65040000                ORG       $65040000
4
5       65040000 7001                      MOVEQ.L   #1,D0
6       65040002 D088                      ADD.L     A0,D0
7       65040004 4A00                      TST.B     D0
8       65040006 4E75                      RTS
9                                          END****** TOTAL ERRORS      0--
****** TOTAL WARNINGS    0--
```

Then the program was converted into an S-record file named TEST.MX as follows:

```
S00F000054455354533353372020010 15E
S30D650400007001D0884A004E75B3
S7056504000091
```

Load this file into MVME143 memory for execution at address $40000 as follows:

```
143-Bug>TM                          (Go into transparent mode to establish
Escape character: $01= ^A          communication with the host system.)

BREAK                               (Press BREAK key to get VERSAdos login
                                    prompt.)
"
login                               (User must log on onto VERSAdos and enter the
"                                   proper catalog to access the file TEST.MX)
"
=^A
                                    (143Bug prompt.)
143-Bug>LO -65000000 ;X=COPY TEXT.MX,#
COPY TEST.MX,#
S00F000054455354533353372020001015E
S30D650400007001D0884A004E75B3
S7056504000091
143-Bug>
```

The S-records are echoed to the terminal because of the "X" option.

The offset address of -65000000 was added to the addresses of the records in TEST.MX and caused the program to be loaded to memory starting at $40000. The text "COPY TEST.MX,#" is a VERSAdos command line that caused the file to be copied by VERSAdos to the host system port which is connected with the MVME143 host port.

```
143-Bug>MD 40000:4;DI
00040000 7001                MOVEQ.L #1,D0
00040002 D088                ADD.L   A0,D0
00040004 4A00                TST.B   D0
00040006 4E75                RTS
143-Bug>
```

The target PC now contains the entry point of the code in memory ($40000).

Macro Define/Display/Delete

MA
NOMA

MA [*name*]
NOMA [*name*]

The *name* can be any combination of 1–8 alphanumeric characters.

The MA command allows the user to define a complex command consisting of any number of Bug primitive commands with optional parameter specifications.

NOMA command is used to delete either a single macro or all macros.

Entering MA without specifying a macro name causes the Bug to list all currently defined macros and their definitions.

When MA is invoked with the name of a currently defined macro, that macro definition is displayed.

Line numbers are shown when displaying macro definitions to facilitate editing via the MAE command. If MA is invoked with a valid name that does not currently have a definition, then the Bug enters the macro definition mode. In response to each macro definition prompt "M=", enter a Bug command, including a carriage return. Commands entered are not checked for syntax until the macro is invoked. To exit the macro definition mode, enter only a carriage return (null line) in response to the prompt. If the macro contains errors, it can either be deleted and redefined or it can be edited with the MAE command. A macro containing no primitive Bug commands (i.e., no definition) are not accepted.

Macro definitions are stored in a string pool of fixed size. If the string pool becomes full while in the definition mode, the offending string is discarded, a message STRING POOL FULL, LAST LINE DISCARDED is printed and the user is returned to the Bug command prompt. This also happens if the string entered would cause the string pool to overflow. The string pool has a capacity of 511 characters. The only way to add or expand macros when the string pool is full is to either edit or delete macro(s).

Bug commands contained in macros may reference arguments supplied at invocation time. Arguments are denoted in macro definitions by embedding a back slash "\" followed by a numeral. Up to ten arguments are permitted. A definition containing a back slash followed by a zero would cause the first argument to that macro to be inserted in place of the "\0" characters.

Bit 1 Ignore File Number (IFN) flag. If 0, the file number field is used to position the tape before any reads or writes are done. If 1, the file number field is ignored, and reads or writes start at the present tape position.

Bit 0 End of File flag. If 0, reads or writes are done until the specified block count is exhausted. If 1, reads are done until the count is exhausted or until a file mark is found. If 1, writes are terminated with a file mark.

k. Retension/Erase =R (R/E)?

For streaming tape devices, this field indicates whether a retension of the tape or an erase should be done when a format operation is scheduled.

Retension: This rewinds the tape to BOT, advances the tape without interruptions to EOT, and then rewinds it back to BOT. Tape retension is recommended by cartridge tape suppliers before writing or reading data when a cartridge has been subjected to a change in environment or a physical shock, has been stored for a prolonged period of time or at extreme temperature, or has been previously used in a start/stop mode.

Erase: This completely clears the tape of previous data and at the same time retensions the tape.

After all the required parameters are entered, the disk access is initiated. If an error occurs, an error status word is displayed. Refer to Appendix F for an explanation of returned error status codes.

Example 1: Read 25 blocks starting at block 370 from device 2 of controller 0 into memory beginning at address $50000.

```
143-Bug>IOP
Controller LUN    =00? (CR)
Device LUN        =00? 2
Read/Write/Format=R? (CR)
Memory Address    =00030000? 50000
Starting Block    =00000000? &370
Number of Blocks =0002? &25
Address Modifier =00? (CR)
143-Bug>
```

Example 2: Write 14 blocks starting at memory location $7000 to file 6 of device 0, controller 4. Append a file mark at the end of the file.


```
143-Bug>IOP
Controller LUN    =00? 4
Device LUN        =02? 0
Read/Write/Format=R? W
Memory Address    =00050000? 7000
File Number       =00000172? 6
Number of Blocks  =0019? e
Flag Byte         =00? %01
Address Modifier  =00? (CR)
143-Bug>
```

I/O "Teach" For Configuring Disk Controller

IOT [;[H][A]]

The IOT command allows the user to "teach" a new disk configuration to 143Bug for use by the TRAP #15 disk functions. IOT lets the user modify the controller and device descriptor tables used by the TRAP #15 functions for disk access. Note that because the 143Bug commands that access the disk use the TRAP #15 disk functions, changes in the descriptor tables affect all those commands. These commands include IOP, BO, BH, and also any user program that uses the TRAP #15 disk functions.

Before attempting to access the disks with the IOP command, the user should verify the parameters and, if necessary, modify them for the specific media and drives used in the system.

Note that during a boot, the configuration sector is normally read from the disk, and the device descriptor table for the LUN used is modified accordingly. If the user desires to read/write using IOP from a disk that has been booted, IOT will not be required, unless the system is reset.

IOT may be invoked with a "H" (Help) option specified. This option instructs IOT to list the disk controllers which are currently available to the system.

Example:

```
143-Bug>iot;h

    Disk Controllers Available

 Lun    Type       Address       # dev
  0    VME320     $FFFB0000        4
143-Bug>
```

IOT may be invoked with an "A" (All) option specified. This option instructs IOT to list all the disk controllers which are currently supported in 143Bug.

When invoked without options, the IOT command enters an interactive subcommand mode where the descriptor table values currently in effect are displayed one-at-a-time on the console for the operator to examine. The operator may change the displayed value by entering a new value or may leave it unchanged by typing only a carriage return. The same special characters used by the Memory Modify (MM) command to access a previous field (^), reopen the same location (=), or exit (.), can be used with IOT. All numerical

values are interpreted as hexadecimal numbers. Decimal values may be entered by preceding the number with an "&".

The first two items of information that the user is prompted for are the Controller LUN and the Device LUN (LUN = Logical Unit Number). These two LUNs specify one particular drive out of many that may be present in the system.

If the Controller LUN and Device LUN selected do not correspond to a valid controller and device, then IOT outputs the message "Invalid LUN" and the user is prompted for the two LUNs again.

After the parameter table for one particular drive has been selected via a Controller LUN and a Device LUN, IOT begins displaying the values in the attribute fields, allowing the user to enter changes if desired.

The parameters and attributes that are associated with a particular device are determined by a parameter and an attribute mask that is a part of the device definition.

The device that has been selected may have any combination of the following parameters and attributes:

a. Sector Size:

   0–128 1–256
   2–512 3–1024 =01?

   The physical sector size specifies the number of data bytes per sector.

b. Block Size:

   0–128 1–256
   2–512 3–1024 =01?

   The block size defines the units in which a transfer count is specified when doing a disk/tape block transfer. The block size can be smaller, equal to, or greater than the physical sector size, as long as the following relationship holds true:

   (Block Size)*(Number of Blocks)/(Physical Sector Size) must be an integer.

c. Sectors/Track =0020?

   This field specifies the number of data sectors per track, and is a function of the device being accessed and the sector size specified.

d. Starting Head =10?

This field specifies the starting head number for the device. It is normally zero for Winchester and floppy drives. It is nonzero for dual-volume SMD drives.

e. Number of Heads =05?

This field specifies the number of heads on the drive.

f. Number of Cylinders =0337?

This field specifies the number of cylinders on the device. For floppy disks, the numbers of cylinders depends on the media size and the track density. General values for 5-1/4 inch floppy disks are shown below:

48 TPI - 40 cylinders
96 TPI - 80 cylinders

g. Precomp. Cylinder =0000?

This field specifies the cylinder number at which precompensation should occur for this drive. This parameter is normally specified by the drive manufacturer.

h. Reduced Write Current Cylinder =0000?

This field specifies the cylinder number at which the write current should be reduced when writing to the drive. This parameter is normally specified by the drive manufacturer.

i. Interleave Factor =00?

This field specifies how the sectors are formatted on a track. Normally, consecutive sectors in a track are numbered sequentially in increments of 1 (interleave factor of 1). The interleave factor controls the physical separation of logically sequential sectors. This physical separation gives the host time to prepare to read the next logical sector without requiring the loss of an entire disk revolution.

j. Spiral Offset =00?

The spiral offset controls the number of sectors that the first sector of each track is offset from the index pulse. This is used to reduce latency when crossing track boundaries.

k. ECC Data Burst Length =0000?

This field defines the number of bits to correct for an ECC error when supported by the disk controller.

l. Step Rate Code =00?

The step rate is an encoded field used to specify the rate at which the read/write heads can be moved when seeking a track on the disk.

The encoding is as follows:

STEP RATE CODE (HEX)	WINCHESTER HARD DISKS	5-1/4 INCH FLOPPY	8-INCH FLOPPY
00	0 msec	12 msec	6 msec
01	6 msec	6 msec	3 msec
02	10 msec	12 msec	6 msec
03	15 msec	20 msec	10 msec
04	20 msec	30 msec	15 msec

m. Single/Double DATA Density =D (S/D)?

Single (FM) or double (MFM) data density should be specified by typing S or D, respectively.

n. Single/Double TRACK Density =D (S/D)?

Used to define the density across a recording surface. This usually relates to the number of tracks per inch as follows:

48 TPI = Single Track Density
96 TPI = Double Track Density

o. Single/Equal_in_all Track zero density =S (S/E)?

This flag specifies whether the data density of track 0 is a single density or equal to the density of the remaining tracks. For the "Equal_in_all" case, the Single/Double data density flag indicates the density of track 0.

p. Slow/Fast Data Rate =S (S/F)?

This flag selects the data rate for floppy disk devices as follows:

S = 250 kHz data rate
F = 500 kHz data rate

q. Gap 1 =07?

This field contains the number of words of zeros that are written before the header field in each sector during format.

r. Gap 2 =08?

This field contains the number of words of zeros that are written between the header and data fields during format and write commands.

s. Gap 3 =00?

This field contains the number of words of zeros that are written after the data fields during format commands.

t. Gap 4 =00?

This field contains the number of words of zeros that are written after the last sector of a track and before the index pulse.

u. Spare Sectors Count =00?

This field contains the number of sectors per track allocated as spare sectors. These sectors are only used as replacements for bad sectors on the disk.

Example 1: Examining the default parameters of a 5-1/4 inch floppy disk.

```
143-Bug>IOT
Controller LUN      =00? 8
Device LUN          =00? 2
Sector Size:
0-128 1-256
2-512 3-1024        =01? (CR)
Block Size:
0-128 1-256
2-512 3-1024        =01? (CR)
Sectors/track       =0010? (CR)
Number of heads     =02? (CR)
Number of cylinders =0050? (CR)
```

```
Precomp. Cylinder     =0028? (CR)
Step Rate Code        =00? (CR)
Single/Double TRACK density=D (S/D)? (CR)
Single/Double DATA density =D (S/D)? (CR)
Single/Equal_in_all Track zero density =S (S/E)? (CR)
Slow/Fast Data Rate   =S (S/F)? (CR)
143-Bug>
```

Example 2: Changing from a 40Mb Winchester to a 70Mb Winchester. (Note that reconfiguration such as this is only necessary when a user wishes to read or write a disk which is different than the default using the IOP command. Reconfiguration is normally done automatically by the BO or BH command when booting from a disk which is different from the default.)

```
143-Bug>IOT
Controller LUN        =00? 8
Device LUN            =00? 2
Sector Size:
0-128 1-256
2-512 3-1024          =01? (CR)
Block Size:
0-128 1-256
2-512 3-1024          =01? (CR)
Sectors/track         =0020? (CR)
Starting head         =00? (CR)
Number of heads       =06? 8
Number of cylinders   =033E? 400
Precomp. Cylinder     =0000? 401
Reduced Write Current Cylinder=0000? (CR)
Interleave factor     =01? 0B
Spiral Offset       =00? (CR)
ECC Data Burst Length=0000? 000B
Reserved Area Units:Tracks/Cylinders =T (T/C)? (CR)
Tracks Reserved for Alternates=0000? (CR)
143-Bug>
```

Example 3: Changing from Fujitsu drive to Fixed/Removable CDC drive. It is necessary to reconfigure two devices, one corresponding to the fixed disk and one corresponding to the removable disk of the CDC drive.

(Fixed Disk)

```
143-Bug>IOT
Controller LUN          =00? 2
Device LUN              =00? (CR)
Sector Size:
0-128 1-256
2-512 3-1024            =02? 1
Block Size:
0-128 1-256
2-512 3-1024            =01? (CR)
Sectors/Track           =0040? (CR)
Starting Head           =00? 10
Number of Heads         =0A? 5
Number of Cylinders     =0337? (CR)
Interleave Factor       =01? (CR)
Spiral Offset           =00? (CR)
Gap 1                   =10? 7
Gap 2                   =20? 8
Spare Sectors Count     =00? (CR)
143-Bug>
```

(Removable Disk)

```
143-Bug>IOT
Controller LUN          =02? (CR)
Device LUN              =00? 1
Sector Size:
0-128 1-256
2-512 3-1024            =01? (CR)
Block Size:
0-128 1-256
2-512 3-1024            =01? (CR)
Sectors/Track           =0040? (CR)
Starting Head           =00? (CR)
Number of Heads         =00? 1
Number of Cylinders     =0337? (CR)
Interleave Factor       =01? (CR)
Spiral Offset           =00? (CR)
Gap 1                   =7? (CR)
Gap 2                   =8? (CR)
Spare Sectors Count     =00? (CR)
143-Bug>
```

Load S-Records From Host

LO

LO [*n*] [*addr*] [;X|-C|T] [=*text*]

3

The LO command is used when data in the form of a file of Motorola S-records is to be downloaded from a host system to the MVME143. The LO command accepts serial data from the host and loads it into memory.

NOTE

The highest baud rate that can be used with the LO command (downloader) is 9600 baud.

The optional port number *n* allows the user to specify which port is to be used for the downloading. If this number is omitted, port 1 is assumed.

The optional *addr* field allows the user to enter an offset address which is to be added to the address contained in the address field of each record. This causes the records to be stored to memory at different locations that would normally occur. the contents of the automatic offset register are not added to the S-record addresses. If the address is in the range $0 to $1F and the port number is omitted, enter a comma before the address to distinguish it from a port number.

The optional text field, entered after the equals sign (=), is sent to the host before 143Bug begins to look for S-records at the host port. This allows the user to send a command to the host device to initiate the download. This text should NOT be delimited by any kind of quote marks. Text is understood to begin immediately following the equals sign and terminate with the carriage return. If the host is operating full duplex, the string is also echoed back to the host port by the host and appears on the user's terminal screen.

In order to accommodate host systems that echo all received characters, the above-mentioned text string is sent to the host one character at a time and characters received from the host are read one at a time. After the entire command has been sent to the host LO keeps looking for a LF character from the host, signifying the end of the echoed command. No data records are processed until this LF is received. If the host system does not echo characters, LO still keeps looking for a LF character before data records are processed. For this reason, it is required in situations where the host system does not echo

Macro Edit

MAE

MAE *name line #* [*string*]

name	any combination of 1-8 alphanumeric characters.
line #	line number in range 1-999.
string	replacement line to be inserted.

The MAE command permits modification of the macro named on the command line. MAE is line oriented and supports the following actions: insertion, deletion, and replacement.

To insert a line, specify a line number between the numbers of the lines that the new line is to be inserted between. The text of the new line to be inserted must also be specified on the command line following the line number.

To replace a line, specify its line number and enter the replacement text after the line number on the command line.

A line is deleted if its line number is specified and the replacement line is omitted.

Attempting to delete a nonexistent line results in an error message being printed. MAE does not permit deletion of a line if the macro consists of only that line. NOMA must be used to remove a macro. To define new macros, use MA; the MAE command operates only on previously defined macros.

Line numbers serve one purpose: specifying the location within a macro definition to perform the editing function. After the editing is complete, the macro definition is displayed with a new set of line numbers.

Examples:

```
143-Bug>MA ABC                    List definitions of macro ABC.
MACRO ABC
010 MD 3000
020 GO \0
143-Bug>
```

```
143-Bug>MAE ABC 15 RD
MACRO ABC
010 MD 3000
020 RD
030 GO \0
143-Bug>
```

Add a line to macro ABC.

This line was inserted.

```
143-Bug>MAE ABC 10 MD 10+R0
MACRO ABC
010 MD 10+R0
020 RD
030 GO \0
143-Bug>
```

Replace line 10.

This line was overwritten.

```
143-Bug>MAE ABC 30
MACRO ABC
010 MD 10+R0
020 RD
143-Bug>
```

Delete line 30.

Enable/Disable Macro Expansion Listing

MAL
NOMAL

MAL
NOMAL

The MAL command allows the user to view expanded macro lines as they are executed. This is especially useful when errors result, as the line that caused the error appears on the display.

The NOMAL command is used to suppress the listing of the macro lines during execution.

The use of MAL and NOMAL is a convenience for the user and in no way interacts with the function of the macros.

Save/Load Macros

MAW
MAR

MAW [*controller LUN*][*del*[*device LUN*][*del block #*]]
MAR [*controller LUN*][*del*[*device LUN*][*del block #*]]

controller LUN – is the LUN of the controller to which the above device is attached. Defaults to LUN 0.

device LUN – is the LUN of the device to save/load macros to/from. Initially defaults to LUN 0.

del – is a field delimiter: comma (,) or spaces ( ).

block # – is the number of the block on the above device that is the first block of the macro list. Initially defaults to block 2.

The MAW command allows the user to save the currently defined macros to disk/tape. A message is printed listing the block number, controller LUN, and device LUN before any writes are made. This message is followed by a prompt (OK to proceed (y/n)?). The user may then decline to save the macros by typing the letter N (uppercase or lowercase). Typing the letter Y (uppercase or lowercase) permits MAW to proceed and write the macros out to disk/tape. The list is saved as a series of strings and may take up to three blocks. If no macros are currently defined, no writes are done to disk/tape and NO MACRO DEFINED is printed.

The MAR command allows the user to load macros that are saved by MAW. Care should be taken to avoid attempting to load macros from a location on the disk/tape other than that written to by the MAW command. While MAR checks for invalid macro names and other anomalies, the results of such a mistake are unpredictable.

NOTE

MAR discards all currently defined macros before loading from disk/tape.

Defaults change each time MAR and MAW are invoked. When either has been used, the default controller, device, and block numbers are set to those used for that command. If macros were loaded from controller 0, device 2, block 8 via command MAR, then the defaults for a later invocation of MAW or MAR would be controller 0, device 2, and block 8.

Errors encountered during I/O are reported along with the 16-bit status word returned by the disk I/O routines.

Examples: (Assume that controller 0, device 2 are accessible.)

```
143-Bug>MAR 0,2,3                          Load macros from block 3.
143-Bug>

143-Bug>MA                                 List macros.
MACRO ABC
010 MD 3000
020 GO \0
143-Bug>

143-Bug>MA ASM                             Define macro ASM.
M=MM \0;DI
M=(CR)
143-Bug>

143-Bug>MA                                 List all macros.
MACRO ABC
010 MD 3000
020 GO \0
MACRO ASM
010 MD \0;DI
143-Bug>

143-Bug>MAW ,,8                            Save macros to block 8, previous device.
WRITING TO BLOCK $8 ON CONTROLLER $0, DEVICE $2
OK to proceed (y/n)? Y                     Carriage return not needed.
143-Bug>
```

Memory Display

MD[S] *addr*[:*count* | *addr*][;[**B**|**W**|**L**|**S**|**D**|**X**|**P**|**DI**]]

The MD[S] command is used to display the contents of multiple memory locations all at once. MD accepts the following data types:

Integer Data Type	Floating Point Data Types
B – Byte	**S** – Single Precision
W – Word	**D** – Double Precision
L – Longword	**X** – Extended Precision
	P – Packed Decimal

The default data type is word. Also, for the integer data types, the data is always displayed in hex along with its ASCII representation. The DI option enables the Resident MC68030 disassembler. No other option is allowed if DI is selected.

Refer to Chapter 2 for use of a function code as part of the *addr* field.

The optional count argument in the MD command specifies the number of data items to be displayed (or the number of disassembled instructions to display if the disassembly option is selected) defaulting to 8 if none is entered. The default count is changed to 128 if the S (sector) modifier is used. Entering only CR at the prompt immediately after the command has completed causes the command to re-execute, displaying an equal number of data items or lines beginning at the next address.

Example 1:

```
143-Bug>md 12000
00012000 2800  1942  2900  1942   2800  1842  2900  2846      (..B)..B(..B).(F
143-Bug>(CR)
00012010 FC20  0050  ED07  9F61   FF00  000A  E860  F060      | .Pm..a....h`p`
```

Example 2: Assume the following processor state: A2=00013500,D5=53F00127

```
143-Bug>md (a2,d5):&19;b
00013627 4F 82 00 C5 9B 10 33 7A  DF 01 6C 3D 4B 50 0F 0F     O..E..3z_.l=KP..
00013637 31 AB 80                                            1+.
143-Bug>
```

Example 3:

```
143-Bug>md 50008;di
00050008 46FC2700              MOVE.W   #9984,SR
0005000C 61FF0000023E          BSR.L    #5024C
00050012 4E7AD801              MOVEC.L  VBR,A5
00050016 41ED7FFC              LEA.L    32764(A5),A0
0005001A 5888                  ADDQ.L   #4,A0
0005001C 2E48                  MOVE.L   A0,A7
0005001E 2C48                  MOVE.L   A0,A6
00050020 13C7FFFB003A          MOVE.B   D7,($FFFB003A).L
143-Bug>
```

Example 4:

```
143-Bug>md 5000;d
00005000 0_3F6_44C1D0F047FC2= 2.4777000000000002_E-0003
00005008 0_423_DAEFF04800000= 1.2749000000000000_E+0011
00005010 0_000_0000000000000= 0.0000000000000000_E+0000
00005018 0_403_0000000000000= 1.6000000000000000_E+0001
00005020 1_3FF_0000000000000=-1.0000000000000000_E+0000
00005028 0_000_00000FFFFFFFF= 2.1219957904712067_E+0314
00005030 0_44D_FDE9F10A8D361= 6.0200000000000000_E+0023
00005038 0_3C0_79CA10C924223= 1.5999999999999999_E+0019
143-Bug>
```

Menu

MENU

3

The MENU command works only if the 143Bug is in the "system" mode (refer to the *ENV Command* in this chapter). When invoked in the "system" mode, it provides a way to exit 143Bug and return to the menu.

The following is an example of command line entries and their definitions.

```
143-Bug>MENU

1         Continue System Start Up
2         Select Alternate Boot Device
3         Go to System Debugger
4         Initiate Service Call
5         Display System Test Errors
6         Dump Memory to Tape
Enter Menu #:
```

When the 143Bug is in "system" mode, a user can toggle back and forth between the menu and Bug by typing a 3 in response to the Enter Menu #: prompt when the menu is displayed. Entering the Bug and then typing MENU in response to the 143-Bug (or 143-Diag) prompt returns you to the system menu.

For details on use of the menu features, refer to Appendix A, *System Mode Operation*.

Memory Modify

MM

MM *addr* [;[[**B**|**W**|**L**|**S**|**D**|**X**|**P**][**A**][**N**]]|[**DI**]]

The MM command is used to examine and change memory locations. MM accepts the following data types:

Integer Data Type	Floating Point Data Types
B – Byte	**S** – Single Precision
W – Word	**D** – Double Precision
L – Longword	**X** – Extended Precision
	P – Packed Decimal

The default data type is word. The MM command (alternate form "M") reads and displays the contents of memory at the specified address and prompts the user with a question mark ("?"). The user may enter new data for the memory location, followed by <CR>, or may simply enter <CR>, which leaves the contents unaltered. That memory location is closed and the next location is opened.

Refer to Chapter 2 for use of a function code as part of the *addr* field.

The user may also enter one of several special characters, either at the prompt or after writing new data, which change what happens when the carriage return is entered. These special characters are as follows:

V or **v** The next successive memory location is opened. (This is the default. It is in effect whenever MM is invoked and remains in effect until changed by entering one of the other special characters.)

^ MM backs up and opens the previous memory location.

= MM re-opens the same memory location (this is useful for examining I/O registers or memory locations that are changing over time).

. Terminates MM command. Control returns to 143Bug.

The N option of the MM command disables the read portion of the command. The A option forces alternate location accesses only.

Example 1:

```
143-Bug>mm 10000                    Access location 10000.
00010000 1234? (CR)
00010002 5678? 4321                 Modify memory.
00010004 9ABC? 8765^                Modify memory and backup.
00010002 4321? (CR)
00010000 1234? abcd.                Modify memory and exit.
```

Example 2:

```
143-Bug>mm 10001;la                 Longword access to location 10001
00010001 CD432187? (CR)             (alternate location accesses).
00010009 00068010? 68010+10=        Modify and reopen location.
00010009 00068020? (CR)
00010009 00068020? .                Exit MM.
```

The DI option enables the one-line assembler/disassembler. All other options are invalid if DI is selected. The contents of the specified memory location are disassembled and displayed and the user is prompted with a question mark ("?") for input. At this point the user has three options:

a. Enter (CR). This closes the present location and continues with disassembly of next instruction.

b. Enter a new source instruction followed by (CR). This invokes the assembler, which assembles the instruction and generates a "listing file" of one instruction.

c. Enter .(CR). This closes the present location and exits the MM command.

If a new source line is entered (choice 2 above), the present line is erased and replaced by the new source line entered. In the hardcopy mode, a line feed is done instead of erasing the line.

If an error is found during assembly, the symbol "^" appears below the field suspected of the error, followed by an error message. The location being accessed is redisplayed.

For additional information about the assembler, refer to Chapter 4.

The examples below were made in the hardcopy mode.

Example 3: Assemble a new source line.

```
143-Bug>mm 10000;di
00010000 46FC2400                MOVE.W   #9216,SR ? divs.w -(a2),d2
00010000 85E2                    DIVS.W   -(A2),D2
00010002 2400                    MOVE.L   D0,D2 ?
```

Example 4: New source line with error.

```
00010008 4E7AD801                MOVEC.L  VBR,A5 ? bchg #$12,9(a5,d6))
00010008                         BCHG     #12,9(A5,D6))
                                                      ^
*** Unknown Field ***
00010008 4E7AD801                MOVEC.L  VBR,A5 ?
```

Example 5: Step to next location and exit MM.

```
143-Bug>m 1000c;di
0001000C 000000FF                OR.B     #255,D0 ? (CR)
00010010 20C9                    MOVE.L   A1,(A0)+ ? .
143-Bug>
```

Example 6:

```
143-Bug>m 7000;x
00007000 0_0000_FFFFFFFF00000000? 1_3C10_84782
0000700C 1_7FFF_00000000FFFFFFFF? 0_001A_F
00007018 0_0000_FFFFFFFF00000000? 6.02E23=
00007018 0_404D_FEF4F885469B0880? ^
0000700C 0_001A_F0000000000000000? (CR)
00007000 1_3C10_84782000000000000? .
143-Bug>
```

Memory Set

MS

MS *addr* [*hexadecimal number*]. . . | ['string']. . .

3

MS command is used to write data to memory starting at the specified address. Hex numbers are not assumed to be of a particular size, so they can contain any number of digits (as allowed by command line buffer size). If an odd number of digits are entered, the least significant nibble of the last byte accessed is be unchanged.

Refer to Chapter 2 for use of a function code as part of the *addr* field.

ASCII strings can be entered by enclosing them in single quotes ('). To include a quote as part of a string, two consecutive quotes should be entered.

Example: Assume that memory is initially cleared:

```
143-Bug>ms 25000 0123456789abcDEF 'This is ''143Bug''' 23456
143-Bug>md 25000:20;b
00025000 0123 4567 89AB CDEF   5468 6973 2069 7320    .#Eg.+MoThis is
00025010 2731 3433 4275 6727   2345 6000 0000 0000    '143Bug'#E'.....
143-Bug>
```

Set Memory Address From VMEbus

OBA

The OBA (Off-Board Address) command allows the user to set the base address of the MVME143 onboard RAM, as seen from the VMEbus. (Refer to Chapter 1.) Therefore, the user should enter the hex number corresponding to the actual base address, so that the offboard external devices on the VMEbus will know where it is. The default (factory-delivered) condition is with the offboard address set to $0.

Example:

```
143-Bug>oba
RAM address from VMEbus = $0 400000          Change $0 to $400000.
143-Bug>oba
RAM address from VMEbus = $400000 (CR)
143-Bug
```

Offset Registers Display/Modify

OF

OF [R*n*[;A]]

3

OF allows the user to access and change pseudo-registers called offset registers. These registers are used to simplify the debugging of relocatable and position-independent modules (refer to Chapter 2).

There are eight offset registers R0-R7, but only R0-R6 can be changed. R7 always has both base and top addresses set to 0. This allows the automatic register function to be effectively disabled by setting R7 as the automatic register.

Each offset register has two values: base and top. The base is the absolute least address that will be used for the range declared by the offset register. The top address is the absolute greatest address that will be used. When entering the base and top, the user may use either an address/address format or an address/count format. If a count is specified, it refers to bytes. If the top address is omitted from the range, then a count of 1Mb is assumed. The top address must equal or exceed the base address. Wraparound is not permitted.

Command usage:

OF – To display all offset registers. An asterisk indicates which register is the automatic register.

OF R*n* – To display/modify R*n*. The user can scroll through the registers in a way similar to that used by the MM command.

OF R*n*;A – To display/modify R*n* and set it as the automatic register. The automatic register is one that is automatically added to each absolute address argument of every command except if an offset register is explicitly added. An asterisk indicates which register is the automatic register.

Range entry: Ranges may be entered in three formats: base address alone, base and top as a pair of addresses, and base address followed by byte count. Control characters "^", "v", "V", "=", and "." may be used. Their function is identical to that in Register Modify (RM) and Memory Modify (MM) commands.

Range syntax: [*base address* [*del top address*]] [^|v|=|.]
or
[*base address* [':' *byte count*]] [^|v|=|.]

Offset register rules:

a. At power up and cold-start reset, R7 is the automatic register.

b. At power-up and cold-start reset, all offset registers have both base and top addresses preset to 0. This effectively disables them.

c. R7 always has both base and top addresses set to 0; cannot be changed.

d. Any offset register can be set as the automatic register.

e. The automatic register is always added to every absolute address argument of every 143Bug command where there is not an offset register explicitly called out.

f. There is always an automatic register. A convenient way to disable the effect of the automatic register is by setting R7 as the automatic register. Note that this is the default condition.

Examples:

Display offset registers.

```
143-Bug>OF
R0 =00000000 00000000  R1 = 00000000 00000000
R2 =00000000 00000000  R3 = 00000000 00000000
R4 =00000000 00000000  R5 = 00000000 00000000
R6 =00000000 00000000  R7*= 00000000 00000000
```

Modify some offset registers.

```
143-Bug>OF R0
R0 =00000000 00000000? 20000 200FF
R1 =00000000 00000000? 25000:200^
R0 =00020000 000200FF? .
```

Look at location $20000.

```
143-Bug>M 20000;DI
00000+R0 41F95445 5354                  LEA.L   ($54455354).L,A0 .
143-Bug>M R0;DI
00000+R0 41F95445 5354                  LEA.L   ($54455354).L,A0 .
143-Bug>
```

Set R0 as the automatic register.

```
143-Bug>OF R0;A
R0*=00020000 000200FF? .
```

To look at location $20000.

```
143-Bug>M 0;DI
00000+R0 41F95445 5354                    LEA.L    ($54455354).L,A0 .
143-Bug>
```

To look at location 0, override the automatic offset.

```
143-Bug>M 0+R7;DI
00000000 FFF8                             DC.W     $FFF8 .
```

Printer Attach/Detach

PA
NOPA

PA [*n*]
NOPA [*n*]

These two commands "attach" or "detach" a printer to the user-specified port. Multiple printers may be attached. When the printer is attached, everything that appears on the system console terminal is also echoed to the "attached" port. PA is used to attach, NOPA is used to detach. If no port is specified, PA does not attach any port, but NOPA detaches all attached ports.

If the port number specified is not currently assigned, PA displays a message. If NOPA is attempted on a port that is not currently attached, a message is displayed.

The port being attached must already be configured. This is done using the Port Format (PF) command. This is done by executing the following sequence prior to "PA*n*".

```
143-Bug>PF4
Logical unit $04 unassigned
Name of board? VME143
Name of port? PTR
Port base address = $FFFE2800? (CR)
Auto Line Feed protocol [Y,N] = N? Y.
OK to proceed (y/n)? Y
143-Bug>
```

For further details, refer to the PF command.

Examples:

```
CONSOLE DISPLAY:                    PRINTER OUTPUT:
143-Bug>PA4
(attaching port 4)                  (printer now attached)
143-Bug>HE NOPA                     143-Bug>HE NOPA
NOPA      Printer detach            NOPA      Printer detach
143-Bug>NOPA                        143-Bug>NOPA
(detach all attached printers)      (printer now detached)
143-Bug>NOPA
No printer attached
143-Bug>
```

Port Format/Detach

PF
NOPF

3

PF[*n*]
NOPF*n*

Port Format (PF) allows the user to examine and change the serial input/output environment. PF may be used to display a list of the current port assignments, configure a port that is already assigned, or assign and configure a new port. Configuration is done interactively, much like modifying registers or memory (RM and MM commands). An interlock is provided prior to configuring the hardware -- the user must explicitly direct PF to proceed.

ONLY NINE PORTS MAY BE ASSIGNED AT ANY GIVEN TIME. PORT NUMBERS MUST BE IN THE RANGE 0 TO $1F.

Listing Current Port Assignments

Port Format lists the names of the module (board) and port for each assigned port number (LUN) when the command is invoked with the port number omitted.

Example:

```
143-Bug>pf
Current port assignments:  (Port #: Board name, Port name)
[00: VME143- "1"]  [01: VME143- "2"]  [02: VME143- "3"]
143-Bug>
```

Configuring A Port

The primary use of Port Format is changing baud rates, stop bits, etc. This may be accomplished for assigned ports by invoking the command with the desired port number. Assigning and configuring may be accomplished consecutively. Refer to *Assigning A New Port* under the *PF Command* paragraph in this chapter.

When Port Format is invoked with the number of a previously assigned port, the interactive mode is entered immediately. To exit from the interactive mode, enter a period by itself or following a new value/setting. While in the interactive mode, the following rules apply:

Only listed values are accepted when a list is shown. The sole exception is that upper- or lowercase may be interchangeably used when a list is shown. Case takes on meaning when the letter itself is used, such as XON character value.

^ Control characters are accepted by hexadecimal value or by a letter preceded by a caret (i.e., Control-A would be "^A").

The caret, when entered by itself or following a value, causes Port Format to issue the previous prompt after each entry.

v Either uppercase or lowercase "v" causes Port Format to resume or prompting in the original order (i.e., Baud Rates, then Parity Type, V ...).

= Entering an equal sign by itself or when following a value causes PF to issue the same prompt again. This is supported to be consistent with the operation of other debugger commands. To resume prompting in either normal or reverse order, enter the letter "v" or a caret "^", respectively.

. Entering a period by itself or following a value causes Port Format to exit from the interactive mode and issue the "OK to proceed (y/n?)".

(CR) Pressing return without entering a value preserves the current value and causes the next prompt to be displayed.

Example:

```
143-Bug>PF 1
Baud rate [110,300,600,1200,2400,4800,9600,19200] = 9600? (CR)
Even, Odd, or No Parity [E,O,N] = N? (CR)
Char width [5,6,7,8] = 8? (CR)
Stop Bits [1,2] = 1? 2                    (new value entered)
(the next response is to demonstrate reversing the order of prompting)
Async, Mono, Bisync, Gen, SDLC, or HDLC [A,M,B,G,S,H] = A ^
Stop Bits [1,2] = 2? .                    (value acceptable, exit interactive mode)
OK to proceed (y/n)? Y                    (carriage return not required)
143-Bug>
```

PF
NOPF

Parameters Configurable By Port Format

Port base address:

Upon assigning a port, the option is provided to set the base address. This is useful for support of modules with adjustable base addressing, such as the MVME050. Entering no value selects the default base address shown.

Baud rate:

The user may choose from the following: 110, 300, 600, 1200, 2400, 4800, 9600, 19200. IF A NUMBER BASE IS NOT SPECIFIED, THE DEFAULT IS DECIMAL, NOT HEXADECIMAL.

Parity type:

Parity may be even (choice E), odd (choice O), or disabled (choice N).

Character width:

The user may select 5-, 6-, 7-, or 8-bit characters.

Number of stop bits:

Only 1 and 2 stop bits are supported.

Synchronization type:

Because the debugger is a polled serial input/output environment, most users use only asynchronous communication. The synchronous modes are permitted.

Synchronization character values:

Any 8-bit value or ASCII character may be entered.

Automatic software handshake:

Current drivers have the capability of responding to XON/XOFF characters sent to the debugger ports. Receiving an XOFF causes a driver to cease transmission until an XON character is received.

Software handshake character values:

The values used by a port for XON and XOFF may be redefined to be any 8-bit value. ASCII control characters or hexadecimal values are accepted.

Assigning A New Port

Port Format supports a set of drivers for a number of different modules and the ports on each. To assign one of these to a previously unassigned port number, invoke the command with that number. A message is then printed to indicate that the port is unassigned and a prompt is issued to request the name of the module (such as VME143, VME050, etc.). Pressing the RETURN key on the console at this point causes PF to list the currently supported modules and ports. Once the name of the module (board) has been entered, a prompt is issued for the name of the port. After the port name has been entered, Port Format attempts to supply a default configuration for the new port.

When a valid port has been specified, default parameters are supplied. The base address of this new port is one of these default parameters. Before entering the interactive configuration mode, the user is allowed to change the port base address. Pressing the RETURN key on the console retains the base address shown.

If the configuration of the new port is not fixed, then the interactive configuration mode is entered. Refer to the *Configuring A Port* paragraph above regarding configuring assigned ports. If the new port does have a fixed configuration, then Port Format issues the "OK to proceed (y/n)?" prompt immediately.

Port Format does not initialize any hardware until the user has responded with the letter "Y" to prompt "OK to proceed (y/n)?". Pressing the BREAK key on the console any time prior to this step or responding with the letter "N" at the prompt leaves the port unassigned. This is only true of ports not previously assigned.

Example: Assigning port 7 to the MVME050 printer port.

```
143-Bug>PF 7
Logical unit $07 unassigned
Name of board? (CR)                (cause PF to list supported modules (boards), ports)
Boards and ports supported:
VME143:  1,2,3,4,PTR
VME050:  1,2,PTR2
Name of board? VME050              (uppercase or lowercase accepted)
Name of port? PTR2
Port base address = $FFFF1080? (CR)
Auto Line Feed protocol [Y,N] = N? .
(interactive mode not entered because hardware has fixed configuration)
OK to proceed (y/n)? Y
143-Bug>
```

NOPF Port Detach

The NOPF command, NOPF*n*, unassigns the port whose number is *n*. Only one port may be unassigned at a time. Invoking the command without a port number, "NOPF", does not unassign any ports.

Put RTC Into Power Save Mode For Storage

PS

The PS command is used to turn off the oscillator in the RTC chip, MK48T02. The MVME143 module is shipped with the RTC oscillator stopped to minimize current drain from the on-chip battery. Normal cold-start of the MVME143 with the 143Bug EPROMs installed gives the RTC a "kick start" to begin oscillation. To disable the RTC, the user must enter "PS".

The SET command restarts the clock. Refer to the *SET Command* in this chapter for further information.

Example:

```
143-Bug>PS
(Clock is in Battery Save Mode)
143-Bug>
```

ROMboot Enable/Disable

RB
NORB

The RB command enables the search for and booting from a routine nominally encoded in on-board ROMs/PROMs/EPROMs/EEPROMs on the MVME143. However, the routine can be in other memory locations, as detailed in the RB command options given below. Refer also to the *ROMboot* paragraph and example in Chapter 1.

NORB disables the search for a ROMboot routine, but does not change the options chosen.

The default condition is with the ROMboot function disabled.

Examples:

```
143-Bug>RB
Boot at power-up only    [Y,N] ? Y (CR)
```
If the user types N, then boot is attempted at any board reset.

```
Enable search of VMEbus [Y,N] ? N (CR)
```
If the user types Y, the search for "BOOT", etc., starts at the end of onboard memory, in 8Kb increments.

```
Boot direct address = $FFF00000 (CR)
```
This default address is the start of the 143Bug EPROMs, so the search here is fast.

```
143-Bug>NORB
ROM boot disabled
143-Bug>
```
This disables the ROMboot function but does not change any options chosen under RB.

Register Display

RD

RD [[+|-|=][*dname*][/]]. . .[[+|-|=][*reg1*[-*reg2*]][/]]. . .

The RD command is used to display the target state, that is, the register state associated with the target program (refer to the GO command). The instruction pointed to by the target PC is disassembled and displayed also. Internally, a register mask specifies which registers are displayed when RD

The arguments are as follows:

+	is a qualifier indicating that a device or register range is to be added.
–	is a qualifier indicating that a device or register range is to be removed, except when used between two register names. In this case, it indicates a register range.
=	is a qualifier indicating that a device or register range is to be set.
/	is a required delimiter between device names and register ranges.
reg1	is the first register in a range of registers.
reg2	is the last register in a range of registers.
dname	is a device name. This is used to quickly enable or disable all the registers of a device. The available device names are:

MPU	Microprocessor Unit
MMU	Memory Management Unit
FPC	Floating Point Coprocessor

Observe the following notes when specifying any arguments in the command line:

a. The qualifier is applied to the next register range only.

b. If no qualifier is specified, a + qualifier is assumed.

c. All device names should appear before any register names.

d. The command line arguments are parsed from left to right, with each field being processed after parsing, thus, the sequence in which qualifiers and registers are organized has an impact on the resultant register mask.

e. When specifying a register range, *reg1* and *reg2* do not have to be of the same class.

f. The register mask used by RD is also used by all exception handler routines, including the trace and breakpoint exception handlers.

The MPU registers in ordering sequence are:

NUMBER AND TYPE OF REGISTERS		MNEMONICS
10	System Registers	(PC,SR,USP,MSP,ISP,VBR,SFC,DFC, CACR,CAAR)
8	Data Registers	(D0–D7)
8	Address Registers	(A0–A7)

(Total: 26 Registers. Note that A7 represents the active stack pointer, which leaves 25 different registers.)

The MMU registers in ordering sequence are:

NUMBER AND TYPE OF REGISTERS		MNEMONICS
5	Address Translation/Control	(CRP,SRP,TC,TT0,TT1)
1	Status	(MMUSR)

The FPC registers in ordering sequence are:

NUMBER AND TYPE OF REGISTERS		MNEMONICS
3	System Registers	(FPCR,FPSR,FPIAR)
8	Data Registers	(FP0–FP7)

Example 1: Default display - MPU registers only.

```
143-Bug>rd
PC  =00004000 SR  =2700=TR:OFF_S._7_..... VBR =00000000
USP =0000F830 MSP =00005C18 ISP*=00006000 SFC =0=FC
CACR=0=D:...._I:...          CAAR=00000000 DFC =0=FC
D0  =00000000 D1  =00000000 D2  =00000000 D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00006000
00004000 4AFC              ILLEGAL
143-Bug>
```

NOTE

An asterisk following a stack pointer name indicates that it is the active stack pointer.

The status register includes a mnemonic portion to help in reading it:

TRACE BITS			
T1	**T0**	**MNEMONIC**	**DESCRIPTION**
0	0	TR:OFF	Trace off
0	1	TR:CHG	Trace on change of flow
1	0	TR:ALL	Trace all states
1	1	TR:INV	Invalid mode

S, M bits: The bit name appears (S,M) if the respective bit is set, otherwise a "." indicates that it is cleared.

Interrupt Mask: A number (0 to 7) indicates current processor priority level.

Condition Codes: The bit name appears (X,N,Z,V,C) if the respective bit is set, otherwise a "." indicates that it is cleared.

The source and destination function code registers (SFC, DFC) include a two character mnemonic:

FUNCTION CODE	MNEMONIC	DESCRIPTION
0	F0	Undefined
1	UD	User Data
2	UP	User Program
3	F3	Undefined
4	F4	Undefined
5	SD	Supervisor Data
6	SP	Supervisor Program
7	CS	CPU Space

The CACR register shows mnemonics for two bits: Enable and Freeze. The bit name (E, F) appears if the respective bit is set, otherwise a "." indicates that it is cleared.

Example 2: To display only the MMU registers.

```
143-Bug>RD =MMU
CRP  =00000001_00000000        SRP  =00000001_00000000
TC   =00000000 TT0  =00000000 TT1  =00000000
MMUSR=0000=......._0          PSR  =0000-........._0
00004000 4AFC                ILLEGAL
143-Bug>
```

The MMUSR register above includes a mnemonic portion, the bits are:

B	Bus error	bit 15
L	Limit Violation	bit 14
S	Supervisor only	bit 13
W	Write protected	bit 11
I	Invalid	bit 10
M	Modified	bit 9
T	Transparent Access	bit 6
N	Number of Levels (3 bits)	bits 2-0

Example 3: To display only the FPC registers.

```
143-Bug>RD =fpc
FPCR =00000000  FPSR =00000000-(CC=....    )   FPIAR=00000000
FP0  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP1  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP2  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP3  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP4  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP5  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP6  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP7  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
00004000 4AFC                ILLEGAL
143-Bug>
```

The floating point data registers are always displayed in extended precision and in scientific notation format. The floating point status register display includes a mnemonic portion for the condition codes. The bit name appears (N, X, I, NAN) if the respective bit is set, otherwise, a "." indicates that it is cleared.

Example 4: To remove D3 through D5 and A2, and add FPSR and FP0, starting with the previous display.

```
143-Bug>RD MPU/-FPC/-D3-D5/-A2/FP0/FPSR
PC   =00004000 SR  =2700=TR:OFF_S._7_..... VBR =00000000
USP  =0000F830 MSP =00005C18 ISP*=00006000 SFC =0=F0
CACR =0=D:...._I:...         CAAR=00000000 DFC =0=F0
D0   =00000000 D1  =00000000 D2  =00000000 D6  =00000000
D7   =00000000 A0  =00000000 A1  =00000000 A3  =00000000
A4   =00000000 A5  =00000000 A6  =00000000 A7  =00006000
FPSR =00000000-(CC=....  )
FP0  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
00004000 4AFC             ILLEGAL
143-Bug>
```

Example 5: To set the display to D6 and A3 only.

```
143-Bug>RD =D6/A3
D6   =00000000 A3   =00000000
00004000 4AFC                ILLEGAL
143-Bug>
```

Note that the above sequence sets the display to D6 only and then adds register A3 to the display.

Example 6: To restore all the MPU registers.

```
143-Bug>rd +mpu
PC   =00004000 SR   =2700=TR:OFF_S._7_..... VBR =00000000
USP  =0000F830 MSP  =00005C18 ISP*=00006000 SFC =0=FC
CACR=0=D:...._I:...           CAAR=00000000 DFC =0=FC
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00006000
00004000 4AFC              ILLEGAL
143-Bug>
```

Note that an equivalent command would have been RD PC–A7.

Remote

REMOTE

The REMOTE command duplicates the remote operation modem functions available from the "system" mode menu command, entry number 4. It is only accessible when the 143Bug is in "system" mode (refer to the *MENU Command* paragraph in Appendix A for details on remote operation).

Cold/Warm Reset

RESET

The RESET command is used to issue a local SCSI bus reset and also allows the user to specify the level of reset operation that will be in effect when a RESET exception is detected by the processor. A reset exception can be generated by pressing the RESET switch on the MVME143 front panel, or by executing a software reset.

Two RESET levels are available:

COLD – This is the standard level of operation, and is the one defaulted to on power-up. In this mode, all the static variables are initialized every time a reset is done.

WARM – In this mode, all the static variables are preserved when a reset exception occurs. This is convenient for keeping breakpoints, offset register values, the target register state, and any other static variables in the system.

NOTE

If the MVME143 is the system controller, pressing the RESET switch resets all the modules in the system, including disk controllers like the MVME320 or MVME360. This may cause the disk controller configuration to be out of phase with respect to the disk configuration tables in memory.

Example 1:

```
143-Bug>RESET
Cold/Warm Reset [C,W] = C? .
143-Bug>
```

Example 2:

```
143-Bug>RESET
Cold/Warm Reset [C,W] = C? W
Execute Soft Reset [Y,N] ? Y
```

Arm for warm start the next time a reset is performed.
Do a software reset now, actually forcing a warm start.

```
Copyright Motorola Inc. 1988, All Rights Reserved

VME143 Monitor/Debugger Release 1.0 - 4/08/88

WARM Start
143-Bug>
```

Register Modify

RM

RM *reg*

3

RM command allows the user to display and change the target registers. It works in essentially the same way as the MM command, and the same special characters are used to control the display/change session (refer to the *MM Command* paragraph in this chapter).

NOTE

reg is the mnemonic for the particular register, the same as it is displayed.

Example 1:

```
143-Bug>RM D5
D5  =12345678? ABCDEF^                Modify register and back up.
D4  =00000000? 3000.                  Modify register and exit.
143-Bug>
```

Example 2:

```
143-Bug>rm sfc
SFC =7=CS     ? 1=                    Modify register and reopen.
SFC =1=UD     ? .                     Exit.
143-Bug>
```

The RM command is also used to modify the MMU registers.

Example 3:

```
143-Bug>rm crp
CRP   =00000001_00000000      ? (CR)
SRP   =00000001_00000000      ? (CR)
TC    =00000000 ? 87654321
TT0   =00000000 ? 12345678
TT1   =00000000 ? 87654321
MMUSR=0000=. . . . . _0? .
```

Register Set

RS

RS *reg* [*hexadecimal number*]. . .

The RS command allows the user to change the data in the specified target register. It works in essentially the same way as the RM command.

NOTE

reg is the mnemonic for the particular register.

Example 1:

```
143-Bug>RS D5 123455678                    Change MPU register.
D5    =12345678
143-Bug>
```

Example 2:

```
143-Bug>rs tt0 87654321                    Change MMU register.
TT0   =87654321
143-Bug>
```

Example 3:

```
143-Bug>rs FP0 0_1234_5                    Change FPC register.
FP0   =0_1234_5000000000000000= 6.6258385370745493_E-3530
143-Bug>
```

Switch Directories

SD

The SD command is used to change from the debugger directory to the diagnostic directory or from the diagnostic directory to the debugger directory.

The commands in the current directory (the directory that the user is in at the particular time) may be listed using the Help (HE) command.

The way the directories are structured, the debugger commands are available from either directory but the diagnostic commands are only available from the diagnostic directory.

Example 1:

```
143-Bug>SD
143-Diag>                    (The user has changed from the debugger )
                             (directory to the diagnostic directory, )
                             (as can be seen by the "143-Diag>"      )
                             (prompt.                                )
```

Example 2:

```
143-Diag>SD
143-Bug>                     (The user is now back in the debugger   )
                             (directory.                             )
```

Set Time And Date

SET

The SET command is interactive and begins with the user entering SET followed by a carriage return. At this time, a prompt asking for MM/DD/YY is displayed. The user may change the displayed date by typing a new date followed by (CR), or may simply enter (CR), which leaves the displayed date unchanged. When the correct date matches the data entered, the user should press the carriage return to establish the current value in the time-of-day clock.

Note that an incorrect entry may be corrected by backspacing or deleting the entire line as long as the carriage return has not been entered.

After the initial prompt and entry, another prompt is presented asking for a calibration value. This value slows down (- value) or speeds up (+ value) the RTC in the MK48T02 chip. Refer to the *MK48T02 Data Sheet* (as mentioned in Chapter 1, herein) for details.

Next, a prompt is presented asking for HH:MM:SS. The user may change the displayed time by typing a new time followed by (CR), or may simply enter (CR), which leaves the displayed time unchanged.

To display the current date and time of day, refer to the TIME command.

Example: To SET a date and time of May 11, 1988 2:05:32 PM the command is as follows:

```
143-Bug>SET
Weekday  xx/xx/xx    xx:xx:xx
Present calibration = -0
Enter date as MM/DD/YY
05/11/88
Enter Calibration value +/- (0 to 31)

Enter time as HH:MM:SS (24 hour clock)
14:05:32
143-Bug>
```

This will start a stopped clock. (Refer to the *PS Command* in this chapter.) This can speed up (+) or slow down (-) the RTC oscillator.

Trace

T

T [*count*]

3

The T command allows execution of one instruction at a time, displaying the target state after execution. T starts tracing at the address in the target PC. The optional *count* field (which defaults to 1 if none entered) specifies the number of instructions to be traced before returning control to 143Bug.

Breakpoints are monitored (but not inserted) during tracing for all trace commands, which allows the use of breakpoints in ROM or write-protected memory. In all cases, if a breakpoint with 0 count is encountered, control is returned to 143Bug.

The trace functions are implemented with the trace bits (T0, T1) in the MC68030 status register; therefore, these bits should not be modified by the user while using the trace commands.

Example: (The following program resides at location $10000.)

```
143-Bug>MD 10000;DI
00010000 2200      MOVE.L  D0,D1
00010002 4282      CLR.L   D2
00010004 D401      ADD.B   D1,D2
00010006 E289      LSR.L   #$1,D1
00010008 66FA      BNE.B   $10004
0001000A E20A      LSR.B   #$1,D2
0001000C 55C2      SCS.B   D2
0001000E 60FE      BRA.B   $1000E
143-Bug>
```

Initialize PC and D0:

```
143-Bug>RM PC
PC  =00008000 ? 10000.
143-Bug>RM D0
D0  =00000000 ? 8F41C.
```

Display target registers and trace one instruction:

```
143-Bug>RD
PC  =00010000 SR  =2700=TR:OFF_S._7_.....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =00000000 D2  =00000000 D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010000 2200              MOVE.L  D0,D1


143-Bug>T
PC  =00010002 SR  =2700=TR:OFF_S._7_.....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =0008F41C D2  =00000000 D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010002 4282              CLR.L   D2
143-Bug>
```

Trace next instruction:

```
143-Bug>(CR)
PC  =00010004 SR  =2704=TR:OFF_S._7_..Z..
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =0008F41C D2  =00000000 D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010004 D401              ADD.B   D1,D2
143-Bug>
```

Trace the next two instructions:

```
143-Bug>T 2
PC  =00010006 SR  =2700=TR:OFF_S._7_.....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =0008F41C D2  =0000001C D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010006 E289              LSR.L   #1,D1
PC  =00010008 SR  =2700=TR:OFF_S._7_.....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =00047A0E D2  =0000001C D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010008 66FA              BNE.B   $10004
143-Bug>
```

Terminal Attach

TA

TA [*port*]

TA command allows the user to assign any serial port to be the console. The port specified must already be assigned (refer to the *PF Command* paragraph in this chapter).

Example 1: Selecting port 2 (logical unit #02) as console.

```
143-Bug>TA 2
```

(No prompt appears unless port 2 was already the console.)

Example 2: Restoring console to port selected at power-up.

```
143-Bug>TA
```

(Prompt now appears at terminal connected to port 0.)

Trace On Change Of Control Flow

TC [*count*]

TC command starts execution at the address in the target PC and begins tracing upon the detection of an instruction that causes a change of control flow, such as JSR, BSR, RTS, etc. This means that execution is in real-time until a change of flow instruction is encountered. The optional count field (which defaults to 1 if none entered) specifies the number of change of flow instructions to be traced before returning control to 143Bug.

Breakpoints are monitored (but not inserted) during tracing for all trace commands, which allows the use of breakpoints in ROM or write-protected memory. Note that the TC command recognizes a breakpoint only if it is at a change of flow instruction. In all cases, if a breakpoint with 0 count is encountered, control is returned to 143Bug.

The trace functions are implemented with the trace bits (T0, T1) in the MC68030 status register; therefore, these bits should not be modified by the user while using the trace commands.

Example: (The following program resides at location $10000.)

```
143-Bug>MD 10000;DI
00010000 2200                MOVE.L  D0,D1
00010002 4282                CLR.L   D2
00010004 D401                ADD.B   D1,D2
00010006 E289                LSR.L   #$1,D1
00010008 66FA                BNE.B   $10004
0001000A E20A                LSR.B   #$1,D2
0001000C 55C2                SCS.B   D2
0001000E 60FE                BRA.B   $1000E
143-Bug>
```

Initialize PC and D0:

```
143-Bug>RM PC
PC  =00008000 ? 10000.
143-Bug>RM D0
D0  =00000000 ? 8F41C.
```

```
143-Bug>rd +mmu
PC   =00004000 SR  =2700=TR:OFF_S._7_..... VBR =00000000
USP  =00005830 MSP =00005C18 ISP*=00006000 SFC =0=FC
CACR =0=D:...._I:...         CAAR=00000000 DFC =0=FC
D0   =00000000 D1  =00000000 D2  =00000000 D3  =00000000
D4   =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0   =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4   =00000000 A5  =00000000 A6  =00000000 A7  =00006000
CRP  =00000001_00000000      SRP =00000001_00000000
TC   =87654321 TT0 =12345678 TT1 =87654321
MMUSR=0000=......._0
00004000 4AFC              ILLEGAL
143-Bug>
```

The RM command is also used to modify the floating point coprocessor (MC68882) registers.

Example 4:

```
143-Bug>rm fpsr
FPSR =00000000-(CC=....    )   ? F000000
FPIAR=00000000 ? (CR)
FP0  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF? 0_1234_5
FP1  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF? 1.25E3
FP2  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF? 1_7F_3FF
FP3  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF? 1100_9261_3
FP4  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF? &564
FP5  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF? 0_5FF_F0AB
FP6  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF? 3.1415
FP7  =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF? -2.74638369E-36.
143-Bug>
```

RM

```
143-Bug>rd +fpc
PC   =00004000 SR   =2700=TR:OFF_S._7_..... VBR =00000000
USP  =00005830 MSP  =00005C18 ISP*=00006000 SFC =0=FC
CACR =0=D:...._I:...          CAAR=00000000 DFC =0=FC
D0   =00000000 D1   =00000000 D2  =00000000 D3  =00000000
D4   =00000000 D5   =00000000 D6  =00000000 D7  =00000000
A0   =00000000 A1   =00000000 A2  =00000000 A3  =00000000
A4   =00000000 A5   =00000000 A6  =00000000 A7  =00006000
FPCR =00000000 FPSR =0F000000-(CC=NZI[NAN]) FPIAR=00000000
FP0  =0_1234_5000000000000000= 6.6258385370745493_E-3530
FP1  =0_4009_9C40000000000000= 1.2500000000000000_E-0003
FP2  =1_3FFF_BFF0000000000000= 1.4995117187500000_E-0000
FP3  =1_3C9D_BCEECF12D061BED9= 3.0000000000000000_E-0261
FP4  =0_4008_8D00000000000000= 5.6400000000000000_E-0002
FP5  =0_41FF_F855800000000000= 2.6012612226385672_E-0154
FP6  =0_4000_C90E5604189374BC= 3.1415000000000000_E-0000
FP7  =1_3F88_E9A2F0B8D678C318= 2.7463836900000000_E-0036
00004000 4AFC               ILLEGAL
143-Bug>
```

TC

Trace on change of flow:

```
143-Bug>TC
00010008 66FA              BNE.B   $10004
PC  =00010004 SR  =2700=TR:OFF_S._7_.....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =00047A0E D2  =0000001C D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010004 D401              ADD.B   D1,D2
```

(Note that this display also shows the change of flow instruction.)

Display Time And Date

TIME

The TIME command presents the date and time in ASCII characters to the console.

To initialize the time-of-day clock, refer to the SET command.

Example: A date and time of Wednesday, May 11, 1988 2:05:32 would be displayed as:

```
143-Bug>TIME
Wednesday  5/11/88    14:05:32
143-Bug>
```

Transparent Mode

TM

TM [*n*] [*escape*]

TM command essentially connects the console serial port and the host port together, allowing the user to communicate with a host computer. A message displayed by TM shows the current escape character (i.e., the character used to exit the transparent mode). The two ports remain "connected" until the escape character is received by the console port. The escape character is not transmitted to the host, and at power up or reset it is initialized to $01=^A.

The optional port number *n* allows the user to specify which port is the "host" port. If omitted, port 1 is assumed.

The ports do not have to be at the same baud rate, but the terminal port baud rate should be equal to or greater than the host port baud rate for reliable operation. To change the baud rate use the Port Format (PF) command.

The optional escape argument allows the user to specify the character to be used as the exit character. This can be entered in three different formats:

ASCII code	:	$03	Set escape character to ^C
control character	:	^C	Set escape character to ^C
ASCII character	:	'c	Set escape character to c

If the port number is omitted and the escape argument is entered as a numeric value, precede the escape argument with a comma to distinguish it from a port number.

Example 1:

```
143-Bug>TM                          Enter TM.
Escape character:  $01=^A           Exit code is always displayed.

^A                                  Exit transparent mode.
```

Example 2:

```
143-Bug>TM ^g                       Enter TM and set escape character
Escape character:  $07=^G           to ^G.

^G                                  Exit transparent mode.
143-Bug>
```

Trace To Temporary Breakpoint

TT

TT *addr*

3

TT command sets a temporary breakpoint at the specified address and traces until a breakpoint with 0 count is encountered. The temporary breakpoint is then removed (TT is analogous to the GT command) and control is returned to 143Bug. Tracing starts at the target PC address.

Breakpoints are monitored (but not inserted) during tracing for all trace commands, which allows the use of breakpoints in ROM or write-protected memory. If a breakpoint with 0 count is encountered, control is returned to 143Bug.

The trace functions are implemented with the trace bits (T0, T1) in the MC68030 status register; therefore, these bits should not be modified by the user while using the trace commands.

Example: (The following program resides at location $10000.)

```
143-Bug>MD 10000;DI
00010000 2200      MOVE.L  D0,D1
00010002 4282      CLR.L   D2
00010004 D401      ADD.B   D1,D2
00010006 E289      LSR.L   #$1,D1
00010008 66FA      BNE.B   $10004
0001000A E20A      LSR.B   #$1,D2
0001000C 55C2      SCS.B   D2
0001000E 60FE      BRA.B   $1000E
143-Bug>
```

Initialize PC and D0:

```
143-Bug>RM PC
PC  =00008000 ? 10000.
143-Bug>RM D0
D0  =00000000 ? 8F41C.
```

Trace to temporary breakpoint:

```
143-Bug>TT 10006
PC  =00010002 SR  =2700=TR:OFF_S._7_.....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =0008F41C D2  =00000000 D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010002 4282              CLR.L   D2
PC  =00010004 SR  =2704=TR:OFF_S._7_..Z..
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =0008F41C D2  =00000000 D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010004 D401              ADD.B   D1,D2
At Breakpoint
PC  =00010006 SR  =2700=TR:OFF_S._7_.....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =0008F41C D2  =0000001C D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010006 E289              LSR.L   #1,D1
143-Bug>
```

Verify S-Records Against Memory

VE

VE [*n*] [*addr*] [;X|-C][=*text*]

3

The VE command is identical to the LO command with the exception that data is not stored to memory but merely compared to the contents of memory.

This command accepts serial data from a host system in the form of a file of Motorola S-records and compares it to data already in the MVME143 memory. If the data does not compare, then the user is alerted via information sent to the terminal screen.

The optional port number *n* allows the user to specify which port is to be used for the downloading. if this number is omitted, port 1 is assumed.

The optional *addr* field allow the user to enter an offset address which is to be added to the address contained in the address field of each record. this causes the records to be compared to memory at different locations than would normally occur. The contents of the automatic offset register are not added to the S-record addresses. (Appendix C has information on S-records.) If the address is in the range $0 to $1F and the port number is omitted, precede the address with a comma to distinguish it from a port number.

The optional *text* field, entered after the equal sign (=), is sent to the host before 143Bug begins to look for S-records at the host port. This allows the user to send a command to the host device to initiate the download. This text should NOT be delimited by any kind o quote marks. Text is understood to begins immediately following the equals sign and terminate with the carriage return. If the host is operating full duplex, the string is also echoed back to the host port by the host and appears on the user's terminal screen.

In order to accommodate host systems that echo all received characters, the above-mentioned text string is sent to the host one character at a time and characters received from the host are read one at a time. After the entire command has been sent to the host, VE keeps looking for an LF character from the host, signifying the end of the echoed command. No data records are processed until this LF is received. If the host system does not echo characters, VE still keeps looking for an LF character before data records are processed. For this reason, it is required in situations where the host system does not echo characters, that the first record transferred by the host system be a header record. The header record is not used, but the LF after the header record serves to break VE out of the loop so that data records are processed.

The other options have the following effects:

-C option - Ignore checksum. A checksum for the data contained within an S-record is calculated as the S-record is read in at the port. Normally, this calculated checksum is compared to the checksum contained within the S-record and if the compare fails, an error message is sent to the screen on completion of the download. If this option is selected, then the comparison is not made.

X option - Echo. This option echoes the S-records to the user's terminal as they are read in at the host port.

During a verify operation, data from an S-record is compared to memory beginning with the address contained in the S-record address field (plus the offset address, if it was specified). If the verification fails, then the non-comparing record is set aside until the verify is complete and then it is printed out to the screen. If three non-comparing records are encountered in the course of a verify operation, then the command is aborted.

If a non-hex character is encountered within the data field of a data record, then the part of the record which had been received up to that time is printed to the screen and the 143Bug error handler is invoked to point to the faulty character.

If the embedded checksum of a record does not agree with the checksum calculated by 143Bug AND if the checksum comparison has not been disabled via the "-C" option, then an error condition exists. A message is output stating the address of the record (as obtained from the address field of the record), the calculated checksum, and the checksum read with the record. A copy of the record is also output. This is a fatal error and causes the command to abort.

Examples:

This short program was developed on a host system.

```
1                           *  Test Program.
2                           *
3              65040000                ORG       $65040000
4
5       65040000 7001                  MOVEQ.L   #1,D0
6       65040002 D088                  ADD.L     A0,D0
7       65040004 4A00                  TST.B     D0
8       65040006 4E75                  RTS
9                                      END

******  TOTAL ERRORS       0--
******  TOTAL WARNINGS     0--
```

Then the program was converted into an S-record file named TEST.MX that looks like this:

```
S00F000054455354533353372020001015E
S30D650400007001D0884A004E75B3
S7056504000091
```

This file was downloaded into memory at address $40000. The program may be examined in memory using the Memory Display (MD) command.

```
143-Bug>MD 40000:4;DI
00040000 7001                  MOVEQ.L #1,D0
00040002 D088                  ADD.L   A0,D0
00040004 4A00                  TST.B   D0
00040006 4E75                  RTS
143-Bug>
```

Suppose that the user wants to make sure that the program has not been destroyed in memory. The VE command is used to perform a verification.

```
143-Bug>VE -65000000;X=COPY TEST.MX,#
S00F000054455354533353372020001015E
S30D650400007001D0884A004E75B3
S7056504000091
Verify passes.
143-Bug>
```

The verification passes. The program stored in memory was the same as that in the S-record file that had been downloaded.

Now change the program in memory and perform the verification again.

```
143-Bug>M 40002
00040002 D088? D089.

143-Bug>VE -65000000;X=COPY TEST.MX,#
S00F000054455354533353372020010 15E
S30D650400007001D0884A004E75B3
S7056504000091

The following record(s) did not verify .....
S30D65040000------88--------B3

143-Bug>
```

The byte which was changed in memory does not compare with the corresponding byte in the S-record.

CHAPTER 4
USING THE ONE-LINE ASSEMBLER/DISASSEMBLER

Introduction

Included as part of the 143Bug firmware is an assembler/disassembler function. The assembler is an interactive assembler/editor in which the source program is not saved. Each source line is translated into the proper MC68030/MC68882 machine language code and is stored in memory on a line-by-line basis at the time of entry. In order to display an instruction, the machine code is disassembled, and the instruction mnemonic and operands are displayed. All valid MC68030 instructions are translated.

The 143Bug assembler is effectively a subset of the MC68030 resident structured assembler. It has some limitations as compared with the resident assembler, such as not allowing line numbers and labels; however, it is a powerful tool for creating, modifying, and debugging MC68030 code.

MC68030 Assembly Language

The symbolic language used to code source programs for processing by the assembler is MC68030 assembly language. This language is a collection of mnemonics representing:

- Operations
  - MC68030 machine-instruction operation codes
  - Directives (pseudo-ops)
- Operators
- Special symbols

Machine-Instruction Operation Codes

That part of the assembly language that provides the mnemonic machine-instruction operation codes for the MC68030/MC68882 machine instructions is described in the *MC68030 32-Bit Microprocessor User's Manual* and the *MC68882 Floating-Point Coprocessor User's Manual* , MC68030UM and MC68881UM. Refer to these manuals for any question concerning operation codes.

Directives

Normally, assembly language can contain mnemonic directives which specify auxiliary actions to be performed by the assembler.

The 143Bug assembler recognizes only two directives called define constant (DC.W) and SYSCALL. These two directives are used to define data within the program and to make calls to 143Bug utilities. Refer to the *DC.W Define Constant Directive* and *SYSCALL System Call Directive* paragraphs in this chapter.

Comparison With MC68030 Resident Structured Assembler

There are several major differences between the 143Bug assembler and the MC68030 resident structured assembler. The resident assembler is a two-pass assembler that processes an entire program as a unit, while the 143Bug assembler processes each line of a program as an individual unit. Due mainly to this basic functional difference, the capabilities of the 143Bug assembler are more restricted:

a. Label and line numbers are not used. Labels are used to reference other lines and locations in a program. The one-line assembler has no knowledge of other lines and, therefore, cannot make the required association between a label and the label definition located on a separate line.

b. Source lines are not saved. In order to read back a program after it has been entered, the machine code is disassembled and then displayed as mnemonic and operands.

c. Only two directives (DC.W and SYSCALL) are accepted.

d. No macro operation capability is included.

e. No conditional assembly is used.

f. Several symbols recognized by the resident assembler are not included in the 143Bug assembler character set. These symbols include > and <. Three other symbols have multiple meaning to the resident assembler, depending on the context (refer to the *Addressing Modes* paragraph in this chapter). These are:

Asterisk (*) -- Multiply *or* current PC.
Slash (/) -- Divide *or* delimiter in a register list.
Ampersand (&) -- AND *or* decimal number.

Although functional differences exist between the two assemblers, the one-line assembler is a true subset of the resident assembler. The format and syntax used with the 143Bug assembler are acceptable to the resident assembler except as described above.

Source Program Coding

A source program is a sequence of source statements arranged in a logical way to perform a predetermined task. Each source statement occupies a line and must be either an executable instruction, a DC.W directive, or a SYSCALL assembler directive. Each source statement follows a consistent source line format.

Source Line Format

Each source statement is a combination of operation and, as required, operand fields. Line numbers, labels, and comments are not used.

Operation Field

Because there is no label field, the operation field may begin in the first available column. It may also follow one or more spaces. Entries can consist of one of three categories:

a. Operation codes which correspond to the MC68030/MC68882 instruction set.

b. Define Constant directive: DC.W is recognized to define a constant in a word location.

c. System Call directive: SYSCALL is used to call 143Bug system utilities.

The size of the data field affected by an instruction is determined by the data size codes. Some instructions and directives can operate on more than one data size. For these operations, the data size code must be specified or a default size applicable to that instruction will be assumed. The size code need not be specified if only one data size is permitted by the operation. The data size code is specified by a period (.), appended to the operation field, and followed by B, W, or L, where:

B = Byte (8-bit data).
W = Word (the usual default size; 16-bit data).
L = Longword (32-bit data).

The data size code is not permitted, however, when the instruction or directive does not have a data size attribute.

Examples (legal):

LEA	(A0),A1	Longword size is assumed (.B, .W not allowed); this instruction loads effective address of the first operand into A1.
ADD.B	(A0),D0	This instruction adds the byte whose address is (A0) to the lowest order byte in D0.
ADD	D1,D2	This instruction adds the low order word of D1 to the low order word of D2. (W is the default size code.)
ADD.L	A3,D3	This instruction adds the entire 32-bit (longword) contents of A3 to D3.

Example (illegal):

SUBA.B	#5,A1	Illegal size specification (.B not allowed on SUBA). This instruction would have subtracted the value 5 from the low order byte of A1; byte operations on address registers are not allowed.

Operand Field

If present, the operand field follows the operation field and is separated from the operation field by at least one space. When two or more operand subfields appear within a statement, they must be separated by a comma. In an instruction like ' ADD D1,D2', the first subfield (D1) is called the source effective address field, and the second subfield (D2) is called the destination effective address field. Thus, the contents of D1 are added to the contents of D2 and the result is saved in register D2. In the instruction 'MOVE D1,D2', the first subfield (D1) is the sending field and the second subfield (D2) is the receiving field. In other words, for most two-operand instructions, the general format '*opcode source, destination*' applies.

Disassembled Source Line

The disassembled source line may not look identical to the source line entered. The disassembler makes a decision on how it interprets the numbers used. If the number is an offset off of an address register, it is treated as a signed hexadecimal offset. Otherwise, it is treated as a straight unsigned hexadecimal. For example,

```
MOVE.L  #1234,5678
MOVE.L  FFFFFFFC(A0),5678
```

disassembles to:

```
0000300021FC0000    12345678  MOVE.L    #$1234,($5678).W
0000300821E8FFFC    5678      MOVE.L    -$4(A0),($5678).W
```

Also, for some instructions, there are two valid mnemonics for the same opcode, or there is more than one assembly language equivalent. The disassembler may choose a form different from the one originally entered. As examples:

a. BRA is returned for BT

b. DBF is returned for DBRA

NOTE

The assembler recognizes two forms of mnemonics for two branch instructions. The BT form (branch conditionally true) has the same opcode as the BRA instruction. Also, DBRA (decrement and branch always) and DBF (never true, decrement, and branch) mnemonics are different forms for the same instruction. In each case, the assembler will accept both forms.

Mnemonics And Delimiters

The assembler recognizes all MC68030 instruction mnemonics. Numbers are recognized as binary, octal, decimal, and hexadecimal, with hexadecimal the default case.

a. Decimal – is a string of decimal digits (0 through 9) preceded by an ampersand (&). Examples are: &12334, -&987654321 .

b. Hexadecimal – is a string of hexadecimal digits (0 through 9, A through F) preceded by an optional dollar sign ($). An example is: $AFE5 .

One or more ASCII characters enclosed by apostrophes (' ') constitute an ASCII string. ASCII strings are right–justified and zero–filled (if necessary), whether stored or used as immediate operands.

```
0000300021FC0000    12345678  MOVE.L    #$1234,(5678).W
005000      0053              DC.W 'S'
005002      223C41424344      MOVE.L    #'ABCD',D1
005008      3536              DC.W '56'
```

The assembler/disassembler recognizes/references these register mnemonics:

Pseudo Registers	
R0–R7	**USER OFFSET REGISTERS**
Main Processor Registers	
PC	Program Counter. Used only in forcing PC-relative addressing.
SR	Status Register.
CCR	Condition Codes Register (Lower eight bits of SR).
USP	User Stack Pointer.
MSP	Master Stack Pointer.
ISP	Interrupt Stack Pointer. VBR Vector Base Register.
SFC	Source Function Code Register.
DFC	Destination Function Code Register.
CACR	Cache Control Register.
CAAR	Cache Address Register.
D0–D7	Data registers.
A0–A7	Address Registers. A7 represents the active system stack pointer, (one of USP, MSP, or ISP), as specified by M and S bits of status register (SR).
MEMORY MANAGEMENT UNIT REGISTERS	
MMUSR	Status Register
CRP	CPU Root Pointer
SRP	Supervisor Root Pointer
TC	Translation Control Register
TT0	Transparent Translation 0
TT1	Transparent Translation 1
FLOATING POINT COPROCESSOR REGISTERS	
FPCR	Control Register
FPSR	Status Register
FPIAR	Instruction Address Register
FP0–FP7	Floating Point Data Registers

Character Set

The character set recognized by the 143Bug assembler is a subset of ASCII, and these are listed as follows:

a. The letters A through Z (uppercase and lowercase)

b. The integers 0 through 9

c. Arithmetic operators: + - * / << >> ! &

d. Parentheses ( )

e. Characters used as special prefixes:

(pound sign) specifies the immediate form of addressing.
$ (dollar sign) specifies a hexadecimal number.
& (ampersand) specifies a decimal number.
@ (commercial at sign) specifies an octal number.
% (percent sign) specifies a binary number.
' (apostrophe) specifies an ASCII literal character string.

f. Five separating characters:

Space
, (comma)
. (period)
/ (slash)
- (dash)

g. The character * (asterisk) indicates current location.

Addressing Modes

Effective addressing modes, combined with operation codes, define the particular function to be performed by a given instruction. Effective addressing and data organization are described in detail in Section 2, *Data Organization and Addressing Capabilities*, of the *MC68030 32-Bit Microprocessor User's Manual*.

The addressing modes of the MC68030 which are accepted by the 143Bug one-line assembler are summarized in Table 4-1.

TABLE 4-1. 143Bug Assembler Addressing Modes

FORMAT	DESCRIPTION
Dn	Data register direct.
An	Address register direct.
(An)	Address register indirect.
(An)+	Address register indirect with postincrement.
-(An)	Address register indirect with predecrement.
d(An)	Address register indirect with displacement.
d(An,Xi)	Address register indirect with index, 8-bit displacement.
(bd,An,Xi)	Address register indirect with index, base displacement.
([bd,An],Xi,od)	Address register memory indirect postindexed.
([bd,An,Xi],od)	Address register memory indirect pre-indexed.
(d16,PC)	Program Counter indirect with displacement.
(d8,PC,Xi)	Program Counter indirect with index, 8-bit displacement.
(bd,PC,Xi)	Program Counter indirect with index, base displacement.
([bd,PC],Xi,od)	Program Counter memory indirect postindexed.
([bd,PC,Xi],od)	Program Counter memory indirect pre-indexed.
(xxxx).W	Absolute word address.
(xxxx).L	Absolute long address.
#xxxx	Immediate data.

The user may use an expression in any numeric field of these addressing modes. The assembler has a built-in expression evaluator. It supports the following operands types:

a. Binary numbers (%10)

b. Octal numbers (@765..0)

c. Decimal numbers (&987..0)

d. Hexadecimal numbers ($FED..0)

e. String literals ('CHAR')

f. Offset registers (R0-R7)

g. Program counter (*)

Allowed operators are:

a. Addition +

b. Subtraction -

c. Multiply *

d. Divide /

e. Shift left <<

f. Shift right >>

g. Bitwise OR !

h. Bitwise AND &

The order of evaluation is strictly left to right with no precedence granted to some operators over others. The only exception to this is when the user forces the order of precedence through the use of parentheses.

Possible points of confusion:

a. The user should keep in mind that where a number is intended and it could be confused with a register, it must be differentiated in some way. For example:

```
CLR     D0      means CLR.W register D0. On the other hand,

CLR     $D0
CLR     0D0
CLR     +D0
CLR     D0+0    all mean CLR.W memory location $D0.
```

b. With the use of '*' to represent both multiply and program counter, how does the assembler know when to use which definition?

   For parsing algebraic expressions, the order of parsing is:

   operand operator operand operator. . .

   with a possible left or right parenthesis.

Given the above order, the assembler can distinguish by placement which definition to use. For example:

```
***     means PC   x   PC
*+*     means PC   +   PC
2**     means  2   *   PC
*&&16   means PC   AND &16
```


When specifying operands, the user may skip or omit entries with the following addressing modes.

a. Address register indirect with index, base displacement.

b. Address register memory indirect postindexed.

c. Address register memory indirect pre-indexed.

d. Program counter indirect with index, base displacement.

e. Program counter memory indirect postindexed.

f. Program counter memory indirect pre-indexed.

For modes address register/program counter indirect with index, base displacement, the rules for omission/skipping are as follows:

a. The user may terminate the operand at any time by specifying ')'. Example:

```
CLR     ()          or
CLR     (,,)        is equivalent to

CLR     (0.N,ZA0,ZD0.W*1)
```

b. The user may skip a field by "stepping past" it with a comma. Example:

```
CLR     (D7)        is equivalent to

CLR     ($D7,ZA0,ZD0.W*1)
```

but

```
CLR     (,,D7)      is equivalent to

CLR     (0.N,ZA0,D7.W*1)
```

c. If the user does not specify the base register, the default 'ZA0' is forced.

d. If the user does not specify the index register, the default 'ZD0.W*1' is forced.

e. Any unspecified displacements are defaulted to '0.N'.

The rules for parsing the memory indirect addressing modes are the same as above with the following additions.

a. The subfield that begins with '[' must be terminated with a matching ']'.

b. If the text given is insufficient to distinguish between the pre-indexed or postindexed addressing modes, the default is the pre-indexed form.

DC.W Define Constant Directive

The format for the DC.W directive is: DC.W *operand*

The function of this directive is to define a constant in memory. The DC.W directive can have only one operand (16-bit value) which can contain the actual value (decimal, hexadecimal, or ASCII). Alternatively, the operand can be an expression which can be assigned a numeric value by the assembler. The constant is aligned on a word boundary as word (.W) size is specified. An ASCII string is recognized when characters are enclosed inside single quotes (' '). Each character (seven bits) is assigned to a byte of memory, with the eighth bit (MSB) always equal to zero. If only one byte is entered, the byte is right justified. A maximum of two ASCII characters may be entered for each DC.W directive. Examples are:

```
00010022    04D2    DC.W    &1234     Decimal number
00010024    AAFE    DC.W    AAFE      Hexadecimal number
00010026    4142    DC.W    'AB'      ASCII string
00010028    5443    DC.W    'TB'+1    Expression
0001002A    0043    DC.W    'C'       ASCII character is right justified
```

SYSCALL System Call Directive

The function of this directive is to aid the user in making the TRAP #15 calls to system functions. The format for this directive is:

SYSCALL *function name*

For example, the following two pieces of code produce identical results.

```
TRAP #$F
DC.W 0
```

or

```
SYSCALL .INCHR
```

Refer to Chapter 5, *SYSTEM CALLS*, for a complete listing of all the functions provided.

Entering And Modifying Source Programs

User programs are entered into the memory using the one-line assembler/ disassembler. The program is entered in assembly language statements on a line-by-line basis. The source code is not saved as it is converted immediately to machine code upon entry. This imposes several restrictions on the type of source line that can be entered.

Symbols and labels, other than the defined instruction mnemonics, are not allowed. The assembler has no means to store the associated values of the symbols and labels in lookup tables. This forces the programmer to use memory addresses and to enter data directly rather than use labels.

Also, editing is accomplished by retyping the entire new source line. Lines can be added or deleted by moving a block of memory data to free up or delete the appropriate number of locations (refer to the *BM Command* paragraph in Chapter 3).

Invoking The Assembler/Disassembler

The assembler/disassembler is invoked using the ;DI option of the Memory Modify (MM) and Memory Display (MD) commands:

MM *addr* **;DI**

where	**(CR)**	sequences to next instruction
	.(CR)	exits command

and

MD[S] *addr*[:*count* | *addr*]**;DI**

The MM (;DI option) is used for program entry and modification. When this command is used, the memory contents at the specified location are disassembled and displayed. A new or modified line can be entered if desired.

The disassembled line can be an MC68030 instruction, a SYSCALL, or a DC.W directive. If the disassembler recognizes a valid form of some instruction, the instruction will be returned; if not (random data occurs), the DC.W $XXXX (always hex) is returned. Because the disassembler gives precedence to instructions, a word of data that corresponds to a valid instruction will be returned as the instruction.

Entering A Source Line

A new source line is entered immediately following the disassembled line, using the format discussed in the *Source Line Format* paragraph in this chapter:

```
143-Bug>MM 10000;DI
00010000  2600                    MOVE.L  D0,D3 ?  ADDQ.L #1,A3
```

When the carriage return is entered terminating the line, the old source line is erased from the terminal screen, the new line is assembled and displayed, and the next instruction in memory is disassembled and displayed:

```
143-Bug>MM 10000;DI
00010000  528B                    ADDQ.L  #1,A3
00010002  4282                    CLR.L   D2  ?
```

If a hardcopy terminal is being used, the above example will look as follows:

```
143-Bug>MM 10000;DI
00010000  2600                    MOVE.L  D0,D3 ?  ADDQ.L #1,A3
00010000  528B                    ADDQ.L  #1,A3
00010002  4282                    CLR.L   D2  ?
```

Another program line can now be entered. Program entry continues in like manner until all lines have been entered. A period is used to exit the MM command.

If an error is encountered during assembly of the new line, the assembler displays the line unassembled with a "^" under the field suspected of causing the error and an error message is displayed. The location being accessed is redisplayed:

```
143-Bug>m 10000;di
00010000 528B                          ADDQ.L   #1,A3 ? lea.l 5(a0,d8),a4
00010000                               LEA.L    5(A0,D8),A4
-----------------------------------------------------^
*** Unknown Field ***
00010000 528B                          ADDQ.L   #1,A3 ?
```

Entering Branch And Jump Addresses

When entering a source line containing a branch instruction (BRA, BGT, BEQ, etc.), do not enter the offset to the branch destination in the operand field of the instruction. The offset is calculated by the assembler. The user must append the appropriate size extension to the branch instruction.

To reference a current location in an operand expression, the character "*" (asterisk) can be used. Examples are:

```
00030000   60004094                    BRA *+$4096
00030000   60FE                        BRA.B *
00030000   4EF90003 0000                JMP *
00030000   4EF00130 00030000           JMP (*,A0,D0)
```

In the case of forward branches or jumps, the absolute address of the destination may not be known as the program is being entered. The user may temporarily enter an "*" for branch to self in order to reserve space. After the actual address is discovered, the line containing the branch instruction can be re-entered using the correct value.

NOTE

Branch sizes must be entered as ".B" or ".W" as opposed to ".S" and ".L".

Assembler Output/Program Listings

A listing of the program is obtained using the Memory Display (MD) command with the ;DI option. The MD command requires both the starting address and the line count to be entered in the command line. When the ;DI option is invoked, the number of instructions disassembled and displayed is equal to the line count.

To obtain a hard copy listing of a program, use the Printer Attach (PA) command to activate the printer port. An MD to the terminal then causes a listing on the terminal and on the printer.

Note again, that the listing may not correspond exactly to the program as entered. As discussed in the *Disassembled Source Line* paragraph in this chapter, the disassembler displays in signed hexadecimal any number it interprets as an offset off of an address register; all other numbers are displayed in unsigned hexadecimal.

CHAPTER 5
SYSTEM CALLS

Introduction

This chapter describes the 143Bug TRAP #15 handler, which allows system calls from user programs. The system calls can be used to access selected functional routines contained within 143Bug, including input and output routines. TRAP #15 may also be used to transfer control to 143Bug at the end of a user program (refer to the *.RETURN Function* paragraph in this chapter).

In the descriptions of some input and output functions, reference is made to the "default input port" or the "default output port". After power-up or reset, the default input and output port is initialized to be port 0 (the MVME143 serial port 1). The defaults may be changed, however, using the .REDIR_I and .REDIR_O functions, as described in this chapter.

Invoking System Calls Through TRAP #15

To invoke a system call from a user program, simply insert a TRAP #15 instruction into the source program. The code corresponding to the particular system routine is specified in the word following the TRAP opcode, as shown in the following example.

Format in user program:

```
TRAP #15     System call to 143Bug
DC.W $xxxx   Routine being requested (xxxx = code)
```

In some of the examples shown in the following descriptions, a SYSCALL macro is used. This macro automatically assembles the TRAP #15 call followed by the Define Constant for the function code. For clarity, the SYSCALL macro is as follows:

```
SYSCALL     MACRO
            TRAP     #15
            DC.W     \1
            ENDM
```

Using the SYSCALL macro, the system call would appear in the user program as follows:

```
SYSCALL     routine name
```

It is, of course, necessary to create an equate file with the routine names equated to their respective codes.

When using the 143Bug one-line assembler/disassembler, the SYSCALL macro and the equates are predefined. Simply write in "SYSCALL" followed by a space and the function, then carriage return.

Example:

```
143-Bug>M 3000;DI
0000 3000 00000000          ORI.B #$0,D0? SYSCALL .OUTLN
0000 3000 4E4F0022          SYSCALL .OUTLN
0000 3004 00000000          ORI.B #$0,D0? .
143-Bug>
```

5

String Formats For I/O

Within the context of the TRAP #15 handler there are two formats for strings:

Pointer/Pointer Format - The string is defined by a pointer to the first character and a pointer to the last character + 1.

Pointer/Count Format - The string is defined by a pointer to a count byte, which contains the count of characters in the string, followed by the string itself.

A line is defined as a string followed by a carriage return and a line feed: (CR)(LF).

SYSTEM CALL ROUTINES

The TRAP #15 functions are summarized in Table 5-1. Refer to the write-ups on the utilities for specific use information.

TABLE 5-1. 143Bug System Call Routines

CODE	FUNCTION	DESCRIPTION	
$0000	.INCHR	Input character	
$0001	.INSTAT	Input serial port status	
$0002	.INLN	Input line	(pointer/pointer format)
$0003	.READSTR	Input string	(pointer/count format)
$0004	.READLN	Input line	(pointer/count format)
$0005	.CHKBRK	Check for break	
$0010	.DSKRD	Disk read	
$0011	.DSKWR	Disk write	
$0012	.DSKCFIG	Disk configure	
$0014	.DSKFMT	Disk format	
$0015	.DSKCTRL	Disk control	
$0020	.OUTCHR	Output character	
$0021	.OUTSTR	Output string	(pointer/pointer format)
$0022	.OUTLN	Output line	(pointer/pointer format)
$0023	.WRITE	Output string	(pointer/count format)
$0024	.WRITELN	Output line	(pointer/count format)
$0025	.WRITDLN	Output line with data	(pointer/count format)
$0026	.PCRLF	Output carriage return and line feed	
$0027	.ERASLN	Erase line	
$0028	.WRITD	Output string with data	(pointer/count format)
$0029	.SNDBRK	Send break	
$0043	.DELAY	Timer delay function	
$0050	.RTC_TM	Time initialization for RTC	
$0051	.RTC_DT	Date initialization for RTC	
$0052	.RTC_DSP	Display time from RTC	
$0053	.RTC_RD	Read the RTC registers	
$0060	.REDIR	Redirect I/O of a TRAP #15 function	
$0061	.REDIR_I	Redirect input	
$0062	.REDIR_O	Redirect output	
$0063	.RETURN	Return to 143Bug	
$0064	.BINDEC	Convert binary to Binary Coded Decimal (BCD)	
$0067	.CHANGEV	Parse value	
$0068	.STRCMP	Compare two strings	(pointer/count format)
$0069	.MULU32	Multiply two 32-bit unsigned integers	
$006A	.DIVU32	Divide two 32-bit unsigned integers	
$006B	.CHK_SUM	Generate checksum	
$0070	.BRD_ID	Return pointer to board ID packet	

.INCHR Function

TRAP FUNCTION: .INCHR - Input character routine

CODE: $0000

DESCRIPTION: Reads a character from the default input port. The character is returned in the stack.

ENTRY CONDITIONS:

SP ==> Space for character (byte)
Word fill (byte)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Character (byte)
Word fill (byte)

EXAMPLE:

```
SUBQ.L   #2,SP        Allocate space for result.
SYSCALL  .INCHR       Call .INCHR.
MOVE.B   (SP)+,D0     Load character in D0.
```

.INSTAT Function

TRAP FUNCTION: .INSTAT – Input serial port status

CODE: $0001

DESCRIPTION: Used to see if there are characters in the default input port buffer. The condition codes are set to indicate the result of the operation.

ENTRY CONDITIONS:

No arguments or stack allocation required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Z(ero) = 1 if the receiver buffer is empty

EXAMPLE:

```
LOOP    SYSCALL  .INSTAT          Any characters?
        BEQ.S    EMPTY            No, branch
        SUBQ.L   #2,A7            Yes, then
        SYSCALL  .INCHR           Read them
        MOVE.B   (SP)+,(A0)+      In buffer
        BRA.S    LOOP             Check for more
EMPTY
```

.INLN Function

TRAP FUNCTION: .INLN – Input line routine

CODE: $0002

DESCRIPTION: Used to read a line from the default input port. The buffer size should be at least 256 bytes.

ENTRY CONDITIONS:

SP ==> Address of string buffer (longword)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Address of last character in the string+1 (longword)

EXAMPLE: If A0 contains the address where the string is to go;

```
SUBQ.L   #4,A7        Allocate space for result.
PEA.L    (A0)         Push pointer to destination.
TRAP     #15          (May also invoke by SYSCALL
DC.W     2            macro ("SYSCALL .INLN").)
MOVE.L   (A7)+,A1     Retrieve address of last character+1.
```

NOTE: A line is a string of characters terminated by (CR). The maximum allowed size is 254 characters. The terminating (CR) is not considered part of the string, but it is returned in the buffer, that is, the returned pointer points to it. Control character processing as described in the *Terminal Input/Output Control* paragraph in Chapter 1, is in effect.

.READSTR Function

.READSTR

TRAP FUNCTION: .READSTR – Read string into variable-length buffer

CODE: $0003

DESCRIPTION: Used to read a string of characters from the default input port into a buffer. On entry, the first byte in the buffer indicates the maximum number of characters that can be placed in the buffer. The buffer size should at least be equal to that number+2. The maximum number of characters that can be placed in a buffer is 254 characters. On exit, the count byte indicates the number of characters in the buffer. Input terminates when a (CR) is received. The (CR) character appears in the buffer, although it is not included in the string count. All printable characters are echoed to the default output port. The (CR) is not echoed. Some control character processing is done:

^G	Bell	Echoed.
^X	Cancel line	Line is erased.
^H	Backspace	Last character is erased.
(DEL)	Same as backspace	Last character is erased.
(LF)	Line Feed	Echoed.
(CR)	Carriage Return	Terminates input.

All other control characters are ignored.

ENTRY CONDITIONS:

SP ==> Address of input buffer (longword)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
The count byte contains the number of bytes in the buffer.

EXAMPLE: If A0 contains the string buffer address:

```
MOVE.B    #75,(A0)      Set maximum string size.
PEA.L     (A0)          Push buffer address.
TRAP      #15           (May also invoke by SYSCALL
DC.W      3             macro ("SYSCALL .READSTR").)
MOVE.B    (A0),D0       Read actual string size.
```

NOTE: This routine allows the caller to dictate the maximum length of input to be less than 254 characters. If more characters are entered, then the buffer input is truncated. Control character processing is described in the *Terminal Input/Output Control* paragraph in Chapter 1.

.READLN Function

TRAP FUNCTION: .READLN - Read line to fixed-length buffer

CODE: $0004

DESCRIPTION: Used to read a string of characters from the default input port. Characters are echoed to the default output port. A string consists of a count byte followed by the characters read from the input. The count byte indicates the number of characters in the input string, excluding (CR)(LF). A string may be up to 254 characters.

ENTRY CONDITIONS:

SP ==> Address of input buffer (longword)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
The first byte in the buffer indicates the string length.

EXAMPLE: If A0 points to a 256-byte buffer;

```
PEA.L    (A0)        Load buffer address
SYSCALL  .READLN     And read a line from default input port.
```

NOTE: The caller must allocate 256 bytes for a buffer. Input may be up to 254 characters. (CR)(LF) is sent to default output following echo of input. Control character processing is described in the *Terminal Input/Output Control* paragraph in Chapter 1.

.CHKBRK Function

.CHKBRK

TRAP FUNCTION: .CHKBRK - Check for break

CODE: $0005

DESCRIPTION: Returns "Zero" status in the condition code register if break status is detected at the default input port.

ENTRY CONDITIONS:

No arguments or stack allocation required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Z flag in CCR if break detected

EXAMPLE:

```
SYSCALL   .CHKBRK
BEQ       BREAK
```

.DSKRD, .DSKWR Functions

.DSKRD
.DSKWR

TRAP FUNCTIONS: .DSKRD – Disk read function
.DSKWR – Disk write function

CODES: $0010
$0011

DESCRIPTION: These functions are used to read and write blocks of data from/to the specified disk or tape device. Information about the data transfer is passed in a command packet which has been built somewhere in memory. (The user program must first manually prepare the packet.) The address of the packet is passed as an argument to the function. The same command packet format is used for .DSKRD and .DSKWR. It is eight words in length and is arranged as follows:

	F E D C B A 9 8	7 6 5 4 3 2 1 0
$00	CONTROLLER LUN	DEVICE LUN
$02	STATUS WORD	
$04	MEMORY ADDRESS	MOST-SIGNIFICANT WORD
$06		LEAST-SIGNIFICANT WORD
$08	BLOCK NUMBER (DISK) OR	MOST-SIGNIFICANT WORD
$0A	FILE NUMBER (STREAMING TAPE)	LEAST-SIGNIFICANT WORD
$0C	NUMBER OF BLOCKS	
$0E	FLAG BYTE	ADDRESS MODIFIER