MVME147BUG 147Bug Debugging Package User's Manual

Part 1 of 2

V147BUGA1/UM1

Preface

The *MVME147Bug -- 147Bug Debugging Package User's Manual* provides general information about the debugger, the debugger command set, use of the one-line assembler/disassembler, system calls, and a diagnostic firmware guide for the 147Bug Debugging Package.

The manual is bound in two parts:

> Part 1 (V147BUGA1/UM1, this volume) contains Chapters 1 through 4.
>
> Part 2 (V147BUGA2/UM1) contains Chapters 5 and 6 and Appendices A through F.
>
> The table of contents and index appear in both volumes.

The manual should be used by anyone who wants general as well as technical information about the 147Bug Debugging Package. A basic knowledge of computers and digital logic is assumed. To use this manual, you should be familiar with the publications listed in the table below.

Related Documentation

The following publications are applicable to the 147Bug debugging package and may provide additional helpful information. If not shipped with this product, they may be purchased by contacting your local Motorola sales office. Non-Motorola documents may be obtained from the sources listed.

Document Title	Motorola Publication Number
MVME147-0*xx* MPU VMEmodule Installation and Use	VME147A/IH
MVME147FW SCSI Firmware User's Manual[2]	MVME147FW/D
MVME147BUG 147Bug Debugging Package User's Manual Parts 1 and 2 (this manual)[2]	V147BUGA1/UM V147BUGA2/UM
MVME147S MPU VMEmodule User's Manual	MVME147S/D
MVME712M Transition Module and P2 Adapter Board Installation and Use	VME712MA/IH
MVME712-12, MVME712-13, MVME712A, MVME712AM, and MVME712B Transition Modules and LCP2 Adapter Board User's Manual	MVME712A/D
MC68030 32-Bit Microprocessor User's Manual	MC68030UM
MC68881/MC68882 Floating-Point Coprocessor User's Manual	MC68881UM
MVME050 System Controller Module User's Manual	MVME050/D

Document Title	Motorola Publication Number
MVME319 Intelligent Disk/Tape Controller User's Manual	MVME319/D
MVME320A VMEbus Disk Controller Module User's Manual	MVME320A/D
MVME320B VMEbus Disk Controller Module User's Manual	MVME320B/D
MVME321 Intelligent Disk Controller User's Manual	MVME321/D
MVME321 IPC Firmware User's Guide	MVME321FW/D
MVME327A VMEbus to SCSI Bus Adapter and MVME717 Transition Module User's Manual	MVME327A/D
MVME350 Streaming Tape Controller VMEmodule User's Manual	MVME350/D
MVME350 IPC Firmware User's Manual	MVME350FW/D
MVME360 SMD Disk Controller User's Manual	MVME360/D

Notes 1. Although not shown in the above list, each Motorola Computer Group manual publication number is suffixed with characters which represent the revision level of the document, such as "/D2" or "/UM2" (the second revision of a manual); a supplement bears the same number as the manual but has a suffix such as "D2A1" or "/UM2A1" (the first supplement to the manual).

2. Manuals shown with a superscript (^{2}) can be ordered as a set with the part number LK-147SET.

The following publications are available from the sources indicated.

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

SCSI Small Computer System Interface; draft X3T9.2/82-2 - Revision 14; Computer and Business Equipment Manufacturers Association, 311 First Street, N. W., Suite 500, Washington D.C. 20001

MK48T02 2K x 8 ZEROPOWER/TIMEKEEPER RAM data sheet; Thompson Components- Mostek Corporation, 1310 Electronics Drive, Carrollton, Texas 75006

WD33C93 SCSI-Bus Interface Controller; WESTERN DIGITAL Corporation, 2445 McCabe Way, Irvine, California 92714

Local Area Network Controller Am7990 (LANCE), Technical Manual, order number 06363A, Advanced Micro Devices, Inc., 901 Thompson Place, P.O Box 3453, Sunnyvale, CA 94088.

Safety Summary
Safety Depends On You

The following general safety precautions must be observed during all phases of operation, service, and repair of this equipment. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the equipment. Motorola, Inc. assumes no liability for the customer's failure to comply with these requirements.

The safety precautions listed below represent warnings of certain dangers of which Motorola is aware. You, as the user of the product, should follow these warnings and all other safety precautions necessary for the safe operation of the equipment in your operating environment.

Ground the Instrument.

To minimize shock hazard, the equipment chassis and enclosure must be connected to an electrical ground. The equipment is supplied with a three-conductor ac power cable. The power cable must be plugged into an approved three-contact electrical outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate in an Explosive Atmosphere.

Do not operate the equipment in the presence of flammable gases or fumes. Operation of any electrical equipment in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits.

Operating personnel must not remove equipment covers. Only Factory Authorized Service Personnel or other qualified maintenance personnel may remove equipment covers for internal subassembly or component replacement or any internal adjustment. Do not replace components with power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service or Adjust Alone.

Do not attempt internal service or adjustment unless another person capable of rendering first aid and resuscitation is present.

Use Caution When Exposing or Handling the CRT.

Breakage of the Cathode-Ray Tube (CRT) causes a high-velocity scattering of glass fragments (implosion). To prevent CRT implosion, avoid rough handling or jarring of the equipment. Handling of the CRT should be done only by qualified maintenance personnel using approved safety mask and gloves.

Do Not Substitute Parts or Modify Equipment.

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the equipment. Contact your local Motorola representative for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings.

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed. You should also employ all other safety precautions which you deem necessary for the operation of the equipment in your operating environment.

WARNING

Dangerous voltages, capable of causing death, are present in this equipment. Use extreme caution when handling, testing, and adjusting.

Manual Terminology

Throughout this manual, a convention has been maintained whereby data and address parameters are preceded by a character which specifies the numeric format as follows:

$	dollar	specifies a hexadecimal number
%	percent	specifies a binary number
&	ampersand	specifies a decimal number

Unless otherwise specified, all address references are in hexadecimal.

An asterisk (*) following the signal name for signals which are *edge significant* denotes that the actions initiated by that signal occur on high to low transition.

In this manual, *assertion* and *negation* are used to specify forcing a signal to a particular state. In particular, *assertion* and *assert* refer to a signal that is active or true; *negation* and *negate* indicate a signal that is inactive or false. These terms are used independently of the voltage level (high or low) that they represent.

Printed in the United States of America
March 1997

Contents

List of Figures

List of Tables

General Information 1

Description of 147Bug

The MVME147Bug (147Bug) package is a powerful evaluation and debugging package for systems built around the MVME147 monoboard microcomputer. It contains facilities for loading and executing user programs under complete operator control for system evaluation. 147Bug includes:

- ❑ Commands for display and modification of memory
- ❑ Breakpoint and tracing capabilities
- ❑ A powerful assembler/disassembler useful for patching programs
- ❑ A self-test at power-up feature that verifies the integrity of the system
- ❑ Various 147Bug routines that handle I/O, data conversion, and string functions available to user programs through the TRAP #15 system calls

Caution

When using a 147Bug TRAP #15 function, the interrupt mask is raised to level 7 and the MMU is disabled during the TRAP #15 operation.

Optional “system” mode that allows autoboot on power-up or reset, and a menu interface to several system commands used in VME Delta Series systems.

The 147Bug consists of three parts:

- ❑ A command-driven, user-interactive software debugger, described in Chapter 2 and hereafter referred to as the *debugger* or *147Bug*
- ❑ A command-driven diagnostic package for the MVME147 hardware, described in Chapter 6 and hereafter referred to as the *diagnostics*

- ❏ A user interface that accepts commands from the system console terminal

When using 147Bug, you operate in either of two directories:

- ❏ **The debugger directory.** In the debugger directory, the debugger prompt `147-Bug>` is displayed and you have all the debugger commands at your disposal.
- ❏ **The diagnostic directory.** In the diagnostic directory, the diagnostic prompt `147-Diag>` is displayed and you have all the diagnostic commands at your disposal as well as all of the debugger commands.

You may examine the commands in the current directory by using the Help (**HE**) command (refer to Chapter 3). You may switch between directories by using the Switch Directories (**SD**) command

Because 147Bug is command-driven, it performs its various operations in response to commands you enter at the keyboard. When you enter a command, 147Bug executes the command and again displays its prompt, except that when you enter a command that causes execution of your target code (for example, **GO**), then control may or may not return to 147Bug, depending on the outcome of the program.

The flow of control in normal 147Bug operation is illustrated in Figure 1-1. The flow of control in “system” mode is illustrated in Figure 1-2.

The 147Bug commands are flexible, powerful, and “user-friendly”, with detailed error messages (refer to Appendix B) and an online help facility.

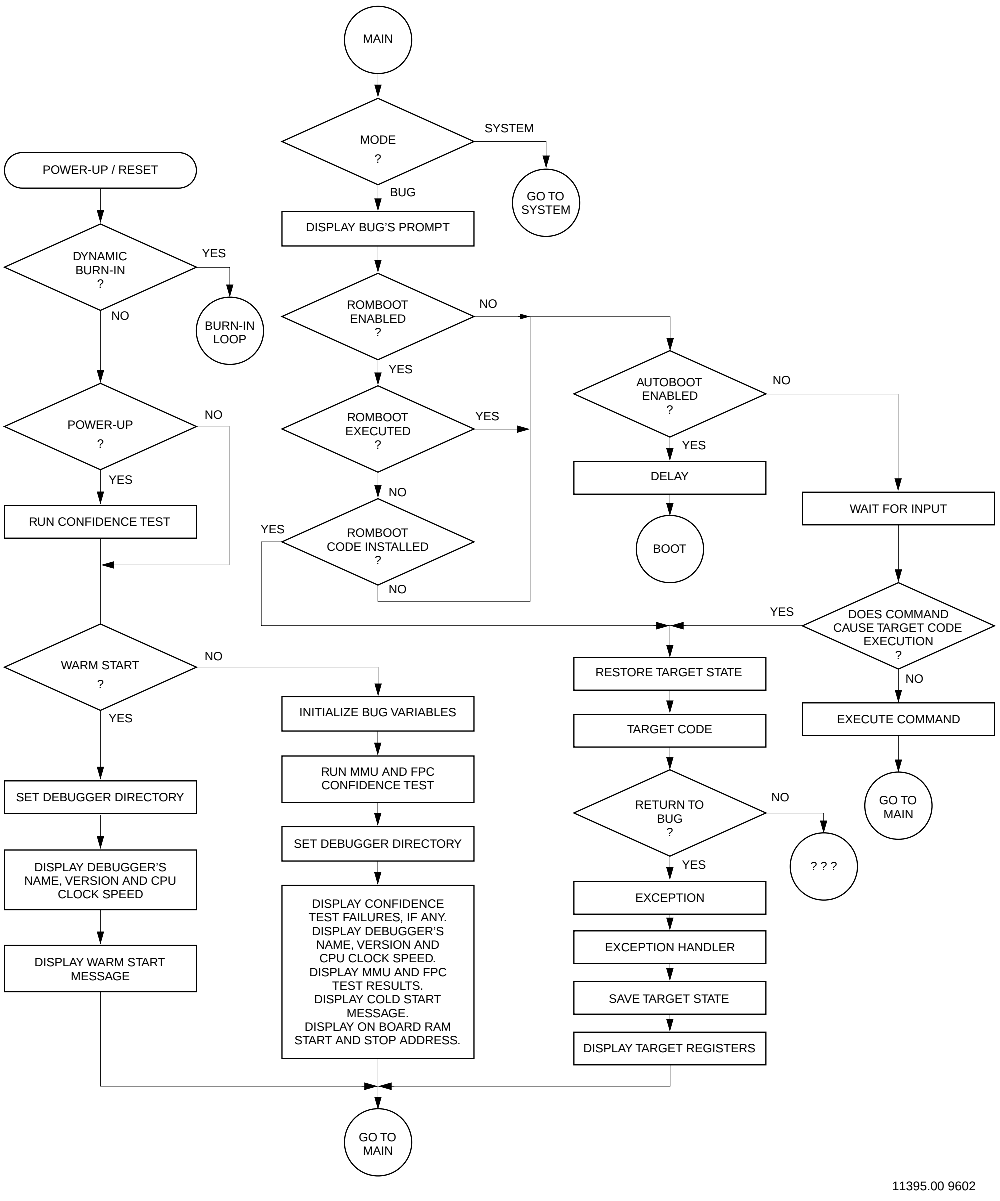


Figure 1-1. Flow Diagram of 147Bug Normal Operational Mode

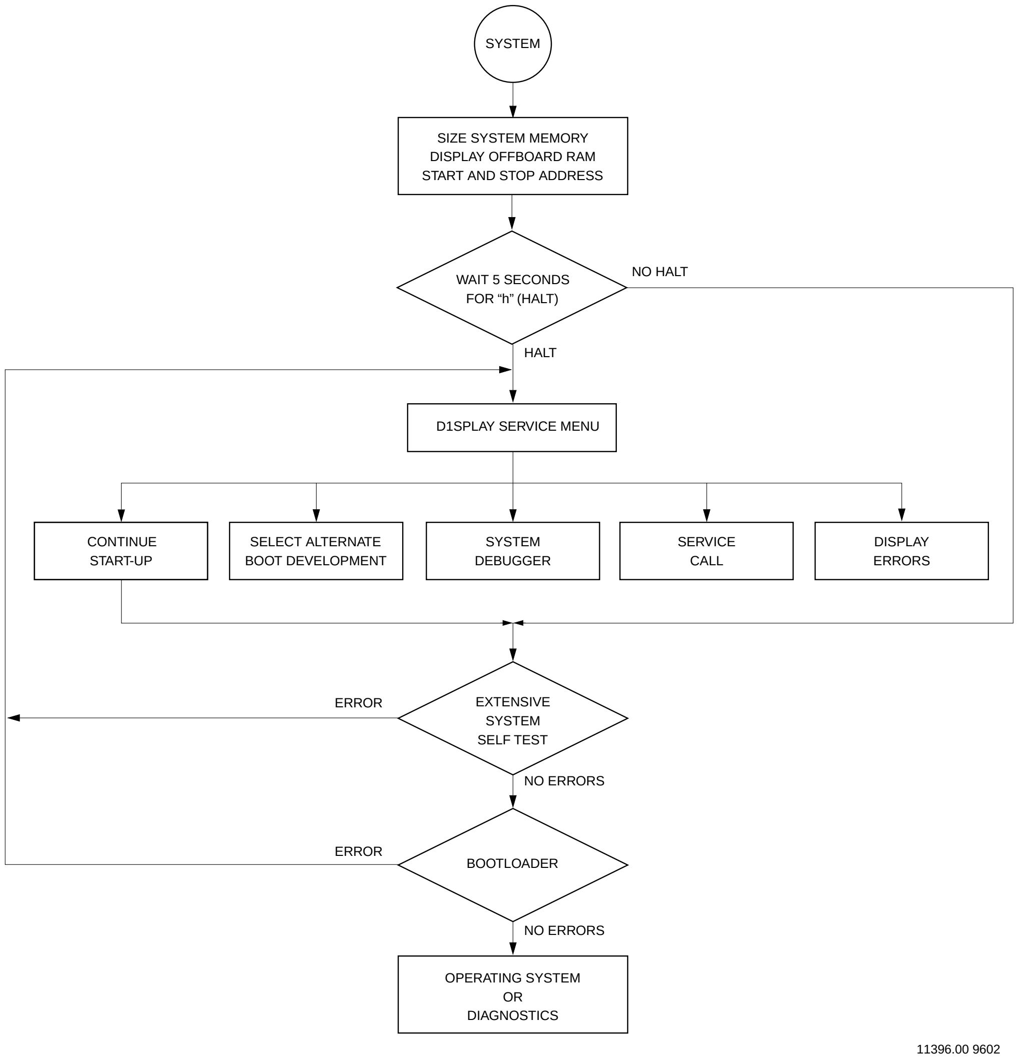


Figure 1-2. Flow Diagram of 147Bug System Operational Mode

How to Use This Manual

If you have never used a debugging package before you should read all of Chapter 1 before attempting to use 147Bug. This gives an overview of 147Bug structure and capabilities.

The *Installation and Start-up* section describes a step-by-step procedure to power up the module and obtain the 147Bug prompt on the terminal screen.

For a question about syntax or operation of a particular 147Bug command, you 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 you already understand how the assembler/ disassembler works. Refer to the assembler/disassembler description in Chapter 4 for details on its use.

Note In the examples shown, all your input is in **BOLD**. This is done for clarity in understanding the examples (to distinguish between characters input by you and characters output by 147Bug). The symbol (**CR**) represents the “carriage return” (**Return** or **Enter)** key on the terminal keyboard.

Installation and Start-up

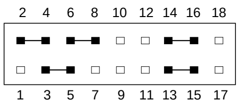
Even though the MVME147Bug EPROMs are installed on the MVME147 module, for 147Bug to operate properly with the MVME147, follow this set-up procedure. Refer to the *MVME147-0xx MPU VMEmodule Installation and Use* manual for header and parts locations.

Note The jumpering instructions that follow apply only to MVME147modules with a suffix-01*x* or -02*x*. If you have earlier boards, consult the MVME147 user’s manual that was furnished with your board.

Caution

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

1. Turn all equipment power OFF. Configure the jumper headers J2 and J3 on the module as required for your particular application.

Header	Jumper Configuration
J2	Header J2 must be configured with jumpers positioned between pins 2-4, 3-5, 6-8, 13-15, and 14-16 as shown. This sets EPROM sockets U22 and U30 for 128K x 8 devices. This is the factory configuration. 
J3	Header J3 enables (jumper installed) or disables (no jumper) the system controller function.

Caution

Be sure chip orientation is correct, with pin 1 oriented with pin 1 silkscreen markings on the board.

2. Be sure that the two 128K x 8 147Bug EPROMs are installed in the U22 and U30 sockets on the MVME147 module, as shown in the table below.

EPROM Socket	EPROM Description
U22	Even bytes, even B*xx* label
U30	Odd bytes, odd B*xx* label)

3. Refer to the set-up procedure for your particular chassis or system and install the MVME147 as instructed.

4. Connect the terminal which is to be used as the 147Bug system console to connector J7 (port 1) on the MVME712/MVME712M 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 MVME147 ports at power-up.

After power-up, the baud rate of the J7 port (port 1) can be reconfigured by using the Port Format (**PF**) command of the 147Bug debugger.

Note In order for high-baud rate serial communication between 147Bug 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 you get garbled messages and missing characters, then you should check the terminal to make sure XON/XOFF handshaking is enabled.

5. If you want to connect device(s) (such as a host computer system or a serial printer) to ports 2, 3, and/or port 4 on the MVME712/MVME712M, connect the appropriate cables and configure the port(s) as detailed in the manual that you received with your transition board. After power-up, these ports can be reconfigured by using the **PF** command of the 147Bug debugger (refer to Chapters 2 and 3 of this manual).

6. Power up the system. The 147Bug executes self-checks and displays the debugger prompt `147-Bug>`.

   If after a delay, the 147Bug begins to display test result messages on the bottom line of the screen in rapid succession, the MVME147 is in the 147Bug “system” mode. If this is not the desired mode of operation, then press the **ABORT** switch on the front panel of the MVME147. When the menu is displayed, enter a **3** to go to the system debugger. (Refer to Appendix A.) The environment may be changed by using the Set Environment (**ENV**) command (Chapter 3).

When power is applied to the MVME147, bit 1 at location $FFFE1029 (Peripheral Channel Controller (PCC) general purpose status register) is set to 1 indicating that power was just applied. (Refer to the *MVME147-0xx MPU VMEmodule Installation and Use* manual for a description of the PCC.) This bit is tested within the "Reset" logic path to see if the power-up confidence test needs to be executed. This bit is cleared by writing a 1 to it, thus preventing any future power-up confidence test execution.

- **Successful Test:** If the power-up confidence test is successful and no failures are detected, the firmware monitor comes up normally, with the **FAIL** LED off.
- **Unsucessful Test:** 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'
... '+12v fuse is open'
... 'Battery low (data may be corrupted)'
... 'Unable to access non-volatile RAM properly'
```

The firmware monitor comes up with the **FAIL** LED on. Refer to the trouble-shooting section of the *MVME147-0xx MPU VMEmodule Installation and Use* manual.

7. After successfully powering up the system, you may wish to use 147Bug's **SET** command (Chapter 3) to verify the Real Time Clock (RTC)'s date and time.

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 you 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 on 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, you want to gain control without Autoboot, hit the **BREAK** key.

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 and the environment has been set for Bug mode (refer to the *Set Environment to Bug or OS - ENV* section in Chapter 3), 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 your module to gain control through the ROMboot linkage, four requirements must be met:

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

2. Your routine must be located within the MVME147 ROM memory map (but the **RB** command can change this to any other portion of the onboard memory, or even off-board VMEbus memory).

3. The ASCII string `“BOOT”` must be located within the specified memory range.
4. Your 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 and verified with the **CS** command. (Refer to the **CS** command description and examples.)

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.
$04	4 bytes	Entry offset	Longword offset from `“BOOT”`.
$08	4 bytes	Routine length	Longword, includes length from module offset $00 to and including checksum.
$0C	?	Routine name	ASCII string containing routine name.

If you wish to make use of ROMboot you do not have to fill a complete ROM. Any partial amount is acceptable, as long as the length reflects where the checksum is correct. By convention within Motorola, the checksum is placed in the two bytes following the routine.

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:

1. The map is searched for the ASCII string `“BOOT”`.
2. If the ASCII string `“BOOT”` is found, it is still undetermined whether the routine is meant to gain control. To verify that

this is the case, the bytes starting from the beginning of "`BOOT`" through the end of the routine (as defined by the 4-byte length at offset $8) are run through the 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 for "`BOOT`" is as follows:

1. Search direct address (as set by the **RB** command).
2. Search non-volatile RAM (first 1K bytes of battery back-up RAM).
3. Search complete ROM map.
4. Search local RAM (if **RB** command has selected to operate on any reset), at all 8K byte boundaries starting at $00006000.
5. Search the VMEbus map (if so selected by the **RB** command) on all 8K byte boundaries starting at the end of the onboard RAM.

The following example performs the following:

1. Outputs a (**CR**)(**LF**) sequence to the default output port.
2. Displays the date and time from the current cursor position.
3. Outputs two more (**CR**)(**LF**) sequences to the default output port.
4. Returns control to 147Bug.

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, from a disk using **IOP**, or by hand using the **MM** command):

```
147-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   ................
```

```
147-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
```

```
147-Bug>CS 6000 602E
Effective Address: 00006000
Effective Address: 0000602D
Even/Odd = F99F
```

Perform checksum on locations 6000 through 602E (refer to **CS** command).

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

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

```
147-Bug>CS 6000 602E
Effective Address: 00006000
Effective Address: 0000602D.
Even/Odd = 0000
```

Verify that checksum is correct**.**

```
147-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   ................
147-Bug>
```

The routine is now recognized by the ROMboot function when it is enabled by the **RB** command.

Restarting the System

You can initialize the system to a known state in three different ways: Reset, Abort, and Break.

Each has characteristics which make it more appropriate than the others in certain situations.

Reset

Pressing and releasing the MVME147 front panel **RESET** switch initiates a reset. COLD and WARM reset modes are available. By default, 147Bug is in COLD reset mode (refer to the **RESET** command description).

- **COLD Reset.** During a cold reset, a total board initialization takes place, as if the MVME147 had just been powered up.

- The breakpoint table and offset registers are cleared.
- The user 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.

❑ **WARM Reset.** A warm reset differs in that:

- The breakpoint table and offset registers are preserved.
- The user registers are preserved.
- All static variables (including disk device and controller parameters) are preserved.

If the particular MVME147 is the system controller, then a system reset is issued to the VMEbus and other modules in the system are reset as well.

The local reset feature (when the MVME147 is NOT the system controller) is a partial system reset, not a complete system reset such as power-up or SYSRESET. When the local bus reset signal is asserted, a local bus cycle may be aborted. Because the VMEchip is connected to both the local bus and the VMEbus, if the aborted cycle is bound for the VMEbus, erratic operation may result.

Communications between the local processor and the VMEbus should be terminated by an abort; reset should be used only when the local processor is halted or the local bus is hung and reset is the last resort.

Reset must be used if the processor ever halts (as evidenced by the MVME147 illuminated **STAT** LED), for example after a double bus fault; or if the 147Bug environment is ever lost (vector table is destroyed, etc.).

Abort

Pressing and releasing the **ABORT** switch on the MVME147 front panel invokes an "abort". When abort is invoked while 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 your 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:

- ❏ SCSI ID set to 7.
- ❏ Memory sized flag is cleared (onboard memory is sized on this reset).
- ❏ AUTOboot is turned off.
- ❏ ROMboot is turned off.
- ❏ Environment set for Bug mode.
- ❏ Automatic SCSI bus reset is turned off.
- ❏ Onboard diagnostic switch is turned on (for this reset only).
- ❏ System memory sizing is turned on (System mode).
- ❏ Console set to port 1 (LUN 0).
- ❏ Port 1 (LUN 0) set to use ROM defaults for initialization.
- ❏ Concurrent mode is turned off.

In this situation, if a failure occurs during the onboard diagnostics, the **FAIL** LED repeatedly flashes a code to indicate the failure. The on/off LED time for code flashing is approximately 0.25 seconds. The delay between codes is approximately two seconds. To complete bug initialization, press the **ABORT** switch while the LED is flashing. When initialization is complete, a failure message is displayed. LED flashes indicate confidence test failures per the following table.

Number of LED Flashes	Description
1	CPU register test failure
2	CPU instruction test failure
3	ROM test failure
4	Onboard RAM test (first 16KB) failure
5	CPU addressing mode test failure
6	CPU exception processing test failure
7	+12 Vdc fuse failure
10	NVRAM battery low
11	Trouble with the NVRAM
12	Trouble with the RTC

Break

Pressing and releasing the **BREAK** key on the terminal keyboard generates a “break”. 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 your 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 you may wish to terminate a debugger command prior to its completion; for example, when displaying a large block of memory. Break allows you 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 147Bug is approximately 256KB of code. The EPROM sockets on the MVME147 are mapped starting at location $FF800000, contained entirely in EPROM, and consist of debugger and diagnostic packages. However, 147Bug code is position-independent and executes anywhere in memory; SCSI firmware code is not position-independent.

The 147Bug requires a minimum of 16KB of contiguous read/write memory to operate.

When programming the PCC slave base address register, in order to select the address at which onboard RAM appears from the VMEbus, refer to the following table.

Table 1-1. DRAM Address Viewed from VMEbus

RBA4	RBA3	RBA2	RBA1	RBA0	Beginning Address	Ending Address	Notes
0	0	0	0	0	$00000000	(1 x DRAMsize)-1	
0	0	0	0	1	1 x DRAMsize	(2 x DRAMsize)-1	1, 2
0	0	0	1	0	2 x DRAMsize	(3 x DRAMsize)-1	1, 2
0	0	0	1	1	3 x DRAMsize	(4 x DRAMsize)-1	1, 2
0	0	1	0	0	4 x DRAMsize	(5 x DRAMsize)-1	1, 2
0	0	1	0	1	5 x DRAMsize	(6 x DRAMsize)-1	1, 2
0	0	1	1	0	6 x DRAMsize	(7 x DRAMsize)-1	1, 2
0	0	1	1	1	7 x DRAMsize	(8 x DRAMsize)-1	1, 2
0	1	0	0	0	8 x DRAMsize	(9 x DRAMsize)-1	1, 2
0	1	0	0	1	9 x DRAMsize	(10 x DRAMsize)-1	1, 2
0	1	0	1	0	10 x DRAMsize	(11 x DRAMsize)-1	1, 2
0	1	0	1	1	11 x DRAMsize	(12 x DRAMsize)-1	1, 2
0	1	1	0	0	12 x DRAMsize	(13 x DRAMsize)-1	1, 2
0	1	1	0	1	13 x DRAMsize	(14 x DRAMsize)-1	1, 2
0	1	1	1	0	14 x DRAMsize	(15 x DRAMsize)-1	1, 2
0	1	1	1	1	15 x DRAMsize	(16 x DRAMsize)-1	1, 2
1	0	0	0	0	16 x DRAMsize	(17 x DRAMsize)-1	1, 2
1	0	0	0	1	17 x DRAMsize	(18 x DRAMsize)-1	1, 2

Table 1-1. DRAM Address Viewed from VMEbus (Continued)

RBA4	RBA3	RBA2	RBA1	RBA0	Beginning Address	Ending Address	Notes
1	0	0	1	0	18 x DRAMsize	(19 x DRAMsize)-1	1, 2
1	0	0	1	1	19 x DRAMsize	(20 x DRAMsize)-1	1, 2
1	0	1	0	0	20 x DRAMsize	(21 x DRAMsize)-1	1, 2
1	0	1	0	1	21 x DRAMsize	(22 x DRAMsize)-1	1, 2
1	0	1	1	0	22 x DRAMsize	(23 x DRAMsize)-1	1, 2
1	0	1	1	1	23 x DRAMsize	(24 x DRAMsize)-1	1, 2
1	1	0	0	0	24 x DRAMsize	(25 x DRAMsize)-1	1, 2
1	1	0	0	1	25 x DRAMsize	(26 x DRAMsize)-1	1, 2
1	1	0	1	0	26 x DRAMsize	(27 x DRAMsize)-1	1, 2
1	1	0	1	1	27 x DRAMsize	(28 x DRAMsize)-1	1, 2
1	1	1	0	0	$00000000	(1 x DRAMsize)-1	1, 3, 4
1	1	1	0	1	1 x DRAMsize	(2 x DRAMsize)-1	1, 3, 4

Notes

1. DRAMsize = the size of the DRAM. For example, if the 4Mb version is used, then DRAMsize = $400000, and (3 x DRAMsize)-1 = $BFFFFF.
2. When beginning address is less then 16MB, the DRAM responds to standard or extended address modifiers. When beginning address is 16MB or greater, the DRAM responds to extended address modifiers only. Note that bits 4 and 5 in the VMEchip Slave Address Modifier Register further control response to standard and extended address modifiers.
3. This combination pertains only to DRAMsize of 16Mb or 32MB.
4. The values shown in the table refer to extended addresses only. In the standard address range the DRAM responds to $000000 through $7FFFFF.

The first 16KB of onboard RAM is used for 147Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME147 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.

The following abbreviated memory map for the MVME147 highlights addresses that might be of particular interest to you. 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 Location	**Function**
`00000000-000003FF`	Target vector area
`00000400-000007FF`	Bug vector area
`00000800-00000803`	MPCR (Multi-Processor Control Register)
`00000804-00000807`	MPAR (Multi-Processor Address Register)
`00000808-000037DF`	Work area and stack for MVME147 debug monitor
`000037E0-00003FFF`	SCSI firmware work area

EPROM Location	**Function**
`FF800000-FF800003`	Supervisor stack address used when **RESET** switch is pressed
`FF800004-FF800007`	Program Counter (PC) used when **RESET** switch is pressed
`FF800008-FF80000B`	Size of code
`FF80000C-FF80000F`	Reserved
`FF83FFFA-FF83FFFB`	Even/odd revision number of the two monitor EPROMs
`FF83FFFC-FF83FFFD`	Even/odd socket number where monitor EPROMs reside
`FF83FFFE-FF83FFFF`	Even/odd checksum of the two monitor EPROMs
`FFA00000-FFBFFFFF`	Reserved for user

Note: $FF800000 to $FF83FFFF in sockets U22 (even) and U30 (odd)
$FFA00000 to $FFBFFFFF in sockets U1 (even), U15 (odd)

BBRAM Location	Function
`FFFE0000-FFFE03FF`	Reserved for user
`FFFE0000-FFFE000F`	Dynamic burnin pattern (0F-00 do burnin loop in factory only)
`FFFE0400-FFFE05FF`	Reserved for operating system use
`FFFE0600-FFFE06C1`	Disk/Tape I/O Map, set via the **IOT** command
`FFFE06C2-FFFE073E`	Reserved for Bug use
`FFFE073F`	Maintain Concurrent Mode through a Power Cycle/Reset, set via the **ENV** command (Y/N)
`FFFE0741`	VMEchip VMEbus Interrupt Handler Mask Register
`FFFE0742`	Power-up confidence test fail flag
`FFFE0743`	CPU clock frequency
`FFFE0744-FFFE0745`	Onboard console port number
`FFFE0746-FFFE0755`	Serial port map (up to 8 ports)
`FFFE0756`	VMEchip Utility Interrupt Mask Register
`FFFE0757`	VMEchip Utility Interrupt Vector Register
`FFFE0758`	VMEchip GCSR Base Address Configuration Register
`FFFE0759`	VMEchip Board Identification Register
`FFFE075A-FFFE075B`	Checksum for VMEchip registers
`FFFE075C-FFFE075F`	VBR saved for MEMFIND routine
`FFFE0760-FFFE0761`	Board base number (BCD)
`FFFE0762`	Board B number (BCD)
`FFFE0763`	Board Rev. letter (ASCII)
`FFFE0764-FFFE0767`	System off-board RAM start address
`FFFE0768-FFFE076B`	System off-board RAM end address

FFFE076C	Execute/Bypass SST memory test, set via the **ENV** command
FFFE076D	Board configuration register
FFFE076E	Reset SCSI bus switch, set via RESET command
FFFE076F	Reserved
FFFE0770	Reserved
FFFE0771	Onboard diagnostic switch
FFFE0772	System memory sizing flag
FFFE0773	Execute/Bypass auto self test, set via **ENV** command
FFFE0774-FFFE0777	End of onboard memory+1, set via memory sizing routine
FFFE0778-FFFE077A	Ethernet station address.
FFFE077B	Onboard memory sizing flag.
FFFE077C-FFFE07A5	SCSI firmware jump table
FFFE077C	Jump to SCSI command entry
FFFE0782	Jump to SCSI reactivation entry
FFFE0788	Jump to SCSI interrupt entry
FFFE078E	Jump to SCSI FUNNEL command entry
FFFE0794	Jump to SCSI come-again entry
FFFE079A	Jump to SCSI RTE entry
FFFE07A0-FFFE07A5	Reserved
FFFE07A6	Local SCSI ID level (7)
FFFE07A7-FFFE07C5	SCSI trace switches (reserved for internal use).
FFFE07C6	AUTOboot controller number, set via **AB** command
FFFE07C7	AUTOboot device number, set via **AB** command
FFFE07C8-FFFE07E3	AUTOboot string, set via **AB** command
FFFE07E4	Off-board address multiplier, set via **OBA** command
FFFE07E5-FFFE07E9	Reserved

`FFFE07EA-FFFE07EF`	ROMboot direct address, set via **RB** command
`FFFE07F0`	AUTOboot enable switch, set via [**NO**]**AB** command (Y/N)
`FFFE07F1`	AUTOboot at power-up switch, set via **AB** command (P/R)
`FFFE07F2`	ROMboot enable switch, set via [**NO**]RB command (Y/N)
`FFFE07F3`	ROMboot from VMEbus switch, set via **RB** command (Y/N)
`FFFE07F4`	ROMboot at power-up switch, set via **RB** command (P/R)
`FFFE07F5`	RTC flag
`FFFE07F6`	Bug/System switch, set via **ENV** command (B/S)
`FFFE07F7`	Reserved
`FFFE07F8-FFFE07FF`	Time of day clock

I/O Hardware Address	**Function**
`FFFE3002-FFFE3003`	Serial port 1
`FFFE3000-FFFE3001`	Serial port 2
`FFFE3802-FFFE3803`	Serial port 3
`FFFE3800-FFFE3801`	Serial port 4
`FFFE2800`	Printer port
`FFFE1000-FFFE102F`	PCC registers
`FFFE1800-FFFE1803`	LANCE (AM7990) registers
`FFFE2000-FFFE201F`	VME gate array registers
`FFFE4000-FFFE401F`	SCSI (WD33C93) registers

Disk I/O Support

147Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 147Bug consist of:

- ❏ Command-level disk operations
- ❏ Disk I/O system calls (only via the TRAP #15 instruction) for use by user programs
- ❏ Defined data structures for disk parameters

Parameters such as the following are kept in tables by 147Bug.

- ❏ Address where the module is mapped
- ❏ Type of devices attached to the controller module
- ❏ Number of devices attached to the controller module

Default values for these parameters are assigned at power-up and cold-start reset, but may be altered as described in the *Default 147Bug Controller and Device Parameters* section 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 147Bug 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. 147Bug does the following:

- ❑ Translates this into an equivalent sector specification
- ❑ Passes it 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 147Bug Commands

The following 147Bug 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, 147Bug remembers them so that the next disk command defaults to use the same controller and device.

IOP (Physical I/O to Disk)

IOP allows you to:

- ❑ Read or write blocks of data
- ❑ Format the specified device in a certain way

IOP does the following:

- ❑ Creates a command packet from the arguments you specified
- ❑ Invokes the proper system call function to carry out the operation

IOT (I/O Teach)

IOT allows you to:

- ❑ Change any configurable parameters and attributes of the device
- ❑ See the controllers available in the system

IOC (I/O Control)

IOC allows you to:

- ❑ Send command packets as defined by the particular controller directly.
- ❑ Look at the resultant device packet after using the **IOP** command

BO (Bootstrap Operating System)

BO does the following:

- ❑ Reads an operating system or control program from the specified device into memory
- ❑ Transfers control to it

BH (Bootstrap and Halt)

BH is used as a debugging tool. It does the following:

- ❑ Reads an operating system or control program from a specified device into memory
- ❑ Returns control to 147Bug

Disk I/O via 147Bug System Calls

All operations that actually access the disk are done directly or indirectly by 147Bug 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 disk/tape into memory.
.DSKWR	Disk write. System call to write blocks from memory onto disk/tape.

.DSKCFIG	Disk configure. This function allows you to change the configuration of the specified device.
.DSKFMT	Disk format. This function allows you 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/tape functions.

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

To perform a disk operation, 147Bug 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 do the following:

❏ Accept a generalized (controller-independent) packet format as an argument

❏ Translate it into a controller-specific packet

❏ Send it to the specified device

Refer to the system call descriptions in Chapter 5 for details on the format and construction of these standardized user packets.

The packets which a controller module expects to receive 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 receiving it. Refer to documentation on the particular controller module for the format of its packets, and for using the IOC command.

Default 147Bug Controller and Device Parameters

The **IOT** command, with the **T** (teach) option specified, must be invoked to initialize the parameter tables for available controllers and devices. This option instructs **IOT** to scan the system for all currently supported disk/tape controllers (refer to Appendix E) and build a map of the available controllers. This map is built in the Bug RAM area, but can also be saved in NVRAM if so instructed. If the map is saved in NVRAM, then after a reset, the map residing in NVRAM is copied to the Bug RAM area and used as the working map. If the map is not saved in NVRAM, then the map is temporary and the **IOT;T** command must be invoked again if a reset occurs.

If the device is formatted and has a configuration area, then during the first device access or during a boot, **IOT** is not required. Reconfiguration is done automatically by reading the configuration area from the device, then the descriptor for the device is modified according to the parameter information contained in the configuration area. (Appendix D has more information on the disk configuration area.)

If the device is not formatted or of unknown format, or has no configuration area, then before attempting to access the device, you should verify the parameters, using **IOT**. The **IOT** command may be used to manually reconfigure the parameter table for any controller and/or device that is different from the default. These are temporary changes and are overwritten with default parameters, if a reset occurs.

The **IOT;T** command should also be invoked any time the controllers are changed or when ever the NVRAM map has been damaged or not initialized (“`No Disk Controllers Available`” is displayed when the **IOT;H** command is invoked).

Disk I/O Error Codes

The 147Bug 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 MVME147 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 MVME147 dual-I/O port RAM by issuing a remote **GO** command using the Multiprocessor Control Register (MPCR). The MPCR, located at shared RAM location base address plus $800, contains one of two longwords used to control communication between processors. The MPCR contents are organized as follows:

Base Address + $800	*	N/A	N/A	N/A	MPCR

The codes stored in the MPCR are of two types:

❑ Status returned (from 147Bug):

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

❑ Command set by the bus master (job requested by some processor):

ASCII G	(Hexadecimal 47)	Use Go Direct (**GD**) logic specifying the MPAR address.
ASCII B	(Hexadecimal 42)	Recognize breakpoints using the Go (**G**) logic.

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

Base Address + $804	MSB	*	*	LSB	MPAR

At power-up, the debug monitor self-test routines initialize RAM, including the memory locations used for multiprocessor 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 your 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 previously set breakpoints are enabled when 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 the execution address. (Any remote processor could examine the MPCR contents.)

If the code being executed 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 147Bug package is a complete set of hardware diagnostics intended for testing and troubleshooting of the MVME147 (refer to Chapter 6). In order to use the diagnostics, you must be in the diagnostic directory. If you are in the debugger directory, you can switch to the diagnostic directory by entering the debugger command Switch Directories (**SD**). The diagnostic prompt `147-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.

Using the Debugger 2

Entering Command Lines

147Bug is command-driven and performs its various operations in response to the commands entered at the keyboard. When the debugger prompt `147-Bug>` appears on the terminal screen, 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 you to correct entry errors, if necessary.

When a command is entered the debugger executes the command and the prompt reappears. However, if the command entered causes execution of your target code; i.e., **GO**, then control may or may not return to the debugger, depending on what the your program does. For example, if a breakpoint has been specified, then control is returned to the debugger when the breakpoint is encountered during execution of your program. Alternately, your program could return control to the debugger by means of the TRAP #15 function **.RETURN** (described in Chapter 5). For more about this, refer to the description in Chapter 3 for the **GO** commands.

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 upper- or lower-case may be used.

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 the 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.

Note The presence of the upward caret (**^**) before a character indicates that the Control or **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 (see **PF** command) 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.
Delete	Delete	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 147Bug 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 147Bug but may be changed by 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 string	A boldface string is a literal such as a command or a program name, and is to be typed just as it appears.
italic string	An italic string is a "syntactic variable" and is to be replaced by one of a class of items it represents.
`Fixed space font`	Used throughout in examples of screen data.
|	A vertical bar separating two or more items indicates that a choice is to be made; only one of the items separated by this symbol should 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.

Follow all inputs by pressing the carriage return key (**Return** or **Enter**). This is shown, as (**CR**), only if it is the only input required.

Command Arguments

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 in the *Expression as a Parameter* section in this chapter).
addr	Address (described in detail in the *Address as a Parameter* section in this chapter).
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 these arithmetic operators:

+	Plus
-	Minus
*	Multiply by
/	Divide by
&	Logical AND
<<	Shift left
>>	Shift right

Numeric values may be expressed in either Hexadecimal, Decimal, Octal, or Binary by immediately preceding them with the proper base identifier, as shown in the following table.

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 examples:

String Literal	Numeric Value (in Hexadecimal)
'A'	41
'ABC'	414243
'TEST'	54455354

Evaluation of an expression is performed according to the following rules:

- ❑ Always evaluated from left to right unless parentheses are used to group part of the expression
- ❑ No operator precedence
- ❑ Sub-expressions within parentheses evaluated first
- ❑ Nested parenthetical sub-expressions evaluated from the inside out

Valid expression examples.

Expression	Result (in Hexadecimal)	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 147Bug 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

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

Table 2-1. Debugger Address Parameter Formats

Format	Example	Description
N	140	Absolute address+contents of automatic offset register.
N+**R***n*	130+R5	Absolute address+contents of the specified offset register (not an assembler-accepted syntax).
(**A***n*)	(A1)	Address register indirect, also post-increment, pre-decrement)
(*d*,**A***n*) or *d*(**A***n*)	(120,A1) 120(A1)	Address register indirect with displacement (two formats accepted).
(*d*,**A***n*,*Xn***)** or *d***(A***n*,*Xn***)**	(&120,A1,D2) &120(A1,D2)	Address register indirect with index and displacement (two formats accepted).
([*bd*,**A***n*,*Xn***],***od***)**	([C,A2,A3],&100)	Memory indirect preindexed.
([*bd*,**A***n***],***Xn*,*od***)**	([12,A3],D2,&10)	Memory indirect postindexed.

Table 2-1. Debugger Address Parameter Formats (Continued)

Format	Example	Description
For the memory indirect modes, fields can be omitted. For example, three of many permutations are as follows:		
([,A*n*],*od*)	([,A1],4)	
([*bd*])	([FC1E])	
([*bd*,,*Xn*])	([8,,D2])	

Notes **1.**

N	Absolute address (any valid expression).
A*n*	Address register *n*.
Xn	Index register n (A*n* or D*n*).
d	Displacement (any valid expression).
bd	Base displacement (any valid expression).
od	Outer displacement (any valid expression).
n	Register number (0 to 7).
R*n*	Offset register *n*.

2. In commands with *range* specified as *addr del addr*, and with size option **W** or **L** chosen, data at the second (ending) address is acted on only if the second address is a proper boundary for a word or longword, respectively.

Offset Registers

Eight pseudo-registers (R0 through 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
- ❑ 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. For additional information about the offset registers, see the **OF** command description.

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 an assembled, relocatable module 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:

```
147Bug>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
147Bug>
```

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

```
147Bug>OF R0
R0 =00000000 00000000? 1327C:16. <CR>
147Bug>MD 0+R0;DI  <CR>
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
147Bug>
```

Port Numbers

Some 147Bug commands give you 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	MVME147 RS-232-D (MVME712/MVME712M serial port 1)
1	MVME147 RS-232-D (MVME712/MVME712M serial port 2)
2	MVME147 RS-232-D (MVME712/MVME712M serial port 3)
3	MVME147 RS-232-D (MVME712/MVME712M serial port 4)
4	MVME147 Printer Port (MVME712/MVME712M printer)

Note These logical port numbers (0, 1, 2, 3, and 4) are referred to as “Serial Port 1“, “Serial Port 2“, “Serial Port 3“, “Serial Port 4“, and “Printer Port”, respectively, by the MVME147 hardware documentation and by the MVME712/MVME712M 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 MVME712/MVME712M panel.

Entering and Debugging Programs

There are various ways to enter your program into system memory for execution. One way is to create the program using 147Bug’s Memory Modify (**MM**) command with the assembler/ disassembler option. The program is entered 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 147Bug 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 **MM** command as outlined above and stored to the host using the Dump (**DU**) command. If a communication link exists between the host system and the MVME147 then the file can be downloaded from the host into MVME147 memory via the debugger Load (**LO**) command.

One more way is by reading in the program from disk, using one of the disk commands:

- BO
- BH
- IOP

When the object code has been loaded into memory, you can:

- ❏ Set breakpoints
- ❏ Run the code
- ❏ Trace through the code

Calling System Utilities from Your Programs

A convenient way of doing character input/output, and many other useful operations has been provided so that you do not have to write these routines into the target code. You have access to various 147Bug routines via the MC68030 TRAP #15 instruction vector. Refer to Chapter 5 for details on the various TRAP #15 utilities available and how to invoke them from within your program.

Preserving the Debugger Operating Environment

This section explains how to avoid contaminating the operating environment of the debugger. 147Bug uses certain of the MVME147 onboard resources and uses onboard memory to contain temporary variables, exception vectors, etc. If you disturb resources upon which 147Bug depends, then the debugger may function unreliably or not at all.

147Bug Vector Table and Workspace

As described in the *Memory Requirements* section in Chapter 1, 147Bug needs 16KB of read/write memory to operate. 147Bug 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, 147Bug:

- ❏ Reserves space for static variables

- ❑ Initializes the static variables to predefined default values
- ❑ Allocates space for the system stack
- ❑ Initializes the system stack pointer to the top of this area

With the exception of the first 1024-byte vector table area, you must be extremely careful not to use the above-mentioned memory areas for other purposes. Refer to the *Memory Requirements* section in Chapter 1 to determine how to dictate the location of the reserved memory areas. If, for example, your 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 your program corrupts the system stack, then an incorrect value may be loaded into the processor Program Counter (PC), causing a system crash.

Tick Timers

The MVME147 uses the PCC tick timer 1 to generate accurate delays for program timing (refer to *MVME147 MPU VMEmodule User's Manual*).

Exception Vectors Used By 147Bug

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

When the debugger handles one of the exceptions listed in Table 2-2, 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 you.

Table 2-2. Exception Vectors Used by 147Bug

Vector Offset	Exception	147Bug Facility
$8	Bus Error	
$10	Illegal Instruction	Breakpoints (used by GO, GN, GT)
$24	Trace	Trace operations (such as T, TC, TT)
$108	Level 7 Interrupt	ABORT push-button
$BC	TRAP #15	System calls (refer to Chapter 5)

Example

Trace one instruction using the debugger.

```
147Bug>RD
PC   =00004000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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 7055              MOVEQ.L     # $55, D0
147Bug>T
PC   =00004002 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
D0   =00000055 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
00004002 4E71              NOP
147Bug>
```

Notice that the value of the target stack pointer register (A7) has not changed even though a trace exception has taken place. Your program may either use the exception vector table provided by 147Bug or it may create a separate exception vector table of its own. The two following sections detail these two methods.

Using the 147Bug Target Vector Table

147Bug initializes and maintains a vector table area for target programs. A target program is any program started by the bug:

- ❑ Manually with the **GO** command
- ❑ Manually with Trace commands (**T**, **TL**, **TT**)
- ❑ Automatically with the **BO** command

The start address of this target vector table area is the base address ($00) of the MVME147 module. This address is loaded into the target-state 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.

147Bug initializes the target vector table with the debugger vectors listed in Table 2-2 and fills the other vector locations with the address of a generalized exception handler (refer to the *147Bug Generalized Exception Handler* section 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 2-2 are overwritten then the accompanying debugger functions are lost.

147Bug maintains a separate vector table for its own use. In general, you do not have to be aware of the existence of the debugger vector table. It is completely transparent and you should never make any modifications to the vectors contained in it.

Creating a New Vector Table

Your program may create a separate vector table in memory to contain its own exception vectors. If this is done, the program must change the value of the VBR to point at the new vector table. In order to use the debugger facilities you can copy the proper vectors from the 147Bug vector table into the corresponding vector locations in your program vector table.

The vector for the 147Bug generalized exception handler (described in detail in the *147Bug Generalized Exception Handler* section in this chapter may be copied from offset $3C (Uninitialized Interrupt) in the target vector table to all locations in your program vector table where a separate exception handler is not used. This provides diagnostic support in the event that your program is stopped by an unexpected exception. The generalized exception handler gives a formatted display of the target registers and identifies the type of the exception.

Example

The following routine 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   $3C(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   $8(A0),$8(A1)         Copy bus error vector.
         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.
         MOVE.L   $108(A0),$108(A1)     Copy ABORT vector.
         LEA.L    COPROCC(PC),A2        Get your 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 your program uses one or more of the exception vectors that are required for debugger operation. Debugger facilities may still be used, however, if your exception handler can determine when to handle the exception itself and when to pass the exception to the debugger.

When an exception occurs which you want to pass on to the debugger; i.e., **ABORT**, your 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 147Bug target program vector table (which your program saved), yielding the address of the 147Bug exception vector. The program then jumps to the address stored at this vector location, which is the address of the 147Bug exception handler.

Your 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 for the particular exception before jumping to the address of the exception handler.

Example

The following example is an exception handler that 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 your code handles exception, if so, branch.
       .
       MOVE.L     BUFVBR,A0    Pass exception to debugger; Get saved 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.
```

147Bug Generalized Exception Handler

147Bug has a generalized exception handler which it uses to handle all of the exceptions not listed in Table 2-2. For all these exceptions, the target stack pointer is left pointing to the top of the exception stack frame created. In this way, if an unexpected exception occurs during execution of your code, you are 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 initiates bus error exception processing.

```
147Bug>RD
PC   =00004000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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 203900F0          MOVE.L      ($F00000).L,D0
147Bug>T

VMEbus Error

Exception: Long Bus Error
Format/Vector=B008
SSW=074D Fault Addr.=00F00000 Data In=FFFFFFFF Data Out=00004006
PC   =00004000 SR   =A700=TR:ALL_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00005FA4 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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   =00005FA4
00004000 203900F0          MOVE.L      ($F00000).L,D0
147Bug>
```

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.

```
147Bug>MD (A7):&44
00005FA4 A700 0000 4000 B008  3EEE 074D FFFF 094E  '..@.0.>n.M...N
00005FB4 00F0 0000 00F0 0000  0000 35EC 2039 0000  p...p....5l 9..
00005FC4 0000 400A 0000 4008  0000 4006 FFFF FFFF ..@...@...@.....
00005FD4 00F0 0000 100F F487  0000 A700 FFFF FFFF .p....t...'.....
00005FE4 0000 7FFF 0000 0000  9F90 0000 0000 6000 ..............'.
00005FF4 0000 0000 0000 0000                              ........
147Bug>
```

Memory Management Unit Support

The Memory Management Unit (MMU) is supported in 147Bug. An MMU confidence check is run at power-up to verify that the registers can be accessed. It also ensures that a context switch can be done successfully. 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 147Bug. Finally, there is a set of diagnostics to test functionality of the MMU.

At power-up, an 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 Register	(DL)
TC	Translation Control Register	(L)
TT0	Transparent Translation 0 Register	(L)
TT1	Transparent Translation 1 Register	(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 memory management unit, 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 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 (*addr*) and function code (*FC*) specification is:

*addr*__^__*FC*

The valid function code mnemonics are shown in the following table:

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

Notes
1. Using an unassigned or reserved function code or mnemonic results in a Long Bus Error message.
2. If the symbol **^** (up arrow or caret) is used without a function code or mnemonic, the function code display is turned off.

Example

Change data at location $5000 in your data space:

```
147Bug>M 5000^ud
00005000^UD 0000 ? 1234.
147Bug>
```

Debugger Command Set 3

Introduction

This chapter contains descriptions of each of the debugger commands and provides one or more examples of each. Table 3-1 summarizes the 147Bug debugger commands.

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 147Bug). The symbol (**CR**) represents the "carriage return" (**Return** or **Enter**) key on your terminal keyboard. The (**CR**) is shown only if it is the only user input.

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
G/GO	Go Execute Target Code
GD	Go Direct (Ignore Breakpoints)

Table 3-1. Debugger Commands (Continued)

Command Mnemonic	Title
GN	Go to Next Instruction and Stop
GT	Go to Temporary Breakpoint
HE	Help
IOC	I/O Control for Disk/Tape
IOP	I/O Physical (Direct Disk/Tape Access)
IOT	I/O “Teach” for Disk Configuration
LO	Load S-records from Host
LSAD	LAN Station Address Display/Set
MA/NOMA	Macro Define/Display/Delete
MAE	Macro Edit
MAL/NOMAL	Enable/Disable Macro Expansion Listing
MAW/MAR	Save/Load Macros
M/MM	Memory Modify
MD	Memory Display
MENU	System Menu
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 CS0
RESET	Cold/Warm Reset
RM	Register Modify
RS	Register Set
SD	Switch Directories
SET	Set Time and Date
T	Trace Instruction
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

Autoboot Enable/Disable - AB/NOAB

Command Input

AB
NOAB

Description

The **AB** command lets you 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 Bootstrap Operating System command, **BO**; Appendix E lists all the possible LUNs). You can also select whether this occurs only at power-up, or at any board reset. These selections are stored in the BBRAM that is part of the MK48T02 (RTC), and remain in effect through power-up or any normal reset. The automatic boot function transfers control to the controller and device specified by the **AB** command.

Note The Reset and Abort option sets the autoboot function to the default condition (disabled) until enabled again by the **AB** command.

The **NOAB** command disables the automatic boot function, but does not change the options chosen. (Refer to Chapter 1 for details on Autoboot.)

Example 1: Enable autoboot function.

```
147-Bug> ab
Controller LUN  =00? (CR)                                  Note 1
Device LUN      =00? (CR)                                  Note 2
Default string  =   ? VME147..                             Note 3
Boot at Power up only or any board Reset [P,R] = P? (CR) Note 4
At power-up only:
Auto Boot from Controller 0, Device 0, VME147..
147-Bug
```

Example 2: Disable autoboot function**.**

```
147-Bug> NOAB                                          Note 5
No Auto Boot from Controller 0, Device 0, VME147...
147-Bug
```

Notes:

1. Select controller for boot.
2. Select device to boot from.
3. Select boot string to pass on.
4. If you select R, then autoboot is attempted at any board reset.
5. This disables the autoboot function, but does not change any options chosen under AB.

Block of Memory Compare - BC

Command Input

BC *range del addr* **[; b | w | l]**

Options (length of data field)

b	Byte
w	Word
l	Longword

Description

The **BC** command compares the contents of the block of memory at addresses defined by *range* to the block of memory, beginning at *addr*. The bytes that differ are displayed along with the addresses. The differences are displayed in two columns; i.e., two to a line.

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 the data that the count is referring to. For example, a count of four with an option of **l** would mean to compare four longwords (or 16 bytes) to the *addr* location. If an option field is specified without a count in the range, an error results. An error also results if the beginning address is greater than the ending address.

Examples

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

```
147-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   ................
```

```
147-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: Memory compares, nothing printed.

```
147-Bug>BC 20000 2001F 21000
Effective address: 00020000
Effective address: 0002001F
Effective address: 00021000
147-Bug>
```

Example 2: Memory compares, nothing printed.

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

Example 3: Create a mismatch, mismatches are printed out.

```
147-Bug>MM 2100F;b
0002100F 21? 0.
147-Bug>BC 20000:20 21000;b
Effective address: 00020000
Effective count  : &32
Effective address: 00021000
0002000F: 21   0002100F: 00
147-Bug>
```

Block of Memory Fill - BF

Command Input

BF *range del data* [*increment*] [**;b** | **w** | **l**]

Arguments

data and *increment* are both expression parameters.

Options (length of data field)

b	Byte
w	Word
l	Longword

Description

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

If the increment you enter does not fit into the data field size, leading bits are truncated to make it fit. If truncation occurs, 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, 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.

For each of the following examples, assume memory from $20000 through $2002F is clear.

Example 1: Default data field length**.**

```
147-Bug>BF 20000,2001F 4E71
Effective address: 00020000
Effective address: 0002001F
147-Bug>MD 20000:18
00020000 4E71 4E71 4E71 4E71  4E71 4E71 4E71 4E71   NqNqNqNqNqNqNqNq
00020010 4E71 4E71 4E71 4E71  4E71 4E71 4E71 4E71   NqNqNqNqNqNqNqNq
00020020 0000 0000 0000 0000  0000 0000 0000 0000   ................
```

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

Example 2: Data larger than specified data field size**.**

```
147-Bug>BF 20000:10 4E71 ;b
Effective address: 00020000
Effective count : &16
Data = $71

147-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: Upper address range not on correct boundary**.**

```
147-Bug>BF 20000,20006 12345678 ; l
Effective address: 00020000
Effective address: 00020003
147-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: Incrementing data**.**

```
147-Bug>BF 20000:18 0 1                Default size is word.
Effective address: 00020000
Effective count : &24

147-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

Command Input

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

Arguments

controller LUN	LUN to which the following device is attached. Defaults to LU 0.
device LUN	LUN of the device to boot from. Defaults to LUN 0.
del	Field delimiter: comma (**,**) or spaces ( ).
string	String that is passed to the operating system or control program loaded. Its syntax and use is completely defined by the loaded program.

Description

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 147Bug debugger and the prompt appears on the terminal screen. Because control is retained by 147Bug, all the 147Bug facilities are available for debugging the loaded program, if necessary.

Example 1

Boot and halt from controller LUN 0, device LUN 1:

```
147-Bug>BH 0,1
147-Bug>
```

Example 2

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

```
147-Bug>BH 3,A,test2;d
147-Bug>
```

Refer to the **BO** command description for more detailed information about what happens during bootstrap loading.

Block of Memory Initialize - BI

Command Input

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 nondestructive; if the parity is correct for a memory location, 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, 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, the message "`RAM FAIL`" is output and the address is given.

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

For the following examples, assume system memory from $0 to $000FFFFF, and that user memory starts at $4000.

Example 1: Range defined as start address and a count.

```
147-Bug>BI 0 : 10000 ;b
Effective address: 00000000
Effective count : &65536
147-Bug>
```

Example 2: Range defined as start and end address**.**

```
147-Bug>BI 4000,FFFFF
Effective address: 00004000
Effective address: 000FFFFF
147-Bug>
```

Example 3: Parity error or memory fault.

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

Block of Memory Move - BM

Command Input

BM *range del addr* [**;b**|**w**|**l**]

Options

b	Byte
w	Word
l	Longword

Description

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* 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 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 $2002F is clear.

```
147-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   ................
147-Bug>BM 21000 2100F 20000
Effective address: 00021000
Effective address: 0002100F
Effective address: 00020000
147-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   ................
147-Bug>
```

Example 2: This utility is very useful for patching assembly code in memory.

Suppose you had a short program in memory at address $20000...

```
147-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
147-Bug>
```

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

```
147-Bug>BM 20002 2000B 20004
Effective address: 00020002
Effective address: 0002000B
Effective address: 00020004
147-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
147-Bug>
```

Now you simply need to enter the NOP at address $20002.

```
147-Bug>MM 20002;DI
00020002   E2A2       ASR.L    D1,D2 ? NOP
00020002   4E71       NOP
00020004   E2A2       ASR.L    D1,D2 ? .
147-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
147-Bug>
```

Bootstrap Operating System - BO

Command Input

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

Arguments

controller LUN	Logical Unit Number (LUN) of the controller to which the following device is attached. Defaults to LUN 0.
device LUN	LUN of the device to boot from. Defaults to LUN 0.
del	Field delimiter: comma (`,`) or spaces ( ).
string	String that is passed to the operating system or control program loaded. Its syntax and use is completely defined by the loaded program.

Description

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 (refer to 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”.`
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; 'IPL*x*', 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, for example, the string pointers, may be used or ignored by the target program.

Examples

`147-Bug>`**BO**	Boot from default *controller LUN, device LUN,* and *string* as defined by **AB** command.
`147-Bug>`**BO 3**	Boot from *controller LUN 3*, default *device LUN*, and *string*.
`147-Bug>`**BO , 3**	Boot from default *controller LUN, device LUN 3*, and default *string*.
`147-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

Command Input

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

Description

The **BR** command allows you 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 to 147Bug. This allows you 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 MPU state on the screen and control is returned to 147Bug.

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, that address is removed from the breakpoint table. If **NOBR** (**CR**) is entered, all entries are deleted from the breakpoint table and the empty table is displayed.

Example

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

Block of Memory Search - BS

Command Input

BS *range del 'text'* [**;b** | **w** | **l**]

BS *range del data del* [*mask*] [**;b** | **w** | **l,n,v**]

Arguments

data and *mask* are both expression parameters.

Options

b	Byte
w	Word
l	Longword
n	Non-aligned
v	Verify

Description

The block search command searches the specified *range* of memory for a match with a *data* pattern entered by you. This command has three modes, as described below.

Mode 1 - Literal text search: In this mode, a search is carried out for the ASCII equivalent of the literal *text* entered by you. This mode is assumed if the single quote (') indicating the beginning and end 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*, no options are allowed. If a match is found, the address of the first byte of the match is output.

Mode 2 - Data search: In this mode, a *data* pattern is entered by you as part of the command line and a size is either entered by you 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:

1. The *data* pattern entered by you is right-justified and leading bits are truncated or leading zeros are added as necessary to make the *data* pattern the specified size.
2. A compare is made with successive bytes, words, or longwords (depending on the size in effect) within the range for a match with the data you entered. 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*.
3. If the "n" (non-aligned) option has been selected, 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.
4. If a match is found, the address of the first byte of the match is output along with the memory contents. If a *mask* was in use, 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, displaying of addresses and data is done only when the memory contents do NOT match the pattern specified by you. 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, 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, you should simply press any character key. To cancel the output and exit the command, you 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, that match is output and a message is output stating that the last match does not lie entirely within the range. You 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:..........
```

```
147-Bug>BS 30000 3002F 'Task Status'
Effective address: 00030000
Effective address: 0003002F
-not found-
```

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

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

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

```
147-Bug>BS 30000 3001F 'ConfigTableStart'
Effective address: 00030000
Effective address: 0003001F
00030014
```

Mode 1: the *text* is found, but it ends outside of the *range*, so the address of its last match extends over range boundary-- first byte and a message are output.

```
147-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 *text* output.

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

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

```
147-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.

```
147-Bug>BS 30000,3002F 3D34 ;n
Effective address: 00030000
Effective address: 0003002F
0003000F|3D34
```

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

```
147-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.

```
147-Bug>BS 30000 3002F 0000 0008;v
Effective address: 00030000
Effective address: 0003002F
0003000E|733D 00030012|2F2F 00030014|436F 0003001C|626C
147-Bug>
```

Mode 3: scan for words with the D3 bit set (non-zero): four locations failed to verify.

Block of Memory Verify - BV

Command Input

BV *range del data* [*increment*] [**;b** | **w** | **l**]

Arguments

data and *increment* are both expression parameters.

Options

b	Byte
w	Word
l	Longword

Description

The **BV** command compares the specified *range* of memory against a *data* pattern. If an *increment* is specified, *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 you 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 *data* or *increment* (if specified) entered does not fit into the *data* field size, leading bits are truncated to make them fit. If truncation occurs, a message is printed stating the *data* pattern and, if applicable, the *increment* value actually used.

If the *range* is specified using a *count*, 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 verified, *data* is verified 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.

```
147-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

147-Bug>BV 20000 2001F 4E71           Default size is word.
Effective address: 00020000
Effective address: 0002001F
147-Bug>                              Verify successful, nothing
                                      printed.
```

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

```
147-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{

147-Bug>BV 20000:30 0;b
Effective address: 00020000
Effective count : &48
0002002A|4A 0002002B|FB 0002002C|4A 0002002D|FB
0002002E|4A 0002002F|FB               Mismatches are printed out.
147-Bug>
```

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

```
147-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   ................

147-Bug>BV 00020000:18,0,1            Default size is word.
Effective address: 00020000
Effective count : &24
00020012|FFFF                         Mismatches are printed out.
147-Bug
```

Checksum - CS

Command Input

CS *address1 address2*

Description

The **CS** command provides access to the same checksum routine used by the firmware. This routine is used in two ways within the firmware monitor.

1. 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 you of the failure.
2. Following a valid power-up test, 147Bug 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* section 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, you need access to the same routine in the preparation of EPROM/ROM routines.

The address parameters can be provided in two forms:

1. An absolute address (32-bit maximum).
2. 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:

1. $FF is placed in each of two bytes within a register. These bytes represent the even and odd bytes as the checksum is calculated.
2. Starting with *address1* the even and odd bytes are extracted from memory and XORed with the bytes in the register.
3. This process is repeated, word by word, until *address2* is reached. This technique allows use of even ending addresses ($20030 as opposed to $2002F).

Examples

Assume the following routine requiring a checksum is in memory. Start at $20000; last byte is at $2002B. Checksum will be placed in bytes at $2002C and $2002D, so they are zero while calculating the checksum.

```
147-Bug>MD 20000:20;w
00020000  424F 4F54 0000 0018  0000 002E 5465 7374   BOOT........Test
00020010  2052 4F4D 424F 4F54  4E4F 0026 4E4F 0052   ROMBOOTNO.&NO.R
00020020  4E4F 0026 4E4F 0026  4E4F 0063 0000 FFFF   NO.&NO.&NO.c....
00020030  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
147-Bug>
```

Disassemble executable instructions.

```
147-Bug>MD 20018;DI
00020018 4E4F0026     SYSCALL    .PCRLF
0002001C 4E4F0052     SYSCALL    .RTC_DSP
00020020 4E4F0026     SYSCALL    .PCRLF
00020024 4E4F0026     SYSCALL    .PCRLF
00020028 4E4F0063     SYSCALL    .RETURN
0002002C 0000FFFF     ORI.B      #$FF,D0    Zeros reserved for
00020030 FFFF         DC.W       $FFFF      checksum.
00020034 FFFF         DC.W       $FFFF
```

Example 1: Using absolute addresses.

```
147-Bug> CS 20000 2002E                Request checksum of routine.
Effective address: 00020000
Effective address: 0002002D
Even/Odd = $F99F                       Checksum of even bytes is $F9.
                                       Checksum of odd bytes is $9F.

147-Bug> M 2002C;w                     Place these bytes in zeroed area
                                       used while calculating checksum.
0002002C 0000 ? F99F.
147-Bug> CS 20000 2002E                Verify checksum.

Effective address: 00020000
Effective address: 0002002D
Even/Odd = $0000                       Result is 0000, good checksum.
147-Bug>
```

Example 2: Using relative offset.

```
147-Bug> OF R3                         Define value of relative offset
R3  =00000000 00000000? 20000.         register 3.

147-Bug> CS 0+R3 2E+R3                 Request checksum of routine.
Effective address: 00000+R3
Effective address: 0002D+R3
Even/Odd = $F99F                       Checksum of even bytes is $F9.
147-Bug>                               Checksum of odd bytes is $9F.

147-Bug> M 2C+R3;w                     Place these bytes in zeroed area
                                       used while checksum was
0000002C+R3 0000 ?F99F.                calculated.

147-Bug> CS 0+R3 2E+R3                 Verify checksum.
Effective address: 00000+R3
Effective address: 0002D+R3
Even/Odd = $0000                       Result is 0000, good checksum.
147-Bug>
```

Data Conversion - DC

Command Input

DC *exp* | *addr*

Description

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, 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

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

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

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

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

```
147-Bug>DC 55&F
00000005 = $5 = &5
```

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

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

```
147-Bug>MD 30000
00030000 11111111  22222222  33333333  44444444  ...."""""3333DDDD

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

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

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

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

Dump S-Records - DU

Command Input

DU [*port*]*del range del*[*text del*][*addr del*][*offset*][**;b** | **w** | **l**]

Options

b	Byte
w	Word
l	Longword

Description

The **DU** command outputs data from memory in the form of Motorola S-records to a port you specify. If *port* is not specified, the S-records are sent to the host port (logical port number 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 is to be incorporated into the header (S0) record of the block of records that is to 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, the address field of the termination record consists of zeros. The termination record is an S7, S8, or S9 record, depending on the address entered. Appendix C has additional information on S-records.

You may also specify an optional offset 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 you to create an S-record file which loads back into memory at a different location than the location from which it was dumped. The default offset is zero.

Caution 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.

Examples: Assume the following routine is in memory starting at $20000 and ending at $20013.

```
147-Bug>MD 20000:10;w
00020000  4E4F 0026 4E4F 0052  4E4F 0026 4E4F 0026   NO.&NO.RNO.&NO.&
00020010  4E4F 0063 FFFF FFFF  FFFF FFFF FFFF FFFF   NO.c............
147-Bug>
```

Disassemble executable instructions.

```
147-Bug>MD 20000;DI
00020000 4E4F0026     SYSCALL    .PCRLF
00020004 4E4F0052     SYSCALL    .RTC_DSP
00020008 4E4F0026     SYSCALL    .PCRLF
0002000C 4E4F0026     SYSCALL    .PCRLF
00020010 4E4F0063     SYSCALL    .RETURN
00020014 FFFF         DC.W       $FFFF
00020016 FFFF         DC.W       $FFFF
00020018 FFFF         DC.W       $FFFF
```

Example 1: Dump memory from $20000 to $2001F to port 1.

```
147-Bug>DU 20000 2001F
Effective address: 00020000
Effective address: 0002001F
147-Bug>
```

Example 2: Dump 10 bytes of memory beginning at $20000 to the terminal screen (port 0).

```
147-Bug>DU 0 20000:&10;b
Effective address: 00020000
Effective count : &10
S0030000FC
S20E020004E4F00264E4F00524E4FA0
S9030000FC
147-Bug>
```

Example 3: Dump memory from $20000 to $2001F to the terminal screen (port 0). Specify a file name of “TEST” in the header record and specify an entry point of $2000A.

```
147-Bug>DU 0 20000 2001F 'test' 2000A
Effective address: 00020000
Effective address: 0002001F
S007000054455354B8
S2140200004E4F00264E4F00524E4F00264E4F0026B1
S2140200104E4F0063FFFFFFFFFFFFFFFFFFFFFFFFE5
S80402000AEF
147-Bug>
```

The following example shows how to upload S-records to a host computer (in this case a system running the UNIX operating system), storing them in the file “FILE1.MX”.

`147-Bug>`**TM** `Escape character: $01=^A` :	Go into transparent mode to establish communication with the host.
(CR) :	Press RETURN or ENTER key to get login prompt.
(login) : :	You must log on to the host and enter the proper directory where FILE1.MX will reside.
# **cat > FILE1.MX** : : : :	At the prompt, invoke the concatenate utility and redirect the output to a file named “FILE1.MX” (the S-records that are to be uploaded).
^A `147-Bug`	Enter escape character (CTRL A) to return to the prompt.

Now enter the command for 147Bug to dump the S-records to the port:

```
147-Bug> DU 20000 2001F 'FILE1'
Effective address: 00020000
Effective address: 0002001F
147-Bug>
```

`147-Bug>`**TM**	Go into transparent mode again.
`Escape character $01 = ^A`	
`:`	
***(INTR)** key)*	Press the "INTR" key to interrupt
`:`	(stop) the "cat" function.
`#` **^d**	When the prompt returns, log off
`:`	of the system.
`login:` **^A**	Enter the escape character (**CTRL A**)
`147-Bug>`	to return to the 147Bug prompt.

EEPROM Programming - EEP

Command Input

EEP *range del addr* [**;w**]

Options

w Word

Description

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 are installed in U1 and U15 (bank 2), and header J1 is configured for the right size EEPROMs. Refer to the *MVME147-0xx MPU VMEmodule Installation and Use* manual for jumper details. U1 and U15 are at addresses starting at $FFA00000 and ending at or below $FFBFFFFF in the main memory map, with the odd-byte chip in U15 and the even-byte chip in U1.

Note that 147Bug is in the EPROMs in U22 and U30 (bank 1), at $FF800000 through $FF83FFFF, with odd bytes in U30 and even bytes in U22.

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

```
147-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   ................
```

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

147-Bug>MD FFA00000:10;w
FFA00000 54 48 49 53 20 49 53 20   41 20 54 45 53 54 21 21   THIS IS A TEST!!
FFA00010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00
00   ................
147-Bug>
```

Example 2

```
147-Bug>EEP 21000:8 FFA00000;w
Effective address: 00021000
Effective count  : &8
Effective address: FFA00000
Programming EEPROM - Done.

147-Bug>MD FFA00000:10;w
FFA00000 54 48 49 53 20 49 53 20   41 20 54 45 53 54 21 21   THIS IS A TEST!!
FFA00010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00
00   ................
147-Bug>
```

Set Environment to Bug or OS - ENV

Command Input

ENV [;D]

Options

D	Update NVRAM with defaults.

Description

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

Note The reset and abort option sets the environment to the default mode (Bug) until changed by the **ENV** command.

When the **ENV** command is invoked, the interactive mode is entered immediately. While in the interactive mode, the following rules apply:

	All numerical values are interpreted as hexadecimal numbers.
	Only listed values are accepted when a list is shown. Uppercase or lowercase may be interchangeably used when a list is shown.
^	Backs up to the previous option.
.	Entering a period by itself or following a new value/setting causes **ENV** to exit the interactive mode. Control returns to the bug.
(CR)	Pressing **Return** (**Enter**) without entering a value preserves the current value and causes the next prompt to be displayed.

If NVRAM has been corrupted it can be repaired by invoking the individual command(s) that correct the bad data or the **ENV** command may be invoked with a **D** (Defaults) option specified. This option instructs **ENV** to update the NVRAM with defaults. The defaults are defined as follows:

Bug mode
Automatic bug self test bypassed
Execute memory tests
Maintain concurrent mode through a power cycle/reset
System memory sizing (System mode only)
Set the seven VMEchip options to defaults
No automatic SCSI bus reset
SCSI ID set to 7
Off board address set to zero
No ROMboot and ROMboot address set to start of ROM
No Autoboot
Set disk map to default
Set console port to zero and all ports use default parameters

Example 1

```
147-Bug>env;D
Update with Auto-Configuration Defaults
Update Non-Volatile RAM [Y/N] = N?(CR)
WARNING: Update(s) Discarded
147-Bug>
```

Example 2

```
147-Bug>env;D
Update with Auto-Configuration Defaults
Update Non-Volatile RAM [Y/N] = N?Y
CPU clock frequency [16,20,25,32] = 25?(CR)
Reset System [Y/N] = N?(CR)
WARNING: Updates will not be in effect until a RESET is performed
147-Bug>
```

Example 3

```
147-Bug>env;D
Update with Auto-Configuration Defaults
Update Non-Volatile RAM [Y/N] = N? Y
CPU clock frequency [16,20,25,32] = 25? (CR)
Reset System [Y/N] = N? Y
```

Firmware now takes the reset path and initializes the MVME147 with the defaults placed in NVRAM.

When **ENV** is invoked without any options you are prompted for the following modes/options:

Two modes are available:

`Bug or System environment`

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

Three Bug options are available:

`Execute/Bypass Bug Self Test`

Execute	This mode enables the extended confidence tests as defined in Appendix A. This automatically puts the Bug in the diagnostic directory.
Bypass	In this mode the extended confidence tests are bypassed, this is the mode defaulted to if NVRAM should fail.

`Execute/Bypass SST Memory Test`

Execute	This is the standard SST memory test mode, and is the one defaulted to if NVRAM should fail. In this mode the SST memory tests are executed as part of the automatic Bug self test.
Bypass	In this mode the SST memory tests are bypassed, but the board memory is zeroed to initialize parity.

`Maintain Concurrent Mode through a Power Cycle/Reset`

Yes	If Concurrent Mode is entered, a Power Cycle or Reset does not terminate the Concurrent Mode. This is the mode defaulted to if NVRAM should fail.
No	Power Cycle or Reset causes an exit from Concurrent Mode.

Three System options are available:

`Execute/Bypass System Memory Sizing`

Execute	This is the standard mode of operation, and is the one defaulted to if NVRAM should fail. In this mode the System Memory Sizing is invoked during board initialization to find the start and end of contiguous system memory.
Bypass	In this mode the System Memory Sizing is bypassed and the message `“No offboard RAM” detected` is displayed.

`Execute/Bypass SST Memory Test`

Execute	This is the standard SST memory test mode, and is the one defaulted to if NVRAM should fail. In this mode the SST memory tests are executed as part of the system self test.
Bypass	In this mode the SST memory tests are bypassed, but the system memory is zeroed to initialize parity.

`Maintain Concurrent Mode through a Power Cycle/Reset`

Yes	If Concurrent Mode is entered, a Power Cycle or Reset does not terminate the Concurrent Mode. This is the mode defaulted to if NVRAM should fail.
No	Power Cycle or Reset causes an exit from Concurrent Mode.

Seven VMEchip options are available:

`Board ID`	Allows unique board identification.
`GCSR Base Address offset`	Sets the base address of the global control and status register in the VMEbus short I/O map. This value is an offset from the start ($FFFF0000) of the map.
`Utility Interrupt Mask`	This is used to enable the VMEchip to respond to specific utility interrupt requests. Refer to the *MVME147-0xx MPU VMEmodule Installation and Use* manual for bit definitions and functional descriptions.
`Utility Interrupt Vector number`	Interrupt vector number ($8 to $F8) for the utility interrupts. Must be in multiples of $8.
`VMEbus Interrupt Mask`	This is used to enable the VMEchip to respond to specific VMEbus interrupt requests. Refer to the *MVME1-0xx MPU VMEmodule Installation and Use* manual for bit definitions and functional descriptions.
`VMEbus Requester Level`	This is used to configure the VMEbus requester level (0 through 3).
`VMEbus Requester Release`	This is used to configure the VMEbus requester release mode (Release: On Request, When Done, or Never).

Example 1

```
147-Bug>env
Bug or System environment [B,S] = B? (CR)                No change.
Execute/Bypass Bug Self Test [E,B] = B? E                Change to execute.
Execute/Bypass SST Memory Test [E,B] = E? (CR)
Maintain Concurrent Mode (if enabled) through a Power Cycle/Reset [Y/N] = Y? (CR
Set VME Chip:
Board ID(def is 0) [0-FF] = $00? (CR)
GCSR base address offset(def is 0F) [0-0F] = $0F? (CR)
Utility Interrupt Mask(def is 0) [0-FE] = $00? (CR)
Utility Interrupt Vector number(def is 60) [8-F8] = $60? 10    Change vector.
VMEbus Interrupt Mask(def is FE) [0-FE] = $FE? (CR)
VMEbus Requester Level(def is 0) [0-3] = 00? (CR)
VMEbus Requester Release(def is ROR) [ROR,RWD,NVR]=ROR? (CR)
147-Bug>
```

Example 2

```
147-Bug> ENV
Bug or System environment [B,S] = B? (CR)                 No change.
Execute/Bypass Bug Self Test [E,B] = E? B                 Change to bypass.
Maintain Concurrent Mode (if enabled) through a Power Cycle/Reset [Y/N] = Y? (CR)
Set VME Chip:
Board ID(def is 0) [0-FF] = $00? 2.                       Change and exit.
147-Bug>
```

Example 3

```
147-Bug>ENV
Bug or System environment [B,S] = B? S                    Change to system.
Execute/Bypass System Memory Sizing [E,B] = E? (CR)
Execute/Bypass SST Memory Test [E,B] = E? (CR)
Maintain Concurrent Mode (if enabled) through a Power Cycle/Reset [Y/N] = Y? (CR)
Set VME Chip:
Board ID(def is 0) [0-FF] = $02? 0                        Change and continue.
GCSR base address offset(def is 0F) [0-0F] = $0F? (CR)
Utility Interrupt Mask(def is 0) [0-FE] = $00? (CR)
Utility Interrupt Vector number(def is 60) [8-F8] = $10? (CR)
VMEbus Interrupt Mask(def is FE) [0-FE] = $FE? ^          Back up.
Utility Interrupt Vector number(def is 60) [8-F8] = $10? 60.   Change and exit.
147-Bug>
```

Firmware now takes the reset path and initializes the MVME147 for the system mode (refer to Appendix A for system mode operation details).

Go Execute Target Code - G/GO

Command Input

G/GO [*addr*]

Description

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:

1. If an address is specified, it is loaded in the target PC.
2. If a breakpoint is set at the target PC address, the instruction at the target PC is traced (executed in trace mode).
3. All breakpoints are inserted in the target code.
4. Target code execution resumes at the target PC address.

At this point control may be returned to 147Bug by various conditions:

1. A breakpoint with 0 count was found.
2. The ABORT or RESET switch on the MVME147 front panel was pressed.
3. An unexpected exception occurred.
4. The TRAP #15 .RETURN function was executed.

Example: Assume that the following program resides at $10000.

```
147-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
147-Bug>
```

Initialize D0, set breakpoints, and start target program.

```
147-Bug>RS D0 52A9C
D0  =00052A9C
```

```
147-Bug>BR 10000 1000E
BREAKPOINTS
00010000              0001000E
```

```
147-Bug>GO 10000
Effective address: 00010000
At Breakpoint
PC   =0001000E SR   =2711=TR:OFF_S._7_X...C  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
0001000E 60FE              BRA.B      $1000E
147-Bug>
```

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

Continue target program execution.

```
147-Bug>G
Effective address: 0001000E
At Breakpoint
PC   =0001000E SR   =2711=TR:OFF_S._7_X...C  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
0001000E 60FE              BRA.B       $1000E
147-Bug>
```

Remove breakpoints and restart the target code:

```
147-Bug>NOBR
BREAKPOINTS
```

```
147-Bug>GO 10000
Effective address: 00010000
```

To exit target code, press the ABORT push-button.

```
Exception: Abort
Format Vector = 0108
PC   =0001000E SR   =2711=TR:OFF_S._7_X...C  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
0001000E 60FE              BRA.B       $1000E
147-Bug>
```

Go Direct (Ignore Breakpoints) - GD

Command Input

GD [*addr*]

Description

GD 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.

The sequence of events is as follows:

1. If an address is specified, it is loaded in the target PC.
2. Target code execution resumes at the target PC.

At this point, control may be returned to 147Bug by various conditions:

1. The ABORT or RESET switch on the MVME147 front panel was pressed.
2. An unexpected exception occurred.
3. The TRAP #15 **.RETURN** function was executed.

Example: Assume that the following program resides at $10000.

```
147-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
147-Bug>
```

Initialize D0, set breakpoints, and start target program.

```
147-Bug>RS D0 52A9C
D0   =00052A9C
```

```
147-Bug> BR 10000 1000E
BREAKpoints
00010000 0001000E
```

```
147-Bug>GD 10000
Effective address: 00010000
```

Note that the breakpoints are not inserted

To exit target code, press the ABORT push-button.

```
Exception: Abort
Format Vector = 0108
PC   =0001000E SR   =2711=TR:OFF_S._7_X...C  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
0001000E 60FE              BRA.B       $1000E
147-Bug>
```

Go to Next Instruction - GN

Command Input

GN

Description

GN 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.

GN is especially helpful when debugging modular code because it allows you to “trace” through a subroutine call as if it were a single instruction.

Example: Assume that the following section of code resides at address $10000.

```
147-Bug>MD 10010:2;DI
00010000 7003          MOVE.L   #$3,D0
00010002 7201          MOVEQ.L  #$1,D1
00010004 6100000A      BSR.W    $10010
00010008 2600          MOVE.L   D0,D3
147-Bug>
```

Assume that the following simple routine resides at address $10010.

```
147-Bug>MD 10010:2;DI
00010010 D081          ADD.L   D1,D0
00010012 4E75          RTS
147-Bug>
```

Execute up to the BSR instruction.

```
147-Bug>BR 10004
BREAKPOINTS
00010004
147-Bug>
```

```
147-Bug>G 10000
Effective address: 00010000
At Breakpoint
PC   =00010004 SR   =2710=TR:OFF_S._7_X....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010004 6100000A          BSR.W      $10010
147-Bug>
```

Use the **GN** command to “trace” through the subroutine call and display the results.

```
147-Bug>GN
Effective address: 00010004
At Breakpoint
PC   =00010008 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010008 2600              MOVE.L     D0,D3
147-Bug>
```

Go to Temporary Breakpoint - GT

Command Input

GT *addr* [:*count*]

Description

GT allows you 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 147Bug by various conditions:

1. A breakpoint with 0 count was found.
2. The ABORT or RESET switch on the MVME147 front panel was pressed.
3. An unexpected exception occurred.
4. The TRAP #15 .RETURN function was executed.

Example: Assume that the following program resides at $10000.

```
147-Bug>MD 010000;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
147-Bug>
```

Initialize D0 and set a breakpoint:

```
147-Bug>RS D0 52A9C
D0  =00052A9C

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

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

```
147-Bug>RS PC 10000
PC   =00010000

147-Bug>GT 10006
Effective address: 00010006
Effective address: 00010000
At Breakpoint
PC   =00010006 SR   =2708=TR:OFF_S._7_.N...  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
D0   =00052A9C D1   =00052A9C D2   =0000009C 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
00010006 E289              LSR.L       #$1,D1
147-Bug>
```

Set another temporary breakpoint at $10006 with a *count* of 13 and continue the target program execution:

```
147-Bug>GT 10006:&13
Effective address: 00010006
Effective address: 00010006
At Breakpoint
PC   =00010006 SR   =2711=TR:OFF_S._7_X...C  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
```

```
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010006 E289              LSR.L       #$1,D1
147-Bug>
```

Set a new temporary breakpoint at $10002 and continue the target program execution:

```
147-Bug>GT 10002
Effective address: 00010002
Effective address: 00010006
At Breakpoint
PC   =0001000E SR   =2711=TR:OFF_S._7_X...C  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
0001000E 60FE              BRA.B       $1000E
147-Bug>
```

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

Help - HE

Command Input

HE [*command*]

Description

HE is the 147Bug help facility. **HE** 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.

Examples

```
147-Bug>HE
AB              Autoboot enable
NOAB            Autoboot disable
BC              Block compare
BF              Block fill
BI              Block initialize
BM              Block move
BS              Block search
BO              Boot operating system
BH              Boot operating system and halt
BR              Breakpoint insert
NOBR            Breakpoint delete
BV              Block verify
CS              Checksum
DC              Data conversion and expression evaluation
DU              Dump S-records
EEP             EEPROM programming
ENV             Set environment to Bug or operating system
GO              Go to target code
G               “Alias” for previous command
GD              Go direct (no breakpoints)
GN              Go and stop after next instruction

Press “RETURN” to continue (CR)

GT              Go and insert temporary breakpoint
HE              Help facility
```

```
IOC           I/O control
IOP           I/O to disk
IOT           I/O “teach”
LO            Load S-records
LSAD          LAN station address display/set
MA            Macro define/display
NOMA          Delete macro(s)
MAE           Macro edit
MAL           Enable macro expansion listing
NOMAL         Disable macro expansion listing
MAR           Load macros
MAW           Save macros
MD            Memory display
MM            Memory modify
M             “Alias” for previous command
MS            Memory set
MENU          System menu
OBA           Set memory address from VMEbus
OF            Offset registers
PA            Printer attach

Press “RETURN” to continue (CR)

NOPA          Printer detach
PF            Port format
NOPF          Port detach
PS            Put RTC into power save mode for storage
RB            ROMboot enable
NORB          ROMboot disable
REMOTE        Connect the remote modem to CSO
RESET         Warm/cold reset
RD            Register display
RM            Register modify
RS            Register set
SD            Switch directory
SET           Set time and date
TA            Terminal attach
T             Trace instruction
TC            Trace on change of flow
TT            Trace to temporary breakpoint
TM            Transparent mode
TIME          Display time and date
VE            Verify S-records
```

To display the command **T**, enter:

```
147-Bug>HE T
T             Trace Instruction
147-Bug>
```

I/O Control for Disk/Tape - IOC

Command Input

IOC

Description

The **IOC** command allows you 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 (CLUN) and device LUN (DLUN) 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 the MVME147 controller. Specify an "attach" operation to the hard disk.

```
147-Bug>IOC
Controller LUN  =00? (CR)
Device LUN      =00? (CR)
Packet address  =000014DC? 10000
00010000 0000 0000 0000 0000 0000 0D00 0000 0000  .0.....N........
00010010 0030 1000 0008 004E 0000 0005 0000 0000  ................
00010020 0001 0100 0000 0000 0000 0000 0000 0000  ................
00010030 0000 0000 0000 0000 0000 0000 0000 0000  ................
Send Packet (Y/N)? Y
147-Bug>
```

I/O Physical (Direct Disk/Tape Access) - IOP

Command Input

IOP

Description

The **IOP** command allows you to read, write, or format any of the supported disk or tape devices. When invoked, this command goes into an interactive mode, prompting you for all the parameters necessary to carry out the command. You 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 you 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 for which you are prompted is as follows:

```
Controller LUN              =00?
```

The Logical Unit Number (LUN defined by the **IOT** command) of the controller to access is specified in this field.

```
Device LUN                  =00?
```

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

```
Read/Write/Format           =R?
```

In this field, you specify the desired function by entering a one-character mnemonic as follows:

R	Read. This reads blocks of data from the selected device into memory.
W	Write. This writes blocks of data from memory to the selected device.
F	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.

For read/write operations, the prompts are as follows:

```
Memory Address                =00004000?
```

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

```
Starting Block                =00000000?
```

For disk (direct access) devices, this field specifies the starting block number to access. For read operations, data is read starting at this block. For write operations, data is written starting at this block.

```
File Number                   =00000000?
```

For tape (sequential access) devices, this field specifies the starting file number to access.

```
Number of Blocks              =0002?
```

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

```
Flag Byte                     =00?
```

For tape devices, this field 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 tape devices use this field.

The currently defined bits are as follows:

Bit 7	Filemark flag. If 1, a filemark was detected at the end of the last operation.
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 (EOF) 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 filemark is found. If 1, writes are terminated with a filemark.

```
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 of $0D is selected by the driver.

If a nonzero value is specified, it is used by the driver for data transfers.

For format operations, the prompts are as follows:

```
Starting Block                =00000000?
```

If the device supports track formatting, this field specifies the track that contains this block is to be formatted.

```
Track/Disk                    =T (T/D)?
```

If the device supports track formatting, this field specifies whether a disk track or the entire disk is formatted when the format operation is selected.

```
Retension/Erase               =R (R/E)?
```

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

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: From a disk device read 25 blocks, starting at block 370 into memory beginning at address $50000. For this example, assume the drive is device 2 of controller 0.

```
147-Bug>IOP
Controller LUN   =00? (CR)
Device LUN       =00? 2
Read/Write/Format=R? (CR)
Memory  Address  =00004000? 50000
Starting Block   =00000000? &370
Number of Blocks =0002? &25
Address Modifier =00? (CR)
147-Bug>
```

Example 2: To a tape device write 14 blocks, starting at memory location $7000 to file 6 and append a filemark at the end of the file. For this example, assume the drive is device 0 of controller 4.

```
147-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)
147-Bug>
```

Example 3: Formatting a disk device, at track that contains block 6. For this example, assume the drive is device 2 of controller 0.

Caution

On devices that support track formatting, this destroys all previous data on the selected track.

On devices that do not support track formatting, this can destroy all previous data on the whole device.

```
147-Bug>IOP
Controller LUN   =04? 0
Device LUN       =00? 2
Read/Write/Format=R? F
Starting Block   =00000006? 0
Track/Disk       =D (T/D)? T
147-Bug>
```

Example 4: Erase a tape device. For this example assume the drive is device 0 of controller 4.

Caution

This completely clears the tape of previous data.

Caution

This completely clears the tape of previous data.

```
147-Bug>IOP
Controller LUN   =00? 4
Device LUN       =02? 0
Read/Write/Format=F? (CR)
Retension/Erase  =R (R/E)? E
147-Bug>
```

I/O Teach for Configuring Disk Controller - IOT

Command Input

IOT [;[A][H][T]]

Options

A	All. List all disk controllers supported by 147Bug.
H	Help. List all disk controllers available to the system.
T	Teach. Probe the system for I/O controllers and build a table of the available controllers.

Description

The **IOT** command allows you to “teach” a new disk configuration to 147Bug for use by the TRAP #15 disk functions. **IOT** lets you modify the controller and device descriptor tables used by the TRAP #15 functions for disk access. Note that because 147Bug 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.

Note that during the first **IOP** command and during a boot, **IOT** is not required. Reconfiguration is done automatically by reading the configuration sector from the device, then the device descriptor table for the LUN used is modified accordingly.

If the device is not formatted or is of unknown format, or has no configuration sector, then before attempting to access the device with the **IOP** command, you should verify the parameters using **IOT** and, if necessary, modify them for the specific media and device.

When the **IOT** command is invoked without options or with a **T** (teach) option, an interactive mode is entered. While in the interactive mode, the following rules apply:

All numerical values are interpreted as hexadecimal numbers. Decimal values may be entered by preceding the number with an ampersand (&).

Only listed values are accepted when a list is shown. Uppercase or lowercase may be interchangeably used when a list is shown.

^	Back up to previous field.
=	Reopen same field.
.	Entering a period by itself or following a new value/setting causes **IOT** to exit the interactive mode. Control returns to the Bug.
(CR)	Pressing **Return** (**Enter**) without entering a value preserves the current value and causes the next prompt to be displayed.

Examples: A and **H** options.

```
147-Bug> IOT;A
           Disk Controllers Supported
    Type        Address      # dev
  VME147     $FFFE4000        *    SCSI - 0-7
  VME327     $FFFFA600        *    SCSI - 0-7
  VME327     $FFFFA600        2
  VME327     $FFFFA700        *    SCSI - 0-7
  VME327     $FFFFA700        2
  VME321     $FFFF0500        8
  VME320     $FFFFB000        4
  VME319     $FFFF0000        8
  VME321     $FFFF0600        8
  VME360     $FFFF0C00        4
  VME360     $FFFF0E00        4
  VME350     $FFFF5000        1
  VME350     $FFFF5100        1
  VME320     $FFFFAC00        4
  VME319     $FFFF0200        8
  VME323     $FFFFA000        4
  VME323     $FFFFA200        4
```

```
147-Bug> IOT;H
       Disk Controllers Available
LUN Type   Address    #dev
0   VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
1   VME147 $FFFE4000   1  SCSI Addr= 1  MICROP   1375
2   VME147 $FFFE4000   1  SCSI Addr= 2  CDC      94171-9
3   VME147 $FFFE4000   1  SCSI Addr= 3  SEAGATE  ST296N/M
4   VME147 $FFFE4000   1  SCSI Addr= 4  ARCHIVE  VIPER 60  21116
5   VME147 $FFFE4000   1  SCSI Addr= 5  ARCHIVE  VIPER 60  21116
6   VME147 $FFFE4000   4  SCSI Addr= 6  SMS      OMTI7000
7   VME320 $FFFFB000   4
8   VME350 $FFFF5000   1
    VME147 $FFFE4000   *  SCSI Addr= 7
147-Bug>
```

IOT may be invoked with a **T** (teach) option specified. This option instructs **IOT** to scan the system for all currently supported disk/tape controllers and build a map of the available controllers. This map is built in the Bug RAM area, but can also be saved in NVRAM if so instructed.

The **IOT;T** command should be invoked any time the controllers are changed or whenever the NVRAM map has been damaged ("No Disk Controllers Available"). The reason for this is that, during a reset, the map residing in NVRAM is copied to the Bug RAM area and used as the working map.

Example: T option.

```
147-Bug> IOT;T
Scanning system for available disk/tape controllers . . .
       Disk Controllers Available

LUN Type   Address    #dev
0   VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
1   VME147 $FFFE4000   1  SCSI Addr= 1  MICROP   1375
2   VME147 $FFFE4000   1  SCSI Addr= 2  CDC      94171-9
3   VME147 $FFFE4000   1  SCSI Addr= 3  SEAGATE  ST296N/M
4   VME147 $FFFE4000   1  SCSI Addr= 4  ARCHIVE  VIPER 60  21116
```

```
LUN  Type   Address    #dev
5    VME147 $FFFE4000   4   SCSI Addr= 6  SMS      OMTI7000
6    VME320 $FFFFB000   4
7    VME350 $FFFF5000   1
     VME147 $FFFE4000   *   SCSI Addr= 7

147-Bug>
Align LUNs to SCSI addresses [Y,N] N? Y
       Disk Controllers Available
LUN  Type   Address    #dev
0    VME147 $FFFE4000   1   SCSI Addr= 0  CDC      94161-9
1    VME147 $FFFE4000   1   SCSI Addr= 1  MICROP   1375
2    VME147 $FFFE4000   1   SCSI Addr= 2  CDC      94171-9
3    VME147 $FFFE4000   1   SCSI Addr= 3  SEAGATE  ST296N/M
4    VME147 $FFFE4000   1   SCSI Addr= 4  ARCHIVE  VIPER 60  21116
5    VME147 $FFFE4000   1   SCSI Addr= 5
6    VME147 $FFFE4000   4   SCSI Addr= 6  SMS      OMTI7000
8    VME320 $FFFFB000   4
9    VME350 $FFFF5000   1
     VME147 $FFFE4000   *   SCSI Addr= 7

Save map in NVRAM [Y,N] N? Y
147-Bug>
```

When invoked without options, the **IOT** command enters an interactive subcommand mode where you can edit the disk map or the descriptor table values currently in effect.

The disk map editor may be invoked with a **Y** (yes) response to the prompt.

```
147-Bug> IOT
Edit Disk Map [Y,N] N? Y
       Disk Controllers Available
LUN  Type   Address    #dev
0    VME147 $FFFE4000   1   SCSI Addr= 0  CDC      94161-9
1    VME147 $FFFE4000   1   SCSI Addr= 1  MICROP   1375
```

```
LUN  Type    Address    #dev
2    VME147  $FFFE4000   1   SCSI Addr= 2  CDC      94171-9
3    VME147  $FFFE4000   1   SCSI Addr= 3  SEAGATE  ST296N/M
4    VME147  $FFFE4000   1   SCSI Addr= 4  ARCHIVE  VIPER 60  21116
5    VME147  $FFFE4000   1   SCSI Addr= 5
6    VME147  $FFFE4000   4   SCSI Addr= 6  SMS      OMTI7000
8    VME320  $FFFFB000   4
9    VME350  $FFFF5000   1
     VME147  $FFFE4000   *   SCSI Addr= 7

Disk Map edit commands:
   C    -Copy
   E    -Edit
   M    -Move
   R    -Remove

                 =E? C
Controller LUN   =00? 0
Before or After [B,A] =A? (CR)
Controller LUN   =00? 4

       Disk Controllers Available

LUN  Type    Address    #dev
0    VME147  $FFFE4000   1   SCSI Addr= 0  CDC      94161-9
1    VME147  $FFFE4000   1   SCSI Addr= 1  MICROP   1375
2    VME147  $FFFE4000   1   SCSI Addr= 2  CDC      94171-9
3    VME147  $FFFE4000   1   SCSI Addr= 3  SEAGATE  ST296N/M
4    VME147  $FFFE4000   1   SCSI Addr= 4  ARCHIVE  VIPER 60  21116
5    VME147  $FFFE4000   1   SCSI Addr= 0  CDC      94161-9
6    VME147  $FFFE4000   1   SCSI Addr= 5
7    VME147  $FFFE4000   4   SCSI Addr= 6  SMS      OMTI7000
9    VME320  $FFFFB000   4
A    VME350  $FFFF5000   1
     VME147  $FFFE4000   *   SCSI Addr= 7

Quit options:
   E    -Edit (edit another LUN)
   Q    -Quit
   S    -Save in NVRAM and quit
```

Create a copy of an LUN after another LUN.

```
                    =Q? E        Edit another LUN

       Disk Controllers Available

LUN  Type   Address   #dev
0    VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
1    VME147 $FFFE4000   1  SCSI Addr= 1  MICROP   1375
2    VME147 $FFFE4000   1  SCSI Addr= 2  CDC      94171-9
3    VME147 $FFFE4000   1  SCSI Addr= 3  SEAGATE  ST296N/M
4    VME147 $FFFE4000   1  SCSI Addr= 4  ARCHIVE  VIPER 60  21116
5    VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
6    VME147 $FFFE4000   1  SCSI Addr= 5
7    VME147 $FFFE4000   4  SCSI Addr= 6  SMS      OMTI7000
9    VME320 $FFFFB000   4
A    VME350 $FFFF5000   1
     VME147 $FFFE4000   *  SCSI Addr= 7

Disk Map edit commands:

   C    -Copy
   E    -Edit
   M    -Move
   R    -Remove

                    =C? M        Move a LUN before another LUN
Controller LUN   =04? 6
Before or After [B,A] =A? B
Controller LUN   =06? 0

       Disk Controllers Available

LUN  Type   Address   #dev
0    VME147 $FFFE4000   1  SCSI Addr= 5
1    VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
2    VME147 $FFFE4000   1  SCSI Addr= 1  MICROP   1375
3    VME147 $FFFE4000   1  SCSI Addr= 2  CDC      94171-9
4    VME147 $FFFE4000   1  SCSI Addr= 3  SEAGATE  ST296N/M
5    VME147 $FFFE4000   1  SCSI Addr= 4  ARCHIVE  VIPER 60  21116
6    VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
7    VME147 $FFFE4000   4  SCSI Addr= 6  SMS      OMTI7000
9    VME320 $FFFFB000   4
A    VME350 $FFFF5000   1
     VME147 $FFFE4000   *  SCSI Addr= 7
```

```
Quit options:
   E    -Edit (edit another LUN)
   Q    -Quit
   S    -Save in NVRAM and quit
                 =Q? E
       Disk Controllers Available
LUN Type   Address    #dev
0   VME147 $FFFE4000   1  SCSI Addr= 5
1   VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
2   VME147 $FFFE4000   1  SCSI Addr= 1  MICROP   1375
3   VME147 $FFFE4000   1  SCSI Addr= 2  CDC      94171-9
4   VME147 $FFFE4000   1  SCSI Addr= 3  SEAGATE  ST296N/M
5   VME147 $FFFE4000   1  SCSI Addr= 4  ARCHIVE  VIPER 60  21116
6   VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
7   VME147 $FFFE4000   4  SCSI Addr= 6  SMS      OMTI7000
8   VME320 $FFFFB000   4
9   VME350 $FFFF5000   1
    VME147 $FFFE4000   *  SCSI Addr= 7
Disk Map edit commands:
   C    -Copy
   E    -Edit
   M    -Move
   R    -Remove
                 =M? R      Remove an LUN.
Controller LUN   =00? 0
       Disk Controllers Available
LUN Type   Address    #dev
0   VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
1   VME147 $FFFE4000   1  SCSI Addr= 1  MICROP   1375
2   VME147 $FFFE4000   1  SCSI Addr= 2  CDC      94171-9
3   VME147 $FFFE4000   1  SCSI Addr= 3  SEAGATE  ST296N/M
4   VME147 $FFFE4000   1  SCSI Addr= 4  ARCHIVE  VIPER 60  21116
5   VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
6   VME147 $FFFE4000   4  SCSI Addr= 6  SMS      OMTI7000
8   VME320 $FFFFB000   4
```

```
LUN  Type    Address    #dev
9    VME350  $FFFF5000    1
     VME147  $FFFE4000    *   SCSI Addr= 7

Quit options:

E        -Edit (edit another LUN)
Q        -Quit
S        -Save in NVRAM and quit

                  =Q? E

       Disk Controllers Available

LUN  Type    Address    #dev
0    VME147  $FFFE4000    1   SCSI Addr= 0  CDC       94161-9
1    VME147  $FFFE4000    1   SCSI Addr= 1  MICROP    1375
2    VME147  $FFFE4000    1   SCSI Addr= 2  CDC       94171-9
3    VME147  $FFFE4000    1   SCSI Addr= 3  SEAGATE   ST296N/M
4    VME147  $FFFE4000    1   SCSI Addr= 4  ARCHIVE   VIPER 60  21116
5    VME147  $FFFE4000    1   SCSI Addr= 0  CDC       94161-9
6    VME147  $FFFE4000    4   SCSI Addr= 6  SMS       OMTI7000
8    VME320  $FFFFB000    4
9    VME350  $FFFF5000    1
     VME147  $FFFE4000    *   SCSI Addr= 7

Disk Map edit commands:

   C     -Copy
   E     -Edit
   M     -Move
   R     -Remove

                 =R? E     Edit an LUN.

Controller LUN   =00? 5
SCSI device [Y,N] =Y? Y
Controller type    = 0147? (CR)
Controller address = $FFFE4000? (CR)
SCSI address (0-7) = 00? 5
SCSI Controller Type:

 D - (147)     Teac Floppy
 E - (147)     Omti (3500/7x00)
 F - (147)     Common Command Set (Win/Floppy)
 F - (327)     Common Command Set (Win)
10 - (All)     CDC (Wren III & Swift)
11 - (All)     Micropolis 1375
12 - (All)     Archive Viper, Teac Tape
13 - (All)     CDC (Wren IV & V), Maxtor 8760
14 - (All)     Seagate
15 - (327)     Common Command Set Rev. 4A (Win)
16 - (All)     Kennedy, HP 1/2" Tape
17 - (147)     Sync Common Command Set (Win/Floppy)
17 - (327)     Sync Common Command Set (Win)
18 - (All)     Exabyte Tape
19 - (All)     IBM
1A - (327)     SONY
                     =10? (CR)

Number of supported devices = 1
DLUN 0 is a Fixed Disk Device

       Disk Controllers Available

LUN  Type   Address    #dev
0    VME147 $FFFE4000   1  SCSI Addr= 0  CDC      94161-9
1    VME147 $FFFE4000   1  SCSI Addr= 1  MICROP   1375
2    VME147 $FFFE4000   1  SCSI Addr= 2  CDC      94171-9
3    VME147 $FFFE4000   1  SCSI Addr= 3  SEAGATE  ST296N/M
4    VME147 $FFFE4000   1  SCSI Addr= 4  ARCHIVE  VIPER 60  21116
5    VME147 $FFFE4000   1  SCSI Addr= 5
6    VME147 $FFFE4000   4  SCSI Addr= 6  SMS      OMTI7000
8    VME320 $FFFFB000   4
9    VME350 $FFFF5000   1
     VME147 $FFFE4000   *  SCSI Addr= 7

Quit options:
   E    -Edit (edit another LUN)
   Q    -Quit
   S    -Save in NVRAM and quit
          =Q? S       Save in NVRAM and quit.
147-Bug>
```

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 screen for you to examine. You may change the displayed value by entering a new value or leave it unchanged.

The first two items of information that you are 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, **IOT** outputs the message "`Invalid LUN`" and you are prompted for the two LUNs again.

```
147-Bug>IOT
Edit Disk Map [Y,N] N? (CR)
Controller LUN     = 00? (CR)
Device LUN         = 00? (CR)
Controller type    = VME147
Controller address = $FFFE4000? (CR)
VME147 Controller SCSI address (0-7) = 07? (CR)        SCSI only.
SCSI Controller Type:                                   SCSI only.

 D -(147)     Teac Floppy
 E -(147)     Omti (3500/7x00)
 F -(147)     Common Command Set (Win/Floppy)
 F -(327)     Common Command Set (Win)
10 -(All)     CDC (Wren III & Swift)
11 -(All)     Micropolis 1375
12 -(All)     Archive Viper, Teac Tape
13 -(All)     CDC (Wren IV & V), Maxtor 8760
14 -(All)     Seagate
15 -(327)     Common Command Set Rev. 4A (Win)
16 -(All)     Kennedy, HP 1/2" Tape
17 -(147)     Sync Common Command Set (Win/Floppy)
17 -(327)     Sync Common Command Set (Win)
18 -(All)     Exabyte Tape
19 -(All)     IBM
1A -(327)     SONY

        =10? (CR)
```

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 you 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:

```
Sector Size:
0-128 1-256
2-512 3-1024     =01?
```

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

```
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) = integer*

```
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.

```
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.

```
Number of Heads       =05?
```

This field specifies the number of heads on the drive.

```
Number of Cylinders   =0337?
```

This field specifies the number of cylinders on the device. For floppy disks, the number 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

```
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.

```
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.

```
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.

```
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.

```
ECC Data Burst Length =0000?
```

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

```
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 (Hexadecimal	Winchester Hard Disks	Slow Data Rate	Fast Data Rate
00	0 ms	12 ms	6 ms
01	6 ms	6 ms	3 ms
02	10 ms	12 ms	6 ms
03	15 ms	20 ms	10 ms
04	20 ms	30 ms	15 ms

```
Single/Double DATA Density =D (S/D)?
```

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

```
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

```
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.

```
Slow/Fast Data Rate   =S (S/F)?
```

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

S = 250 kHz data rate (5-1/4 inch floppy, usually)
F = 500 kHz data rate (8-inch, 3-1/2 inch floppy, usually)

```
Gap 1                 =07?
```

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

```
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.

```
Gap 3                 =00?
```

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

```
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.

```
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.

```
147-Bug>IOT
Edit Disk Map [Y,N] N? (CR)
Controller LUN        =00? 8
Device LUN            =00? 2
Controller type       =VME320
Controller address    =$FFFFB000? (CR)
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)
147-Bug>
```

Example 2: Changing from a 40MB Winchester to a 70MB Winchester. (Note that reconfiguration such as this is only necessary when the device is not formatted or of an unknown format, or has no configuration sector. Reconfiguration is normally done automatically by the **IOP**, **BO**, or **BH** commands.

```
147-Bug>IOT
Edit Disk Map [Y,N] N? (CR)
Controller LUN       =00? 8
Device LUN           =00? (CR)
Controller type      =VME320
Controller address   =$FFFFB000? (CR)
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)
147-Bug>
```

Load S-Records from Host - LO

Command Input

LO [*port*] [*addr*] [**;x | -c | t**] [=*text*]

Options

-c	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, the comparison is not made.
x	Echo. Echoes the S-records to your terminal as they are read in at the host port.
t	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 147Bug, and they set a different code in D4.

Description

This 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 MVME147. 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 "*port*" allows you 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 you 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 than 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 147Bug begins to look for S-records at the host port. This allows you 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 your 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 line feed (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 an (LF) character before data records are processed.

For this reason, in situations where the host system does not echo characters, it is required 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 S-record format (refer to Appendix C) allows for 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, you need only enter **G** or **GO** instead of **G** *addr* or **GO** *addr* to execute the code that was downloaded.

If a nonhexadecimal character is encountered within the data field of a data record, the part of the record which had been received up to that time is printed to the screen and the 147Bug 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 147Bug AND if the checksum comparison has not been disabled via the "**-c**" option, 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, 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 has already been stored to memory if the command aborts due to an error.

Examples

Suppose a host system was used to create this program:

```
1                        *  Test Program.
2                        *
3           65040000                    ORG          $65040000
4
5    6504000 7001                       MOVEQ.L      #$1,D0
6    6504002 D088                       ADD.L        A0,D0
7    6504004 4A00                       TST.B        D0
8    6504006 4E75                       RTS
9                                       END
******  TOTAL ERRORS     0--
******  TOTAL WARNINGS   0--
```

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

```
S00F000054455354533335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
```

Load this file into MVME147 memory for execution at address $40000 as follows:

```
147-Bug>TM                      Go into transparent mode to establish
Escape character: $01= ^        communication with the host.
   :
(CR)                            Press RETURN or ENTER to get login
   :                            prompt.
(login)                         You must log onto the host and enter
   :                            the proper directory to access the file
   :                            TEST.MX.
= ^A                            Enter the escape character (CTRL A)
147-Bug>                        to return to the 147Bug prompt.
```

```
147-Bug>LO -65000000 ;x=cat TEST.MX,#
cat TEST.MX,#
S00F000054455354533335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
147-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 FILE.MX and caused the program to be loaded to memory starting at $40000. The text "cat TEST.MX" is the host system command line that caused the file to be copied by the host to the port which is connected with the MVME147 host port.

```
147-Bug>TM                      Go into transparent mode again.
Escape character: $01= ^A
   :
# d                             At the prompt, log off the system.
    :
login: ^A                       Enter the escape character (CTRL A)
147-Bug>                        to return to the 147Bug prompt.
```

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

LAN Station Address Display/Set - LSAD

Command Input

LSAD

Description

The **LSAD** command is used for examining and updating the Ethernet station address.

Every MVME147 with LAN support is assigned an Ethernet station address. The address is $08003E2*xxxxx*, where *xxxxx* is the unique number assigned to the module; i.e., every MVME147 has a different value for *xxxxx*.

Each Ethernet station address is displayed on a label attached to the backplane connector P2. In addition, the *xxxxx* portion of the Ethernet station address is stored in BBRAM location $FFFE0778 as $2*xxxxx*.

If Motorola networking software is running on an MVME147, it uses the 2*xxxxx* value from BBRAM to complete the Ethernet station address ($08003E2*xxxxx*). The user must assure that the value of 2*xxxxx* is maintained in BBRAM. If the value of 2*xxxxx* is lost in BBRAM, you should use the number on the P2 connector label to restore it.

Example 1: Display Ethernet station address.

```
147-Bug> LSAD
LAN Station Address = $08003E200000
To set the Station Address:
   Enter the code located on the back of the front panel:
$08003E2_____(CR)
147-Bug>
```

Example 2: Change Ethernet station address.

```
147-Bug> LSAD
LAN Station Address = $08003E200000
To set the Station Address:
   Enter the code located on the back of the front panel:
$08003E2_____1
LAN Station Address = $08003E200001
147-Bug>
```

Macro Define/Display/Delete - MA/NOMA

Command Input

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

Arguments

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

Description

The **MA** command allows you to define a complex command consisting of any number of debugger 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 debugger 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 debugger enters the macro definition mode. In response to each macro definition prompt "`M=`", enter a debugger 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 debugger commands; i.e., no definition, is 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 you are returned to the debugger 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 either to delete or edit macro(s).

Debugger 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. Similarly, the second argument would be used whenever the sequence "**\1**" occurred.

Thus, 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 macro name that does not have a definition, an error message is printed.

Examples

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

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

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

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

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

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

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

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

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

Macro Edit - MAE

Command Input

MAE *name line#* [*string*]

Arguments

name	Any combination of 1 through 8 alphanumeric characters.
line#	Line number in range 1 through 999.
string	Replacement line to be inserted.

Description

The **MAE** command permits modification of the macro named in 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 displayed. **MAE** does not permit deletion of a line if the macro consists only of 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

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

147-Bug>  MAE ABC 15 RD              Add a line to macro ABC.
MACRO ABC
010 MD 3000
020 RD                               This line was inserted.
030 GO \0
147-Bug>
147-Bug>  MAE ABC 10 MD 10+R0        Replace line 10.
MACRO ABC
010 MD 10+R                          This line was overwritten.
020 RD
030 GO \0
147-Bug>
147-Bug>  MAE ABC 30                 Delete line 30.
MACRO ABC
010 MD 10+R0
020 RD
147-Bug>
```

Enable/Disable Macro Expansion Listing - MAL/NOMAL

Command Input

MAL
NOMAL

Description

The **MAL** command allows you 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 you and in no way interacts with the function of the macros.

Save/Load Macros - MAW/MAR

Command Input

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

Arguments

controller LUN	This is the logical unit number of the controller to which the following device is attached. Initially defaults to LUN 0.
device LUN	This is the logical unit number of the device to save/load macros to/from. Initially defaults to LUN 0.
block #	This is the number of the block on the above device that is the first block of the macro list. Initially defaults to block 2.

Description

The **MAW** command allows you 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)?`). You 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 to 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 displayed.

The **MAR** command allows you to load macros that have previously been 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** check 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**, 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 I/O routines.

Examples: Assume that controller 0, device 2 is accessible**.**

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

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

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

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

147-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.
147-Bug>
```

Memory Modify - M/MM

Command Input

MM *addr*[**;**[[**b** | **w** | **l** | **s** | **d** | **x** | **p**][**a**][**n**]] | [**di**]]
M *addr*[**;**[[**b** | **w** | **l** | **s** | **d** | **x** | **p**][**a**][**n**]] | [**di**]]

Options

MM accepts the following data types:

Integer Data Type		**Floating-Point Data Type**	
b	Byte	**s**	Single Precision
w	Word (default)	**d**	Double Precision
l	Longword	**x**	Extended Precision
		p	Packed Precision

The **n** option of the **MM** command disables the read portion of the command. The **a** option forces alternate location accesses only.

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 you are prompted with a question mark ("?") for input. At this point, you have three options:

1. Enter (**CR**). This closes the present location and continues with disassembly of next instruction.
2. Enter a new source instruction followed by (**CR**). This invokes the assembler, which assembles the instruction and generates a "listing file" of one instruction.
3. 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 linefeed 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.

Description

The **M** or **MM** command is used to examine and change memory locations.

The **MM** command (alternate form **M**) reads and displays the contents of memory at the specified address and prompts you with a question mark ("?"). You may enter new data for the memory location, followed by **CR**, or you 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.

You 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. However, these special characters cannot be used if the **di** option is selected. They 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 147Bug.

Example 1

```
147-Bug>M 10000                    Access location 10000.
00010000 1234? (CR)
00010002 5678?  4321               Modify memory.
00010004 9ABC?  8765^              Modify memory and back up.
00010002 4321?  (CR)
00010000 1234?  abcd.              Modify memory and exit.
```

Example 2

```
147-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 examples below were made in the hardcopy mode.

Example 3: Assemble a new source line.

```
147-Bug>MM 10000;di
00010000 46FC2400            MOVE.W  $2400,SR ? divs.w -(A2),D2
00010000 85E2                DIVS.W  -(A2),D2
00010002 2400                MOVE.L  D0,D2 ? (CR)
147-Bug>
```

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 ? (CR)
147-Bug>
```

Example 5: Step to next location and exit **MM**.

```
147-Bug>M 1000C;di
FFE1000C 000000FF            OR.B    #255,D0 ? (CR)
FFE10010 20C9                MOVE.L  A1,(A0)+ ? .
147-Bug>
```

Example 6

```
147-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_F000000000000000?(CR)
00007000 1_3C10_8478200000000000?.
147-Bug>
```

Memory Display - MD

Command Input

MD[**s**]*addr*[**:***count* | *addr*][**;** [**b** | **w** | **l** | **s** | **d** | **x** | **p**] | [**di**]]

Arguments

s	The optional sector modifier **s**, appended to the **MD** command, changes the default *count* to 128.
count	The optional *count* argument specifies the number of data items to be displayed (or the number of disassembled instructions to display, if the disassembly option is selected). The default is 8 if no *count* is entered and the **s** (sector) modifier is not used.

Options

MD accepts the following data types:

Integer Data Type		Floating-Point Data Type	
b	Byte	**s**	Single Precision
w	Word (default)	**d**	Double Precision
l	Longword	**x**	Extended Precision
		p	Packed Precision

For the integer data types, the data is always displayed in hexadecimal along with its ASCII representation.

The **di** option enables the resident MC68030 disassembler. No other option is allowed if **di** is selected.

Description

This command is used to display the contents of multiple memory locations all at once. 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.

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

Example 1

```
147-Bug>MD 12000
00012000 2800 1942 2900 1942  2800 1842 2900 2846  (..B)..B(..B).(F
147-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.

```
147-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.
147-Bug>
```

Example 3: Disassemble eight instructions, starting at $50008

```
147-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
147-Bug>
```

Example 4: To display eight double precision floating point numbers at location 50008, the user enters the following command line.

```
147-Bug>MD 50008;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 0_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
147-Bug>
```

Menu - MENU

Command Input

MENU

Description

The **MENU** command works only if the 147Bug is in the "system" mode (refer to the **ENV** command). When invoked in the system mode, it provides a way to exit 147Bug and return to the menu.

When the 147Bug is in system mode, you 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 `147-Bug>` or `147-Diag>` prompt to return to the system menu.

For details on use of the system mode menu features, refer to Appendix A.

Example

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

```
147-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 #:
```

Memory Set - MS

Command Input

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

Arguments

addr	Refer to Chapter 2 for use of a function code as part of the *addr* field.
hexadecimal number	Hexadecimal 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 unchanged.
string	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.

Description

Memory Set is used to write data to memory starting at the specified address.

Example: Assume that memory is initially cleared.

```
147-Bug>MS 25000 0123456789abcDEF 'This is "147Bug"' 23456
147-Bug>MD 25000:10
00025000 0123 4567 89AB CDEF  5468 6973 2069 7320   .#Eg.+MoThis is.
00025010 2731 3437 4275 6727  2345 6000 0000 0000   '147Bug'#E`.....
147-Bug>
```

Set Memory Address from VMEbus - OBA

Command Input

OBA

Description

The **OBA** (Off-Board Address) command allows you to set the base address of the MVME147 onboard RAM, as seen from the VMEbus (refer to Chapter 1). Therefore, you should enter the hexadecimal number corresponding to the actual base address, so that the off-board external devices on the VMEbus will know where it is. The default condition is with the off-board address set to $0. These selections are stored in the BBRAM that is part of the MK48T02 (RTC), and remain in effect through power-up or any reset.

Example 1: Display base addresses for 8MB board.

```
147-Bug>OBA
RAM address from VMEbus = $00000000? 1234          Note 1

Base addresses are:  $00000000, $00800000, $01000000, $01800000,
                     $02000000, $02800000, $03000000, $03800000,
                     $04000000, $04800000, $05000000, $05800000,
                     $06000000, $06800000, $07000000, $07800000,
                     $08000000, $08800000, $09000000, $09800000,
                     $0A000000, $0A800000, $0B000000, $0B800000,
                     $0C000000, $0C800000, $0D000000, $0D800000

RAM address from VMEbus = $00000000? (CR)          Note 2
147-Bug>
```

Example 2: Change base address for 8MB board.

```
147-Bug>OBA
RAM address from VMEbus = $00000000? 800000        Note 3
147-Bug>
```

Example 3: Display/change base address for 16MB board.

```
147-Bug>OBA
RAM address from VMEbus = $00000000? 1234

Base addresses are: $00000000, $01000000, $02000000, $03000000,
                     $04000000, $05000000, $06000000, $07000000,
                     $08000000, $09000000, $0A000000, $0B000000,
                     $0C000000, $0D000000, $0E000000, $0F000000,
                     $10000000, $11000000, $12000000, $13000000,
                     $10000000, $15000000, $16000000, $17000000,
                    $10000000, $19000000, $1A000000, $1B000000
Base Address options: 1, 2

16/32 Mbyte Extended/Standard Addressing options available:

1 = Extended - $00000000-$00FFFFFF,  Standard - $000000-$7FFFFF
2 = Extended - $01000000-$01FFFFFF,  Standard - $000000-$7FFFFF

RAM address from VMEbus = $00000000? 2                       Note 4
147-Bug>
```

Example 4: Change base address without option.

```
147-Bug>OBA
RAM address from VMEbus (option 2) = $01000000? 0            Note 5
147-Bug>
```

Notes

1. Any value that is not a base address or option, displays the base addresses for the board based on the onboard RAM size.
2. Pressing return without entering an address preserves the current address.
3. Change base address from $0 to $800000.
4. Select option 2, onboard RAM responds to extended addresses from $01000000 to $01FFFFFF, and standard addresses from $000000 to $7FFFFF.
5. Return the base address to the default address of $0. Onboard RAM responds to extended addresses from $0 to end of onboard RAM, and standard addresses from $0 to $FFFFFF.

Offset Registers Display/Modify - OF

Command Input

OF [**R***n*[;**A**]]

Options

R*n*	Register to be modified.
A	Denotes automatic register.

Description

OF allows you 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 is used for the range declared by the offset register. The top address is the absolute greatest address that is used. When entering the base and top, you 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. Wrap-around is not permitted.

Command Usage

OF	Display all offset registers. An asterisk indicates which register is the automatic register.
OF R*n*	Display/modify **R***n*. You can scroll through the register in a way similar to that used by the **MM** command.

OF R*n*;A	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 the **RM** and **MM** commands.
range syntax	[*base address* [*del top address*]] [**^** | **v** | **=** | **.**] or [*base address* [**:** *byte count*]] [**^** | **v** | **=** | **.**]

Offset Register Rules

1. At power-up and cold start reset, R7 is the automatic register.
2. At power-up and cold start reset, all offset registers have both base and top addresses preset to 0. This effectively disables them.
3. R7 always has both base and top addresses set to 0; it cannot be changed.
4. Any offset register can be set as the automatic register.
5. The automatic register is always added to every absolute address argument of every 147Bug command where there is not an offset register explicitly called out.
6. 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.

```
147-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
147-Bug>
```

Modify some offset registers.

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

Look at location $20000.

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

Set R0 as the automatic register.

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

Look at location $20000.

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

Look at location 0, override the automatic offset.

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

Printer Attach/Detach - PA/NOPA

Command Input

PA [*port*]
NOPA [*port*]

Argument

port Port number.

Description

These two commands “attach” or “detach” a printer to the 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 an “unassigned” message. If **NOPA** is attempted on a port that is not currently attached, an “unassigned” message is displayed.

The port being attached must already be configured. This is done using the Port Format (**PF**) command, and executing the following sequence prior to “PA *port*”.

```
147-Bug>PF4
Logical unit $04 unassigned
Name of board? VME147
Name of port? PTR
Port base address = $FFFE2800? (CR)
DTE, DCE, or Printer [T,C,P] = P? (CR)
Auto Line Feed protocol [Y,N] = N? Y.
OK to proceed (y/n)? Y
147-Bug>
```

For further details, refer to the **PF** command.

Examples

Console display: **Printer output:**

```
147-Bug>PA4                                    Attach printer port 4
147-Bug>HE NOPA      147-Bug>HE NOPA
NOPA    Printer Detach   NOPA    Printer Detach
147-Bug>NOPA         147-Bug>NOPA      Detach all printers
147-Bug>NOPA
No printer attached
147-Bug>
```

Port Format/Detach - PF/NOPF

Command Input

PF [*port*]
NOPF [*port*]

Argument

port	Port number.

Description

Port Format (**PF**) allows you 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 -- you must explicitly direct **PF** to proceed.

Any onboard serial port configured via the **PF** command saves the configuration values (baud rate, parity, character width, and number of stop bits) in BBRAM. The configuration remains in effect through power-up or any normal reset.

Note The Reset and Abort option sets BBRAM for Port 1 (LUN 0), to use the ROM defaults for port configuration. (Refer to the *Installation and Start-up* section for details on terminal set-up.)

Caution 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

```
147-Bug>PF
Current port assignments:  (Port #: Board name, Port name)
[00: MVME147- "1"]  [01: MVME147- "2"]  [02: MVME147- "3"]
[03: MVME147- "4"]  [04: MVME147- "PTR"]
Console port = LUN $00
147-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 the *Assigning a New Port* section in this command discussion.

When **PF** 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 uppercase 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 (CTRL 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 prompting in the original order (i.e., baud rate, parity type, ,...).

=	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: Changing the number of stop bits on port number 1.

```
147-Bug>PF 1
Baud rate [110,300,600,1200,2400,4800,9600,19200] = 9600?
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 is not
147-Bug>                              required.
```

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 boards with adjustable base addressing; e.g., the MVME050. Entering no value selects the default base address shown.

Baud rate:

You 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:

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

Number of stop bits:

Only 1 and 2 stop bits are supported.

Synchronization type:

As 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 MVME147, MVME050, etc.). Pressing the **Return** or **Enter** key on the console at this point causes **PF** to list the currently supported modules and ports. When 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, you are allowed to change the port base address. Pressing the **Return**/**Enter** 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* section in this command discussion. 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 you have 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.

```
147-Bug>PF 7
Logical Unit $07 unassigned
Name of board? (CR)           Cause PF to list supported boards,
Boards and ports supported:   ports.
MVME147: 1,2,3,4,PTR
MVME050: 1,2,PTR2
```

```
Name of board? MVME050     Upper- or lowercase accepted
Name of port? PTR2
Port base address = $FFFF1080? (CR)
Auto Line Feed protocol [Y,N] = N? .
```

Interactive mode is not entered because hardware has fixed configuration.

```
OK to proceed (y/n)? Y
147-Bug>
```

NOPF Port Detach

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

Put RTC in Power Save Mode for Storage - PS

Command Input

PS

Description

The **PS** command is used to turn off the oscillator in the RTC chip, MK48T02. The MVME147 module is shipped with the RTC oscillator stopped to minimize current drain from the onchip battery. Normal cold start of the MVME147 with the 147Bug EPROMs installed gives the RTC a “kick start” to begin oscillation. To disable the RTC, you must enter **PS**.

The **SET** command restarts the clock. Refer to the **SET** command for further information.

Example

```
147-Bug>PS                    Clock is in battery save mode
147-Bug>
```

ROMboot Enable/Disable - RB/NORB

Command Input

RB
NORB

Description

The **RB** command enables the search for and booting from a routine nominally encoded in onboard ROMs/PROMs/EPROMs/EEPROMs on the MVME147. However, the routine can be in other memory locations, as detailed in the **RB** command options given below. Refer also to the *ROMboot* section in Chapter 1. The search for and execution of a ROMboot routine is done ONLY in the Bug mode and is excluded from the system mode. If ROMboot and AUTOboot (refer to **AB** command) are enabled, ROMboot is executed first and if there is a return to the Bug, AUTOboot is executed. You also can select whether this occurs only at power-up, or at any board reset. These selections are stored in the BBRAM that is part of the MK48T02 (RTC), and remain in effect through power-up or any normal reset.

Note The Reset and Abort option sets the ROMboot function to the default condition (disabled) until enabled again by the RB command.

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

Example 1: Enable ROMboot function.

```
147-Bug> RB
Boot at Power-up only or any board Reset [P,R] = P?(CR)  Note 1
Enable search of VMEbus [Y,N] = N? (CR)                 Note 2
Boot direct address = $FF800000? (CR)                  Note 3
ROM boot enabled
147-Bug>
```

Example 2: Disable ROMboot function.

```
147-Bug> NORB
ROM boot disabled                                    Note 4
147-Bug>
```

Notes

1. If R is entered, then boot is attempted at any board reset.
2. If Y is entered, the search for "`BOOT`", etc. starts at the end of onboard memory, in 8KB increments.
3. This is the first address that is searched for "`BOOT`", etc. and may be set by you to point to the ROMboot routine, so the search is faster. The default address is the start of the 147Bug EPROMs.
4. This disables the ROMboot function, but does not change any options chosen under RB.

Register Display - RD

Command Input

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

Arguments

+	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.
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
	FPC Floating-point coprocessor
	MMU Memory management unit
reg1	is the first register in a range of registers.
reg2	is the last register in a range of registers.

Description

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 the **RD** command is executed. At reset time, this mask is set to display the MPU registers. This register mask can be changed with the **RD** command. The optional arguments allow you to enable or disable the display of any register or group of registers. This is useful for showing only the registers of interest, minimizing unnecessary data on the screen; and also in saving screen space, which is reduced particularly when coprocessor registers are displayed.

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

1. The qualifier is applied to the next register range only.
2. If no qualifier is specified, a + qualifier is assumed.
3. All device names should appear before any register names.
4. 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.
5. When specifying a register range, *reg1* and *reg2* do not have to be of the same class.
6. 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 of Registers	**Type of Registers**	**Mnemonics**
10	System Registers	PC, SR, USP, MSP, SP, 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 FPC registers in ordering sequence are:

Number of Registers	**Type of Registers**	**Mnemonics**
3	System Registers	FPCR, FPSR, FPIAR
8	Data Registers	FP0-FP7

The MMU registers in ordering sequence are:

Number of Registers	Type of Registers	Mnemonics
5	Address Translation/Control	CRP, SRP, TC, TT0, TT1
1	Status	MMUSR

Example 1

```
147-Bug>RD
PC  =00004000 SR  =2700=TR:OFF_S._7_..... VBR =00000000
USP =00005830 MSP =00005C18 ISP*=00006000 SFC =0=F0
CACR=0=D:.... I:...         CAAR=00000000 DFC =0=F0
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
147-Bug>
```

Notes

1. An asterisk (*) following a stack pointer name indicates that it is the active stack pointer.
2. The status register includes a mnemonic portion to help in reading it:

Trace Bits:	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 from 0 to 7 indicates the 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 Cache Control Register (CACR) 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: Display only the MMU registers.

```
147-Bug>RD =MMU
CRP  =00000001_00000000       SRP  =00000001_00000000
TC   =00000000 TT0  =00000000 TT1  =00000000
MMUSR=0000=......._0
00004000 4AFC               ILLEGAL
147-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: Display only the FPC registers.

```
147-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
147-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: Remove D3 through D5 and A2, and add FPSR and FP0, starting with the previous display.

```
147-Bug>RD MPU/-FPC/-D3-D5/-A2/FP0/FPSR
PC  =00004000 SR  =2700=TR:OFF_S._7_..... VBR =00000000
USP =00005830 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
147-Bug>
```

Example 5: Set the display to D6 and A3 only.

```
147-Bug>RD =D6/A3
D6   =00000000 A3   =00000000
00013000 4AFC               ILLEGAL
147-Bug>
```

Note that the above sequence sets the display to D6 only and then adds register A3 to the display.

Example 6: Restore all the MPU registers.

```
147-Bug>RD +MPU
PC  =00004000 SR  =2700=TR:OFF_S._7_..... VBR =00000000
USP =00005830 MSP =00005C18 ISP*=00006000 SFC =0=F0
CACR=0=D:.... I:...         CAAR=00000000 DFC =0=F0
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
147-Bug>
```

Note that an equivalent command would have been **RD PC-A** or **RD = MPU**.

Remote - REMOTE

Command Input

REMOTE

Description

The **REMOTE** command duplicates the remote (modem operation) functions available from the "system" mode **MENU** command, entry number 4. It is accessible from either the "bug" or "system" mode (refer to **MENU** command in Appendix A for details on remote operation).

The modem type, baud rate, and concurrent flag are saved in the BBRAM that is part of the MK48T02 (RTC) and, remain in effect through any normal reset. If the MVME147 and the modem do not share the same power supply then, the selections remain in effect through power-up; otherwise no guarantees are made as to the state of the modem.

Note The Reset and Abort option sets the "dual console" (concurrent) mode to the default condition (disabled), until enabled again by the **REMOTE** command.

Cold/Warm Reset - RESET

Command Input

RESET

Description

The **RESET** command is used to issue a local SCSI bus reset and also allows you to specify the level of reset operation that is in effect when a RESET exception is detected by the processor. A reset exception can be generated by pressing the RESET switch on the MVME147 front panel, or by executing a software reset.

When the **ENV** command is invoked, the interactive mode is entered immediately. While in the interactive mode, the following rules apply:

	Only listed values are accepted when a list is shown. Uppercase or lowercase may be interchangeably used when a list is shown.
^	Backs up to the previous field.
.	Entering a period by itself or following a new value/setting causes **RESET** to exit the interactive mode. Control returns to the Bug.
(**CR**)	Pressing return without entering a value preserves the current value and causes the next prompt to be displayed.:

`Reset local SCSI bus [Y/N}`	Selecting this (**Y**) causes an immediate reset of the local MVME147 SCSI bus via the PCC SCSI port interrupt control register.
`Automatic reset of SCSI buses [Y/N}`	Selecting this (**Y**) causes a SCSI bus reset command to be issued, at reset time, to each available SCSI controller.

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.

Example 1: Do a local SCSI bus reset and exit**.**

```
147-Bug>RESET
Reset Local SCSI Bus [Y,N] N? Y.
147-Bug>
```

Example 2: Arm automatic SCSI bus resets and exit.

```
147-Bug>RESET
Reset Local SCSI Bus [Y,N] N?(CR)
Automatic reset of known SCSI Buses on RESET [Y,N] =N? Y.
```

Example 3: Arm warm resets and execute a software reset.

```
147-Bug> RESET
Reset Local SCSI Bus [Y,N] N? (CR)
Automatic reset of known SCSI Buses on RESET [Y,N] = Y?(CR)
Cold/Warm Reset [C,W] = C? W
Execute Soft Reset [Y,N] N? Y

Copyright Motorola Inc. 1989, 1990  All Rights Reserved
VME147 Monitor/Debugger Release 2.3 - 3/30/90
CPU running at 25 MHz
WARM Start
147-Bug>
```

Register Modify - RM

Command Input

RM *reg*

Arguments

reg	The mnemonic for the particular register, the same as it is displayed.

Description

RM allows you 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).

Example 1

```
147-Bug>RM D5
D5   =12345678? ABCDEF^          Modify register and back up.
D4   =00000000? 3000.            Modify register and exit.
147-Bug>
```

Example 2

```
147-Bug>RM SFC
SFC  =7=CS    ? 1=               Modify register and reopen.
SFC  =1=UD    ? .                Exit.
147-Bug>
```

The **RM** command is also used to modify the memory management unit registers.

Example 3

```
147-Bug>RM CRP
CRP  =00000001_00000000        ?(CR)
SRP  =00000001_00000000        ?(CR)
TC   =00000000 ?87654321
TT0  =00000000 ?12345678
TT1  =00000000 ?87654321
MMUSR=0000=. ....._0? .
```

```
147-Bug>RD =MMU
CRP  =00000001_00000000      SRP =00000001_00000000
TC   =87654321 TT0 =12345678 TT1 =87654321
MMUSR=0000=......._0
00004000 4AFC             ILLEGAL
147-Bug>
```

The **RM** command is also used to modify the floating-point coprocessor registers (MC68882).

Example 4

```
147-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.
147-Bug>
```

```
147-Bug>RD =FPC
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
147-Bug>
```

Register Set - RS

Command Input

RS *reg* [*hexadecimal number*]. . .

Arguments

reg The mnemonic for the particular register.

Description

The **RS** command allows you to change the data in the specified target register. It works in essentially the same way as the **RM** command.

Example 1

```
147-Bug>RS D5 12345678                Change MPU register.
D5   =12345678
147-Bug>
```

Example 2

```
147-Bug>RS TT0 87654321               Change MMU register.
TT0  =87654321
147-Bug>
```

Example 3

```
147-Bug>RS FP0 0_1234_5               Change FPC register.
FP0  =0_1234_5000000000000000= 6.6258385370745493_E-3530
147-Bug>
```

Switch Directories - SD

Command Input

SD

Description

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 you are in at the particular time) may be listed using the **HE** (Help) 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

```
147-Bug>SD
147-Diag>
```

You have changed from the debugger directory to the diagnostic directory, as can be seen by the `147-Diag>` prompt.

Example 2

```
147-Diag>SD
147-Bug>
```

You are now back in the debugger directory.

Set Time and Date - SET

Command Input

SET

Description

The **SET** command is interactive and begins with you entering **SET** followed by a carriage return. At this time, a prompt asking for *MM/DD/YY* is displayed. You 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, you 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,) for details.

Next, a prompt is presented asking for *HH:MM:SS*. You 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: Set a date and time of May 16, 1990 2:05:32 PM.

```
147-Bug>SET
Weekday  xx/xx/xx    xx:xx:xx
Present calibration = -0
Enter date as MM/DD/YY.
05/11/90
Enter Calibration value +/- (0 to 31)
(CR)
Enter time as HH:MM:SS (24 hour clock)
14:05:32
147-Bug>
```

Trace - TRACE

Command Input

T [*count*]

Description

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 147Bug.

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 147Bug.

The trace functions are implemented with the trace bits (T0, T1) in the MC68030 status register, therefore, these bits should not be modified while using the trace commands.

Example: Assume that the following program resides at location $10000.

```
147-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
147-Bug>
```

Initialize PC and D0:

```
147-Bug>RS PC 10000
PC   =00010000
147-Bug>
```

```
147-Bug>RS D0 8F41C
D0   =0008F41C
147-Bug>
```

Display target registers and trace one instruction:

```
147-Bug>RD
PC   =00010000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:.... I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010000 2200               MOVE.L      D0,D1
147-Bug>T
PC   =00010002 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:.... I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010002 4282               CLR.L       D2
147-Bug>
```

Trace next instruction:

```
147-Bug>(CR)
PC   =00010004 SR   =2704=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:.... I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010004 4D01               ADD.B       D1,D2
147-Bug>
```

Trace the next two instructions:

```
147-Bug>T2
PC   =00010006 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:.... I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010006 E289              LSR.L      #$1,D1
PC   =00010008 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:.... I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010008 66FA              BNE.B      $10004
147-Bug>
```

Terminal Attach - TA

Command Input

TA [*port*]

Description

TA command allows you to assign any serial port to be the console. The port specified must already be assigned (refer to the Port Format (**PF**) command). Any onboard serial port selected as console is saved in the BBRAM that is part of the MK48T02 RTC, and remains in effect through power-up or any normal reset.

Note The reset and abort option returns the console port to the default port (port 1, LUN 0).

Example 1: Selecting port 3 (logical unit #02) as console.

```
147-Bug>TA 2  (See note below)
Changing the Console Port from [0: VME147- "1"] to [2: VME147 "3"]
```

Example 2: Restoring console to default port (port 1, LUN 0).

```
147-Bug>TA
Changing the Console Port from [2: VME147- "3"] to [0: VME147- "1"]
```

Note Console changed to port 3 and no prompt appears, unless port 3 was already the console. All keyboard exchanges and displays are now made through port 3. This remains in effect (through power-up or reset) until either another TA command has been issued or the reset and abort option has been invoked.

Trace on Change of Control Flow - TC

Command Input

TC [*count*]

Description

The **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 147Bug.

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 147Bug.

The trace functions are implemented with the trace bits (T0, T1) in the MC68030 status register, therefore, these bits should not be modified while using the trace commands.

Example: Assume that the following program resides at location $10000.

```
147-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
147-Bug>
```

Initialize PC and D0:

```
147-Bug>RS PC 10000
PC   =00010000
147-Bug>

147-Bug>RS D0 8F41C
D0   =0008F41C
147-Bug>
```

Trace on change of flow:

```
147-Bug>TC
00010008 66FA               BNE.B      $10004
PC   =00010004 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:.... I:...           CAAR =00000000 DFC  =0=F0
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   =00006000
00010004 4D01               ADD.B   D1,D2
147-Bug>
```

Note that the above display also shows the change of flow instruction.

Display Time and Date - TIME

Command Input

TIME

Description

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 16, 1990 2:05:32 would be displayed as:

```
147-Bug>TIME
Wednesday  5/16/90    14:05:32
147-Bug>
```

Transparent Mode - TM

Command Input

TM [*port*] [*escape*]

Description

TM essentially connects the console serial port and the host port together, allowing you 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 "*port*" allows you 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 console 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 **PF** command.

The optional escape argument allows you 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

```
147-Bug>TM                                 Enter TM.
Escape character:  $01=^A                  Exit code is always displayed.
^A                                         Exit transparent mode.
147-Bug>
```

Example 2

```
147-Bug>TM ^ g                             Enter TM and set escape
Escape character:  $07=^G                  character to ^G.
^G                                         Exit transparent mode.
147-Bug>
```

Trace to Temporary Breakpoint - TT

Command Input

TT*addr*

Description

TT 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 147-Bug. 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 147Bug.

The trace functions are implemented with the trace bits (T0, T1) in the MC68030 status register, therefore, these bits should not be modified while using the trace commands.

Example: Assume that the following program resides at location $10000.

```
147-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
147-Bug>
```

Initialize PC and D0:

```
147-Bug>RS PC 10000
PC   =00010000
147-Bug>
```

147-Bug>**RS D0 8F41C**

```
D0   =0008F41C
147-Bug>
```

Trace to temporary breakpoint:

147-Bug>**TT 10006**

```
PC   =00010002 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:.... I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010002 4282              CLR.L      D2
PC   =00010004 SR   =2704=TR:OFF_S._7_..Z..  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:.... I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010004 D401              ADD.B      D1,D2
At Breakpoint
PC   =00010006 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:.... I:...          CAAR =00000000 DFC  =0=F0
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   =00006000
00010006 E289              LSR.L      #$1,D1
147-Bug>
```

Verify S-Records Against Memory - VE

Command Input

VE [*port*] [*addr*] [**;x** | **-c**] [=*text*]

Options

-c	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, the comparison is not made.
x	Echo. Echoes the S-records to your terminal as they are read in at the host port.

Description

This 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.

The **VE** 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 MVME147 memory. If the data does not compare, then you are alerted via information sent to the terminal screen.

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

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

The optional *addr* field allows you 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.

(For information on S-records, refer to Appendix C.) 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 equals sign (=), is sent to the host before 147Bug begins to look for S-records at the host port. This allows you 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 your 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 a line feed (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, in situations where the host system does not echo characters, it is required 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.

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, the command is aborted.

If a non-hexadecimal character is encountered within the data field of a data record, the part of the record which had been received up to that time is printed to the screen and the 147Bug 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 147Bug AND if the checksum comparison has not been disabled via the "**-c**" option, 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 this program was compiled and converted into an S-Record file named TEST.MX as follows:

```
S00F00005445535453335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
```

This file was downloaded into memory at address $40000 (refer to the **LO** command for more information). The program may be examined in memory using the **MD** command.

```
147-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
147-Bug>
```

Suppose you want to make sure that the program has not been destroyed in memory. The **VE** command is used to perform a verification.

```
147-Bug>VE -65000000 ;x=cat TEST.MX
S00F000054455354533335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
Verify passes.
147-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.

```
147-Bug>M 40002
00040002 D088 ? D089 .
147-Bug>VE -65000000 ;x=cat TEST.MX
S00F000054455354533335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091

The following record(s) did not verify .....

S30D65040000------88--------B3
147-Bug>
```

The byte that was changed in memory does not compare with the corresponding byte in the S-record.

Using the One-Line Assembler/Disassembler 4

Introduction

Included as part of the 147Bug 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 147Bug 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

The part of the assembly language that provides the mnemonic machine-instruction operation codes for the MC68030/MC68882 machine instructions is described in the *MC68030UM 32-Bit Microprocessor User's Manual* and *MC68881UM Floating-Point Coprocessor User's Manual*. 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 147Bug assembler recognizes only two directives called DC.W (define constant) and SYSCALL. These directives are used to define data within the program, and to make calls to 147Bug utilities. Refer to the *DC.W - Define Constant Directive* and *SYSCALL - System Call Directive* sections in this chapter.

Comparison with MC68030 Resident Structured Assembler

There are several major differences between the 147Bug 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 147Bug assembler processes each line of a program as an individual unit. Due mainly to this basic functional difference, the capabilities of the 147Bug assembler are more restricted:

1. 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.

2. 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.

3. Only two directives (DC.W and SYSCALL) are accepted.
4. No macro operation capability is included.
5. No conditional assembly is used.
6. Several symbols recognized by the resident assembler are not included in the 147Bug assembler character set. These symbols include > and <.
7. Three symbols, the ampersand (&), the slash (/), and the asterisk (*), have multiple meanings to the resident assembler, depending on the context:

&	Ampersand	AND ***or*** decimal number prefix.
/	Slash	Divide ***or*** delimiter in a register list.
*	Asterisk	Multiply ***or*** current PC.

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 147Bug 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:

❑ Operation codes	Correspond to the MC68030/MC68882 instruction set.
❑ Define constant directive	DC.W is recognized to define a constant in a word location.
❑ System call directive	SYSCALL is used to call 147Bug 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 is 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 a **b**, **w**, or **l**, which represents the size:

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 the 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 <EA> field. Thus, the contents on 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 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 from an address register, it is treated as a signed hexadecimal offset. Otherwise, it is treated as a straight unsigned hexadecimal.

Example

```
MOVE.L       #1234,5678
MOVE.L       FFFFFFFC(A0),5678
```

This disassembles to:

```
00003000  21FC0000 12345678    MOVE.L     #$1234,($5678).W
00003008  21E8FFFC 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. For example,

1. BRA is returned for BT.
2. 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 accepts 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.

Binary is a string of binary digits (0 and 1) preceded by a percent sign (%).

Octal is a string of octal digits (0 through 7) preceded by a "commercial at" sign (@).

Decimal is a string of decimal digits (0 through 9) preceded by an ampersand (&).

Hexadecimal is a string of hexadecimal digits (0 through 9, A through F), optionally preceded by a dollar sign ($).

Examples

Binary	`%1000110`
Octal	`@456`
Decimal	`&12334` `-&987654321`
Hexadecimal	`$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.

```
00005000    21FC0000 12345668   MOVE.L   #$1234,($5678).W
00005008    0053                DC.W     'S'
0000500A    223C41424344        MOVE.L   #'ABCD',D1
00005010    3536                DC.W     '56'
```

The following register mnemonics are recognized/referenced by the assembler/ disassembler:

Pseudo-Registers

R0-R7	User Offset Registers

Main Processor Registers

PC	Program Counter; used only in forcing program counter-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; address register A7 represents the active system stack pointer, that is, one of USP, MSP, or ISP, as specified by the M and S bits of the status register (SR).

Floating-Point Coprocessor Registers

FPCR	Control Register
FPSR	Status Register
FPIAR	Instruction Address Register
FP0-FP7	Floating-Point Data Registers

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

Character Set

The character set recognized by the 147Bug assembler is a subset of ASCII, and is listed below:

- The letters A through Z (uppercase and lowercase)
- The integers 0 through 9
- Arithmetic operators: + - * / << >> ! &
- Parentheses ( )
- Characters used as special prefixes:

#	Pound sign	The intermediate form of addressing
$	Dollar sign	Hexadecimal number
&	Ampersand	Decimal number
@	Commercial at sign	Octal number
%	Percent sign	Binary number
'	Apostrophe	ASCII literal character string

- Five separating characters:

	Space
,	Comma
.	Period
/	Slash
-	Dash

- The asterisk character (*) indicates the current location.

Addressing Modes

Effective address 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 the *Data Organization and Addressing Capabilities* section of the *MC68030 32-Bit Microprocessor User's Manual*.

The following table summarizes the addressing modes of the MC68030 which are accepted by the 147Bug one-line assembler.

Table 4-1. 147Bug Assembler Addressing Modes

Format	Description
D*n*	Data register direct
A*n*	Address register direct
(A*n*)	Address register indirect
(A*n*)+	Address register indirect with post-increment
-(A*n*)	Address register indirect with pre-decrement
d(A*n*)	Address register indirect with displacement
d(A*n*,X*i*)	Address register indirect with index, 8-bit displacement
(*bd*,A*n*,X*i*)	Address register indirect with index, base displacement.
([*bd*,A*n*],X*i*,*od*)	Address register memory indirect post-indexed
([*bd*,A*n*,X*i*],*od*)	Address register memory indirect pre-indexed
*d*16(PC)	Program counter indirect with displacement
*d*8(PC,Xi)	Program counter indirect with index, 8-bit displacement
(*bd*,PC,X*i*)	Program counter indirect with index, base displacement
([*bd*,PC],X*i*,*od)*	Program counter memory indirect post-indexed
([*bd*,PC,X*i*],*od*)	Program counter memory indirect pre-indexed
(*xxxx*).W	Absolute word address
(*xxxx*).L	Absolute long address
#*xxxx*	Immediate data

You may use an expression in any numeric field of these addressing modes. The assembler has a built-in expression evaluator that supports the following operand types and operators:

Type	Example
Binary numbers	`%10`
Octal numbers	`@765..0`
Decimal numbers	`&987..0`
Hexadecimal numbers	`$FED..0`
String literals	`'CHAR'`
Offset registers	`R0 - R7`
Program counter	`*`

Allowed operators are:

+	Plus	Add
-	Minus	Subtract
*	Asterisk	Multiply
/	Slash	Divide
<<	Left angle brackets	Shift left
>>	Right angle brackets	Shift right
!	Exclamation mark	Bitwise OR
&	Ampersand	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 you force the order of precedence through the use of parentheses.

Possible points of confusion:

1. Keep in mind that where a number is intended and it could be confused with a register, it must be differentiated in some way.

```
CLR     D0          Means CLR.W register D0.
CLR     $D0         On the other hand,
CLR     0D0         all mean CLR.W
CLR     +D0         memory location $D0.
CLR     D0+0
```

2. 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.

   Example

`***`	Means PC * PC
`*+*`	Means PC + PC
`2**`	Means 2 * PC
`*&&16`	Means PC AND &16

3. When specifying operands, you may skip or omit entries with the following addressing modes.

   a. Address register indirect with index, base displacement.

   b. Address register memory indirect post-indexed.

   c. Address register memory indirect pre-indexed.

   d. Program counter indirect with index, base displacement.

   e. Program counter memory indirect post-indexed.

   f. Program counter memory indirect pre-indexed.

4. For modes address register/program counter indirect with index, base displacement, the rules for omission/skipping are as follows:

   a. You may terminate the operand at any time by specifying ")".

   Example

   ```
   CLR      ( )
   ```
   or

```
CLR     (,,)                is equivalent to
CLR     (0.N,ZA0,ZD0.W*1)
```

b. You 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 you do not specify the base register, the default “ZA0” is forced.

d. If you do not specify the index register, the default “ZD0.W*1” is forced.

e. Any unspecified displacements are defaulted to “0.N”.

5. 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 post-indexed 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 and word **.w** 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

`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 you in making the appropriate TRAP #15 entry to 147Bug functions as defined in Chapter 5. The format for this directive is:

SYSCALL *function name*

Example

The following two pieces of code produce identical results.

```
TRAP     #$F
DC.W     0
```

or

```
SYSCALL .INCHR
```

Entering and Modifying Source Programs

Your 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 Block Move (**BM**) command).

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 is returned; if not (random data occurs), the DC.W $*xxxx* (always hexadecimal) is returned. Because the disassembler gives precedence to instructions, a word of data that corresponds to a valid instruction is returned as the instruction.

Entering a Source Line

A new source line may be entered immediately following the disassembled line, using the format discussed in the *Source Line Format* section in this chapter.

```
147-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.

```
147Bug>MM 10000;DI
00010000  528B         ADDQ.L  #1,A3
00010002  4282         CLR.L   D2 ?(CR)
```

If a hardcopy terminal is being used, the above example would look as follows:

```
147Bug>MM 10000;DI
00010000  2600         MOVE.L  D0,D3 ? ADDQ.L #1,A3
00010000  528B         ADDQ.L  #1,A3
00010002  4282         CLR.L   D2 ? (CR)
```

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.

```
147Bug>MM 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 ? (CR)
```

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. You must append the appropriate size extension to the branch instruction.

To reference a current location in an operand expression, the asterisk character (*) can be used.

Examples

```
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. You 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** or **.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 hardcopy listing of a program, use the Printer Attach (**PA**) command to activate the printer port. An **MD** command 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* section in this chapter, the disassembler displays in signed hexadecimal any number it interprets as an offset from an address register; all other numbers are displayed in unsigned hexadecimal.

Index

I N D E X

INDEX

INDEX