

Field Support Manual

MVME 141BUG Debugging Package  
P90X0

Philips Telecommunication and Data Systems



**PHILIPS**



**Field Support Manual**

**MVME 141BUG Debugging Package  
P90X0**

**Philips Telecommunication and Data Systems**

---



**PHILIPS**

A Publication of  
**PHILIPS TELECOMMUNICATIE EN DATA SYSTEMEN  
NEDERLAND B.V.**  
Customer Service Documentation and Training  
Apeldoorn, The Netherlands

**Pub. No.** 5122 991 37341

**Date** August 1988

Great care has been taken to ensure that the information contained in this handbook is accurate and complete. Should any errors or omissions be discovered or should any user wish to make a suggestion for improving this handbook, he is invited to send the relevant details to:

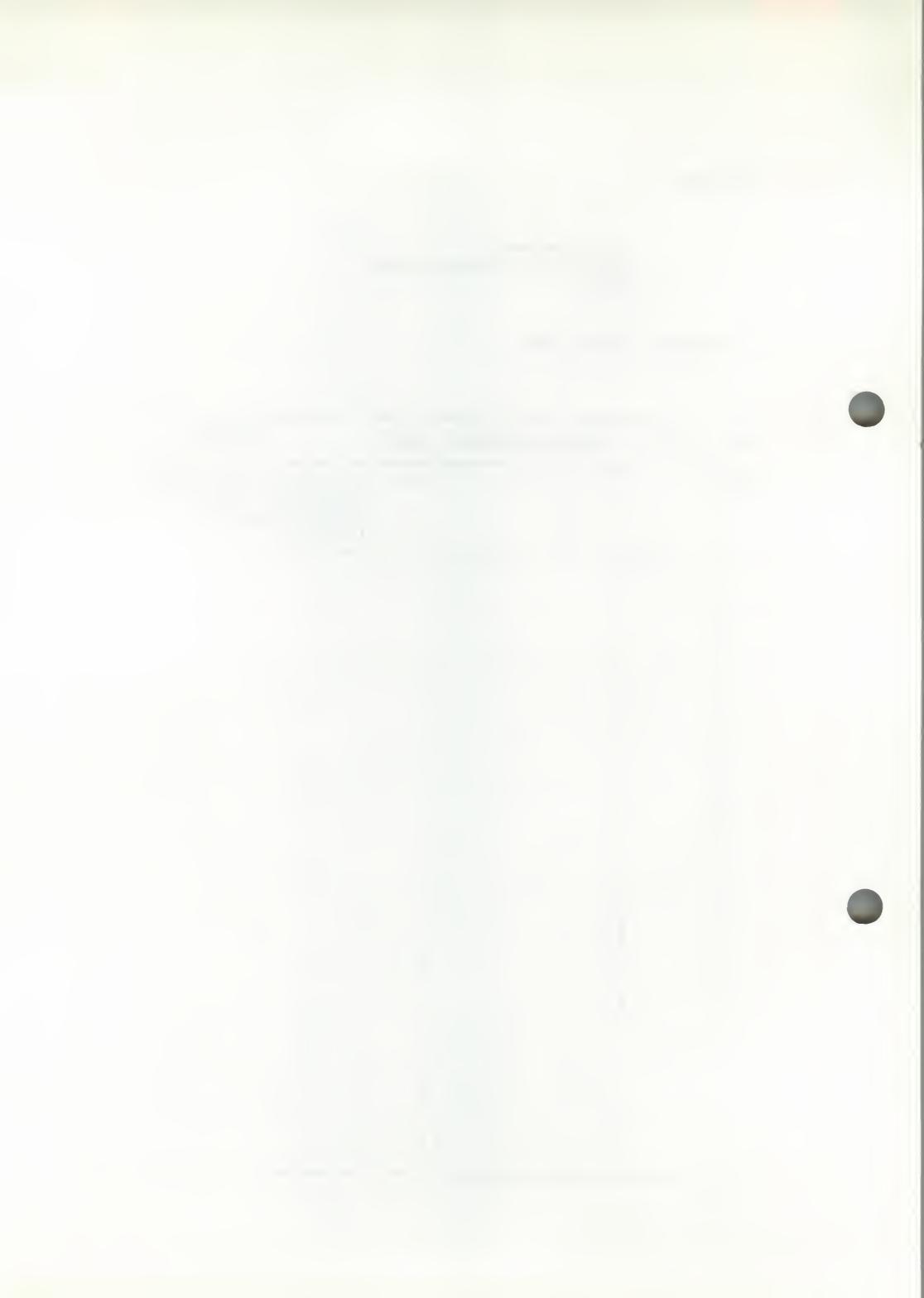
**PHILIPS TELECOMMUNICATIE EN DATA SYSTEMEN**  
Customer Service Documentation and Training  
P.O. BOX 245  
7300 AE APELDOORN  
THE NETHERLANDS

**STATUS RECORD**

TITLE : Field Support Manual  
MVME 141BUG Debugging Package  
P90X0

PUBLICATION NUMBER : 5122 991 3734X

X	UPD.	SI No.	PAGES AFFECTED	DATE	REMARKS
1	New		All	8808	This manual covers the Motorola manual MVME 141BUG Debugging Package/D2 ( d.d. 8807 )



MVME141BUG/D2

JULY 1988

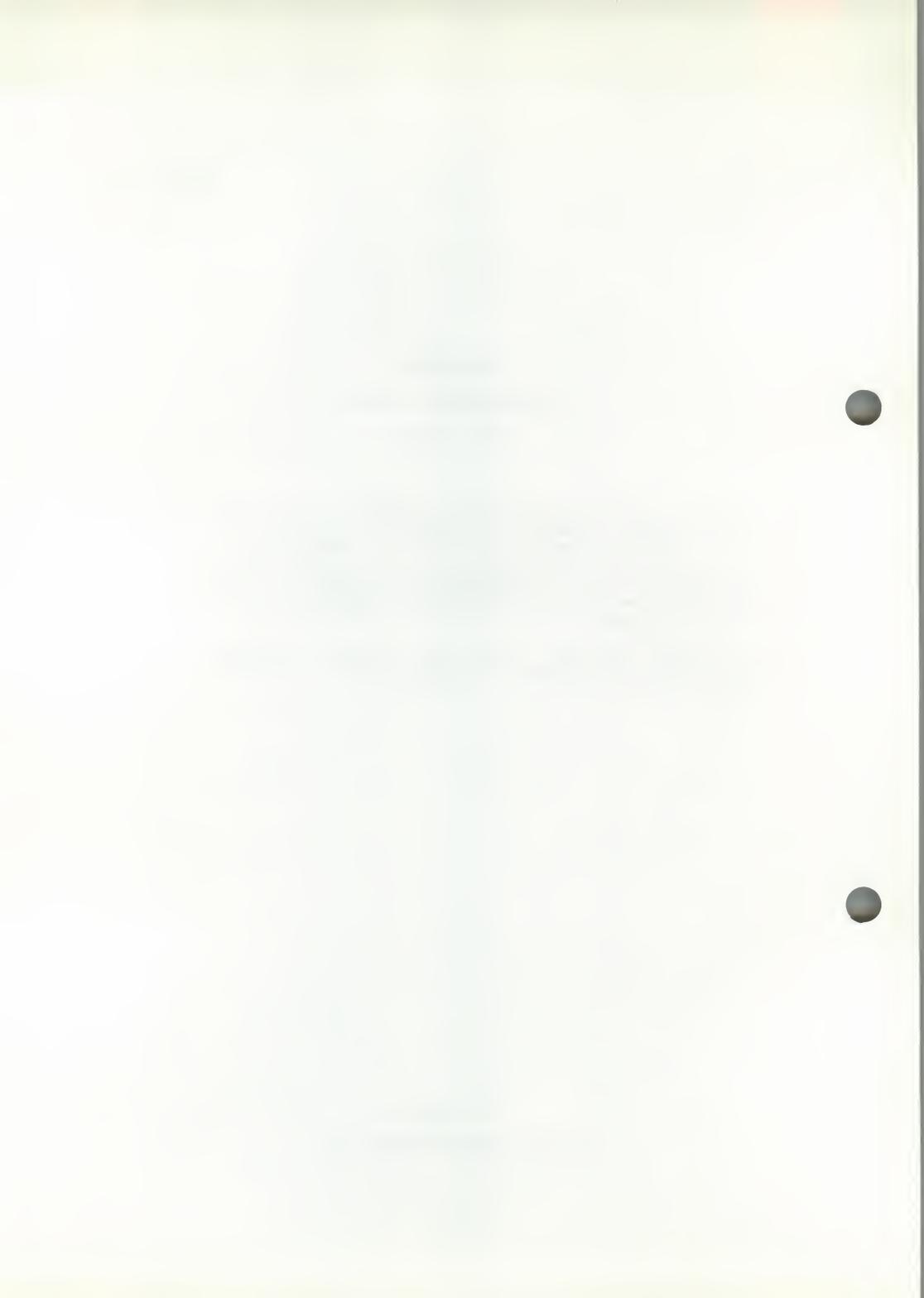
**MVME141BUG**  
**141Bug DEBUGGING PACKAGE**  
**USER'S MANUAL**

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

EXORmacs, HDS-300, HDS-400, SYSTEM V/68, VERSAdos, VMEmodule, VME Delta Series, and 141Bug are trademarks of Motorola Inc.

First Edition

Copyright 1988 by Motorola Inc.

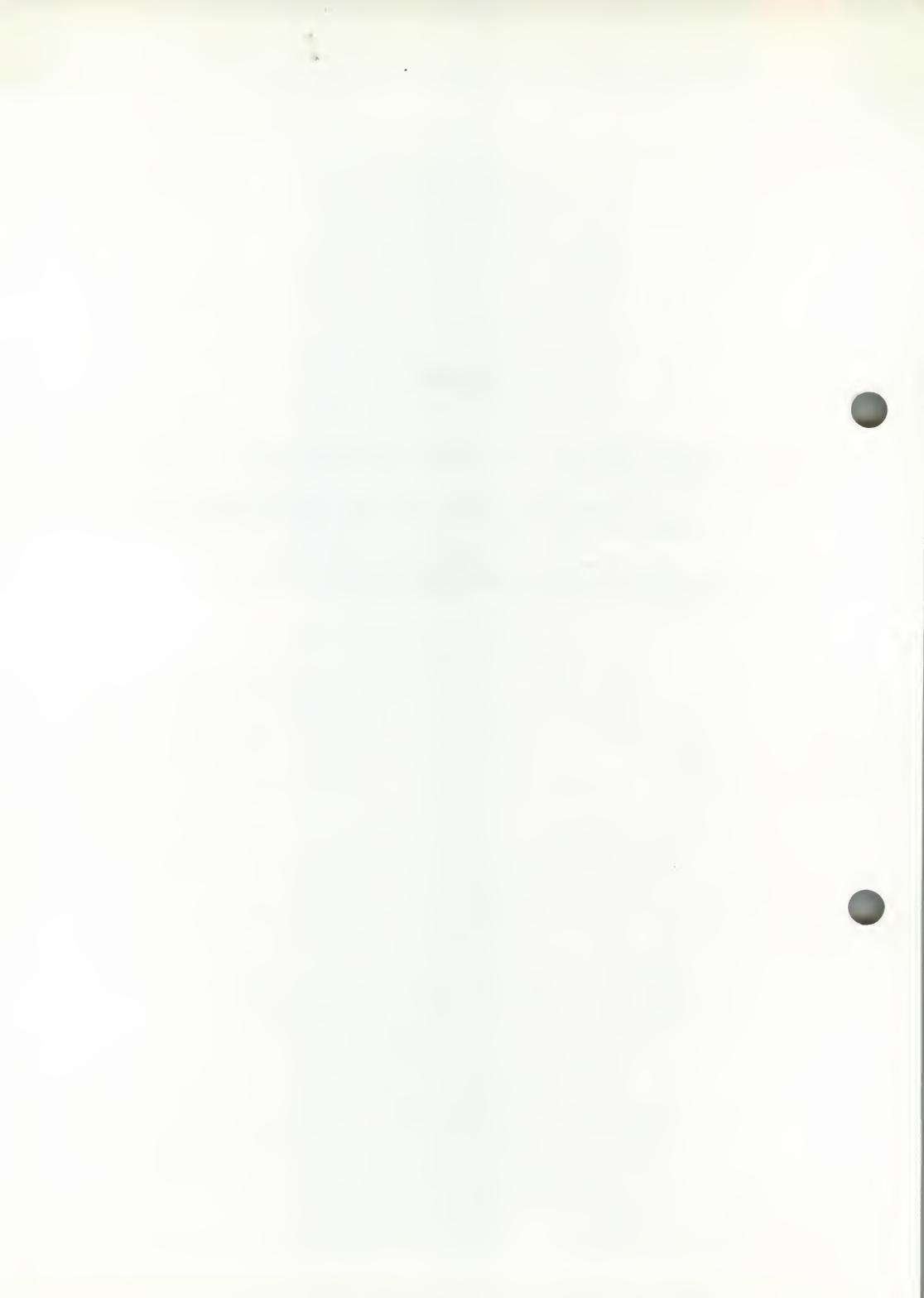


## PREFACE

Unless otherwise specified, all address references are in hexadecimal throughout this manual.

An asterisk (\*) following the signal name for signals which are level significant denotes that the signal is true or valid when the signal is low.

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.



## TABLE OF CONTENTS

	<u>Page</u>
CHAPTER 1 - GENERAL INFORMATION	
1.1 DESCRIPTION OF MVME141Bug .....	1-1
1.2 HOW TO USE THIS MANUAL .....	1-4
1.3 INSTALLATION AND STARTUP .....	1-4
1.4 AUTOBOOT .....	1-6
1.5 ROMboot .....	1-7
1.6 RESTARTING THE SYSTEM .....	1-10
1.6.1 Reset .....	1-10
1.6.2 Abort .....	1-11
1.6.3 Reset and Abort - Restore Battery Backed Up RAM .....	1-11
1.6.4 Break .....	1-12
1.7 MEMORY REQUIREMENTS .....	1-12
1.8 DISK I/O SUPPORT .....	1-14
1.8.1 Blocks Versus Sectors .....	1-14
1.8.2 Disk I/O via 141Bug Commands .....	1-14
1.8.2.1 IOP (Physical I/O to Disk) .....	1-14
1.8.2.2 IOT (I/O Teach) .....	1-15
1.8.2.3 IOC (I/O Control) .....	1-15
1.8.2.4 BO (Bootstrap Operating System) .....	1-15
1.8.2.5 BH (Bootstrap and Halt) .....	1-15
1.8.3 Disk I/O via 141Bug System Calls .....	1-15
1.8.4 Default 141Bug Controller and Device Parameters .....	1-16
1.8.5 Disk I/O Error Codes .....	1-16
1.9 MULTIPROCESSOR SUPPORT .....	1-16
1.9.1 Multi-Processor Control Register (MPCR) Method .....	1-17
1.9.2 VMEchip Method .....	1-18
1.10 DIAGNOSTIC FACILITIES .....	1-19
1.11 REFERENCE MANUALS .....	1-19
CHAPTER 2 - USING THE 141Bug DEBUGGER	
2.1 ENTERING DEBUGGER COMMAND LINES .....	2-1
2.1.1 Syntactic Variables .....	2-3
2.1.1.1 Expression as a Parameter .....	2-3
2.1.1.2 Address as a Parameter .....	2-4
2.1.2 Port Numbers .....	2-7
2.2 ENTERING AND DEBUGGING PROGRAMS .....	2-7
2.3 CALLING SYSTEM UTILITIES FROM USER PROGRAMS .....	2-8
2.4 PRESERVING THE DEBUGGER OPERATING ENVIRONMENT .....	2-8
2.4.1 141Bug Vector Table and Wordspace .....	2-8
2.4.2 Exception Vectors Used by 141Bug .....	2-9
2.4.2.1 Using 141Bug Target Vector Table .....	2-10
2.4.2.2 Creating a New Vector Table .....	2-10
2.4.2.3 141Bug Generalized Exception Handler .....	2-12
2.5 MEMORY MANAGEMENT UNIT SUPPORT .....	2-13
2.6 FUNCTION CODE SUPPORT .....	2-14

## TABLE OF CONTENTS (cont'd)

	<u>Page</u>
 CHAPTER 3 - THE 141Bug DEBUGGER COMMAND SET	
3.1 INTRODUCTION .....	3-1
3.2 AUTOBOOT ENABLE/DISABLE .....	3-3
3.3 BLOCK OF MEMORY COMPARE .....	3-4
3.4 BLOCK OF MEMORY FILL .....	3-6
3.5 BOOTSTRAP OPERATING SYSTEM AND HALT .....	3-8
3.6 BLOCK OF MEMORY INITIALIZE .....	3-9
3.7 BLOCK OF MEMORY MOVE .....	3-10
3.8 BOOTSTRAP OPERATING SYSTEM .....	3-12
3.9 BREAKPOINT INSERT/DELETE .....	3-14
3.10 BLOCK OF MEMORY SEARCH .....	3-15
3.11 BLOCK OF MEMORY VERIFY .....	3-18
3.12 CHECKSUM .....	3-20
3.13 DATA CONVERSION .....	3-23
3.14 DUMP S-RECORDS .....	3-24
3.15 EEPROM PROGRAMMING .....	3-27
3.16 SET ENVIRONMENT TO BUG/OPERATING SYSTEM .....	3-28
3.17 GO DIRECT (IGNORE BREAKPOINTS) .....	3-30
3.18 GO TO NEXT INSTRUCTION .....	3-32
3.19 GO EXECUTE USER PROGRAM .....	3-34
3.20 GO TO TEMPORARY BREAKPOINT .....	3-36
3.21 HELP .....	3-38
3.22 I/O CONTROL FOR DISK .....	3-39
3.23 I/O PHYSICAL (DIRECT DISK ACCESS) .....	3-40
3.24 I/O "TEACH" FOR CONFIGURING DISK CONTROLLER .....	3-43
3.25 LOAD S-RECORDS FROM HOST .....	3-50
3.26 LAN STATION ADDRESS DISPLAY/SET .....	3-53
3.27 MACRO DEFINE/DISPLAY/DELETE .....	3-54
3.28 MACRO EDIT .....	3-57
3.29 ENABLE/DISABLE MACRO EXPANSION LISTING .....	3-59
3.30 SAVE/LOAD MACROS .....	3-60
3.31 MEMORY DISPLAY .....	3-62
3.32 MENU .....	3-64
3.33 MEMORY MODIFY .....	3-65
3.34 MEMORY SET .....	3-68
3.35 SET MEMORY ADDRESS FROM VMEbus .....	3-69
3.36 OFFSET REGISTERS DISPLAY/MODIFY .....	3-70
3.37 PRINTER ATTACH/DETACH .....	3-72
3.38 PORT FORMAT/DETACH .....	3-73
3.38.1 Listing Current Port Assignments .....	3-73
3.38.2 Configuring a Port .....	3-73
3.38.3 Parameters Configurable by Port Format .....	3-74
3.38.4 Assigning a New Port .....	3-75
3.38.5 NOPF Port Detach .....	3-76
3.39 PUT RTC INTO POWER SAVE MODE FOR STORAGE .....	3-77
3.40 ROMBOOT ENABLE/DISABLE .....	3-78
3.41 REGISTER DISPLAY .....	3-79

## TABLE OF CONTENTS (cont'd)

	<u>Page</u>
3.42 REMOTE .....	3-84
3.43 COLD/WARM RESET .....	3-85
3.44 REGISTER MODIFY .....	3-86
3.45 REGISTER SET .....	3-89
3.46 SWITCH DIRECTORIES .....	3-90
3.47 SET TIME AND DATE .....	3-91
3.48 TRACE .....	3-92
3.49 TERMINAL ATTACH .....	3-94
3.50 TRACE ON CHANGE OF CONTROL FLOW .....	3-95
3.51 DISPLAY TIME AND DATE .....	3-96
3.52 TRANSPARENT MODE .....	3-97
3.53 TRACE TO TEMPORARY BREAKPOINT .....	3-98
3.54 VERIFY S-RECORDS AGAINST MEMORY .....	3-100
<b>CHAPTER 4 - USING THE ONE-LINE ASSEMBLER/DISASSEMBLER</b>	
4.1 INTRODUCTION .....	4-1
4.1.1 MC68030 Assembly Language .....	4-1
4.1.1.1 Machine-Instruction Operation Codes .....	4-1
4.1.1.2 Directives .....	4-1
4.1.2 Comparison with MC68030 Resident Structured Assembler .....	4-2
4.2 SOURCE PROGRAM CODING .....	4-2
4.2.1 Source Line Format .....	4-2
4.2.1.1 Operation Field .....	4-3
4.2.1.2 Operand Field .....	4-4
4.2.1.3 Disassembled Source Line .....	4-4
4.2.1.4 Mnemonics and Delimiters .....	4-4
4.2.1.5 Character Set .....	4-6
4.2.2 Addressing Modes .....	4-6
4.2.3 DC.W Define Constant Directive .....	4-9
4.2.4 SYSCALL System Call Directive .....	4-10
4.3 ENTERING AND MODIFYING SOURCE PROGRAMS .....	4-10
4.3.1 Invoking the Assembler/Disassembler .....	4-10
4.3.2 Entering a Source Line .....	4-11
4.3.3 Entering Branch and Jump Addresses .....	4-12
4.3.4 Assembler Output/Program Listings .....	4-12
<b>CHAPTER 5 - SYSTEM CALLS</b>	
5.1 INTRODUCTION .....	5-1
5.1.1 Invoking System Calls Through TRAP #15 .....	5-1
5.1.2 String Formats for I/O .....	5-2
5.2 SYSTEM CALL ROUTINES .....	5-3
5.2.1 .INCHR Function .....	5-4
5.2.2 .INSTAT Function .....	5-5
5.2.3 .INLN Function .....	5-6

## TABLE OF CONTENTS (cont'd)

	<u>Page</u>
5.2.4 .READSTR Function .....	5-7
5.2.5 .READLN Function .....	5-8
5.2.6 .CHKBRK Function .....	5-9
5.2.7 .DSKRD, .DSKWR Functions .....	5-10
5.2.8 .DSKCFIG Function .....	5-13
5.2.9 .DSKFMT Function .....	5-17
5.2.10 .DSKCTRL Function .....	5-19
5.2.11 .OUTCHR Function .....	5-21
5.2.12 .OUTSTR, .OUTLN Functions .....	5-22
5.2.13 .WRITE, .WRITELN Functions .....	5-23
5.2.14 .PCRLF Function .....	5-24
5.2.15 .ERASLN Function .....	5-25
5.2.16 .WRITD, .WRITDLN Functions .....	5-26
5.2.17 .SNDBRK Function .....	5-28
5.2.18 .TM_INI Function .....	5-29
5.2.19 .TM_STRO Function .....	5-30
5.2.20 .TM_RD Function .....	5-31
5.2.21 .DELAY Function .....	5-32
5.2.22 .RTC_TM Function .....	5-33
5.2.23 .RTC_DT Function .....	5-34
5.2.24 .RTC_DSP Function .....	5-35
5.2.25 .RTC_RD Function .....	5-36
5.2.26 .REDIR Function .....	5-37
5.2.27 .REDIR_I, .REDIR_O Functions .....	5-38
5.2.28 .RETURN Function .....	5-39
5.2.29 .BINDEC Function .....	5-40
5.2.30 .CHANGEV Function .....	5-41
5.2.31 .STRCMP Function .....	5-42
5.2.32 .MULU32 Function .....	5-43
5.2.33 .DIVU32 Function .....	5-44
5.2.34 .CHK_SUM Function .....	5-45
5.2.35 .BRD_ID Function .....	5-46

## CHAPTER 6 - 141Bug DIAGNOSTIC FIRMWARE GUIDE

6.1 SCOPE .....	6-1
6.2 OVERVIEW OF DIAGNOSTIC FIRMWARE .....	6-1
6.3 SYSTEM START-UP .....	6-1
6.4 DIAGNOSTIC MONITOR .....	6-3
6.4.1 Monitor Start-Up .....	6-3
6.4.2 Command Entry and Directories .....	6-4
6.4.3 Help - Command HE .....	6-4
6.4.4 Self Test - Prefix/Command ST .....	6-4
6.4.5 Switch Directories - Command SD .....	6-5
6.4.6 Loop-On-Error Mode - Prefix LE .....	6-5
6.4.7 Stop-On-Error Mode - Prefix SE .....	6-5
6.4.8 Loop-Continue Mode - Prefix LC .....	6-5
6.4.9 Non-Verbose Mode - Prefix NV .....	6-6

## TABLE OF CONTENTS (cont'd)

	<u>Page</u>
6.4.10 Display Error Counters - Command DE .....	6-6
6.4.11 Clear (Zero) Error Counters - Command ZE .....	6-6
6.4.12 Display Pass Count - Command DP .....	6-6
6.4.13 Zero Pass Count - Command ZP .....	6-6
6.5 UTILITIES .....	6-6
6.5.1 Write Loop - Command WL.size .....	6-7
6.5.2 Read Loop - Command RL.size .....	6-7
6.5.3 Write/Read Loop - Command WR.size .....	6-7
6.6 MPU TESTS FOR THE MC68030 - Command MPU .....	6-8
6.6.1 MPU A - Register Test .....	6-9
6.6.2 MPU B - Instruction Test .....	6-10
6.6.3 MPU C - Address Mode Test .....	6-11
6.6.4 MPU D - Exception Processing Test .....	6-12
6.7 MC68030 ONCHIP CACHE TESTS - Command CA30 .....	6-13
6.7.1 CA30 A - Basic Data Caching Test .....	6-14
6.7.2 CA30 B - Data Cache Tag RAM Test .....	6-15
6.7.3 CA30 C - Data Cache Data RAM Test .....	6-17
6.7.4 CA30 D - Data Cache Valid Flags Test .....	6-18
6.7.5 CA30 E - Data Cache Burst Fill Test .....	6-19
6.7.6 CA30 F - Basic Instruction Caching Test .....	6-20
6.7.7 CA30 G - Unlike Instruction Function Codes Test .....	6-21
6.7.8 CA30 H - Disable Test .....	6-22
6.7.9 CA30 I - Clear Test .....	6-23
6.8 MEMORY TESTS - Command MT .....	6-24
6.8.1 MT A - Set Function Code .....	6-25
6.8.2 MT B - Set Start Address .....	6-26
6.8.3 MT C - Set Stop Address .....	6-28
6.8.4 MT D - Set Bus Data Width .....	6-30
6.8.5 MT E - March Address Test .....	6-31
6.8.6 MT F - Walk a Bit Test .....	6-32
6.8.7 MT G - Refresh Test .....	6-33
6.8.8 MT H - Random Byte Test .....	6-35
6.8.9 MT I - Program Test .....	6-37
6.8.10 MT J - TAS Test .....	6-39
6.8.11 MT K - Brief Parity Test .....	6-40
6.8.12 MT L - Extended Parity Test .....	6-42
6.8.13 Description of Memory Error Display Format .....	6-43
6.9 MEMORY MANAGEMENT UNIT TESTS - Command MMU .....	6-44
6.9.1 MMU A - RP Register .....	6-45
6.9.2 MMU B - TC Register .....	6-46
6.9.3 MMU C - Super_Prog Space .....	6-47
6.9.4 MMU D - Super_Data Space .....	6-48
6.9.5 MMU E - Write/Mapped-Read Pages .....	6-49
6.9.6 MMU F - Read Mapped ROM .....	6-50
6.9.7 MMU G - Fully Filled ATC .....	6-52
6.9.8 MMU H - User_Data Space .....	6-53
6.9.9 MMU I - User_Prog Space .....	6-54

## TABLE OF CONTENTS (cont'd)

	<u>Page</u>
6.9.10 MMU J - Indirect Page .....	6-55
6.9.11 MMU K - Page-Desc Used-Bit .....	6-56
6.9.12 MMU L - Page-Desc Modify-Bit .....	6-57
6.9.13 MMU M - Segment-Desc Used-Bit .....	6-58
6.9.14 MMU P - Invalid Page .....	6-59
6.9.15 MMU Q - Invalid Segment .....	6-60
6.9.16 MMU R - Write-Protect Page .....	6-61
6.9.17 MMU S - Write-Protect Segment .....	6-62
6.9.18 MMU V - Upper-Limit Violation .....	6-63
6.9.19 MMU W - Lower-Limit Violation .....	6-64
6.9.20 MMU X - Prefetch on Invalid-Page Boundary .....	6-65
6.9.21 MMU Y - Modify-Bit and Index .....	6-67
6.9.22 MMU 0 - Read/Modify/Write Cycle .....	6-68
6.9.23 Table Walk Display Format .....	6-70
6.10 REAL-TIME CLOCK TEST - Command RTC .....	6-71
6.11 BUS ERROR TEST - Command BERR .....	6-73
6.12 FLOATING POINT COPROCESSOR (MC68882) TEST - Command FPC ..	6-74
6.13 MC68681 DUART (SIO) TESTS .....	6-75
6.13.1 Data Formats Test .....	6-76
6.13.2 Baud Rate Test .....	6-77
6.13.3 TX/RX Ready IRQ Test .....	6-78
6.13.4 TX/RX Ready Test .....	6-80
6.13.5 FIFO Full Test .....	6-81
6.13.6 BREAK Test .....	6-82
6.13.7 SIO Timer Test .....	6-83
6.14 VME GATE ARRAY TEST - Command VMEGA .....	6-84
APPENDIX A - MVME141Bug SYSTEM MODE OPERATION .....	A-1
APPENDIX B - DEBUGGING PACKAGE MESSAGES .....	B-1
APPENDIX C - S-RECORD OUTPUT FORMAT .....	C-1
APPENDIX D - INFORMATION USED BY BO AND BH COMMANDS .....	D-1
APPENDIX E - DISK CONTROLLER DATA .....	E-1
APPENDIX F - DISK COMMUNICATION STATUS CODES .....	F-1
INDEX .....	IN-1

## TABLE OF CONTENTS (cont'd)

Page

### LIST OF ILLUSTRATIONS

FIGURE 1-1.	Flow Diagram of 141Bug (Normal) Operational Mode .....	1-2
FIGURE 1-2.	Flow Diagram of 141Bug (System) Operational Mode .....	1-3
FIGURE 6-1.	Sample Table Walk Display .....	6-70

### LIST OF TABLES

TABLE 2-1.	Formats for Debugger Address Parameters .....	2-5
TABLE 2-2.	Exception Vectors Used by 141Bug .....	2-9
TABLE 3-1.	Debugger Commands .....	3-1
TABLE 4-1.	141Bug Assembler Addressing Modes .....	4-7
TABLE 5-1.	141Bug System Call Routines .....	5-3
TABLE 6-1.	MC68030 MPU Diagnostic Tests .....	6-8
TABLE 6-2.	MC68030 Cache Diagnostic Tests .....	6-13
TABLE 6-3.	Memory Diagnostic Tests .....	6-24
TABLE 6-4.	Memory Management Unit Diagnostic Tests .....	6-44
TABLE 6-5.	Sample Table Walk Display .....	6-70



## CHAPTER 1 - GENERAL INFORMATION

## 1.1 DESCRIPTION OF MVME141Bug

The MVME141Bug package is a powerful evaluation and debugging tool for systems built around the MVME141 microcomputer module. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 141Bug includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassembler useful for patching programs, and a self test on power-up feature which verifies the integrity of the system. Various 141Bug routines that handle I/O, data conversion, and string functions are available to user programs through the TRAP #15 handler. In addition, 141Bug provides as an option a "system" mode that allows autoboot on power up or reset, and a menu interface to several system commands used in VME Delta Series systems.

141Bug consists of three parts: a command-driven user-interactive software debugger, described in Chapter 2 and hereafter referred to as the debugger, a command-driven diagnostic package for the MVME141 hardware, described in Chapter 6 and hereafter referred to as the diagnostics, and a user interface which accepts commands from the system console terminal.

When using 141Bug, the user operates out of either the debugger directory or the diagnostic directory. If the user is in the debugger directory, then the debugger prompt "141-Bug>" is displayed and the user has all of the debugger commands at his disposal. If in the diagnostic directory, then the diagnostic prompt "141-Diag>" is displayed and the user has all of the diagnostic commands at his disposal as well as all of the debugger commands. The user may switch between directories by using the Switch Directories (SD) command or may examine the commands in the particular directory that the user is currently in by using the Help (HE) command (refer to Chapter 3).

Because 141Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. The flow of control in normal 141Bug operation is illustrated in Figure 1-1. The flow of control in system 141Bug operation is illustrated in Figure 1-2. When a command is entered, 141Bug executes the command and the prompt reappears. However, if a command is entered which causes execution of user target code (for example, "GO"), then control may or may not return to 141Bug, depending on the outcome of the user program.

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

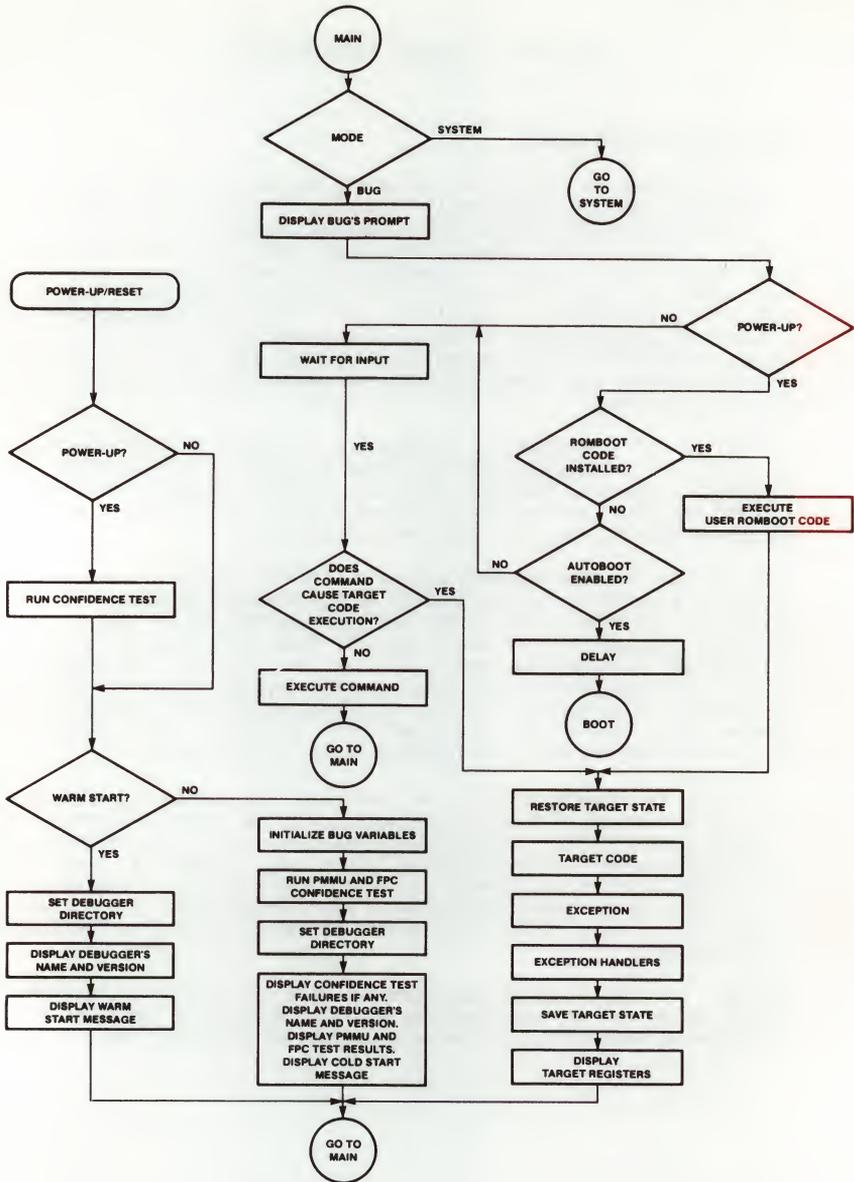


FIGURE 1-1. Flow Diagram of 141Bug (Normal) Operational Mode

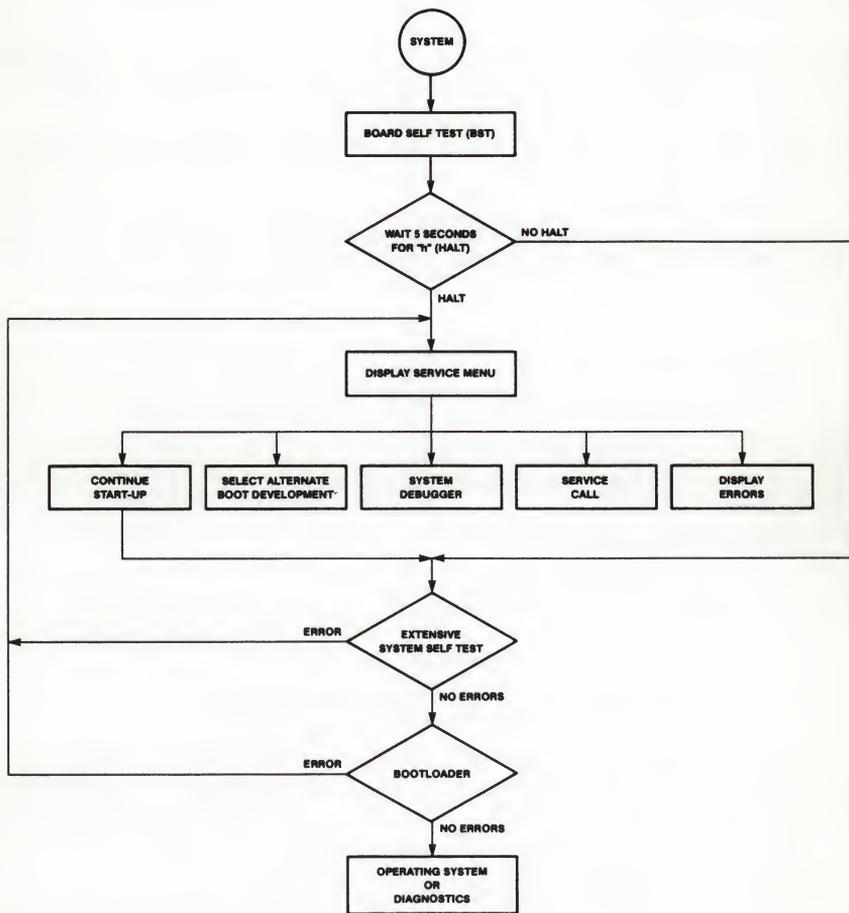


FIGURE 1-2. Flow Diagram of 141Bug (System) Operational Mode

## 1.2 HOW TO USE THIS MANUAL

Users who have never used a debugging package before should read all of Chapter 1 before attempting to use 141Bug. This gives an idea of 141Bug structure and capabilities.

Paragraph 1.3, "Installation and Startup", describes a step-by-step procedure to power up the module and obtain the 141Bug prompt on the terminal screen.

For a question about syntax or operation of a particular 141Bug command, the user may turn to the entry for that particular command in the chapter describing the command set (refer to Chapter 3).

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

### NOTE

In the examples shown, all user input is in **BOLD**. This is done for clarity in understanding the examples (to distinguish between characters input by the user and characters output by 141Bug). The symbol (CR) represents the carriage return key on the terminal keyboard. Whenever this symbol appears, it means a carriage return entered by the user.

## 1.3 INSTALLATION AND STARTUP

Even though the MVME141Bug EPROMs are installed on the MVME141 module, for 141Bug to operate properly with the MVME141, follow this set-up procedure.

### CAUTION

**INSERTING OR REMOVING MODULES WHILE POWER IS APPLIED COULD DAMAGE MODULE COMPONENTS.**

1. Turn all equipment power OFF. Refer to the MVME141 User's Manual and configure the header jumpers on the module as required for the user's particular application. There are no jumper configurations specifically dictated by 141Bug. The Bug works correctly with all jumpers in the as-shipped factory configuration.
2. Refer to the MVME141 User's Manual and configure header J1 for the user's particular application. J1 enables or disables the system controller function of the MVME141, and also sets the group address of the global CSR for the VME controller chip.

3. Be sure that the two 128K x 8 141Bug EPROMs are installed in sockets U56 (odd bytes) and U78 (even bytes) on the MVME141 module.
4. Refer to the set-up procedure for the user's particular chassis or system for details concerning the installation of the MVME141.
5. Connect the terminal which is to be used as the 141Bug system console to connector J5 (port 1) on the MVME141 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 MVME141 ports at power-up.

After power-up, the baud rate of the J5 port (port 1) can be reconfigured by using the Port Format (PF) command of the 141Bug debugger.

#### NOTE

In order for high-baud rate serial communication between 141Bug and the terminal to work, the terminal must do some handshaking. If the terminal being used does not do hardware handshaking via the CTS line, then it must do XON/XOFF handshaking. If the user gets garbled messages and missing characters, then he should check the terminal to make sure XON/XOFF handshaking is enabled.

6. If it is desired to connect device(s) (such as a host computer system or a serial printer) to port 2, connect the appropriate cable to connector J4 and configure the port as detailed in the MVME141 User's Manual. After power-up, this port can be reconfigured by using the PF command of the 141Bug debugger.
7. Power up the system. 141Bug executes self-checks and displays the debugger prompt "141-Bug>".

If after a delay, the 141Bug begins to display test result messages on the bottom line of the screen in rapid succession, the MVME141 is in the Bug "system" mode. If this is not the desired mode of operation, then press the ABORT switch on the front panel of the MVME141. When the MENU is displayed, enter a 3 to go to the system debugger. The environment may be changed by using the set environment (ENV) command. Refer to the Bug operation in the system mode in this manual.

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

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

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

The firmware monitor comes up with the FAIL LED on.

#### 1.4 AUTOBOOT

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

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

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

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

**CAUTION**

THIS INFORMATION APPLIES TO BOTH THE MVME350 AND THE MVME141. ALTHOUGH STREAMING TAPE CAN BE USED TO AUTOBOOT, THE SAME POWER SUPPLY MUST BE CONNECTED TO THE STREAMING TAPE DRIVE, CONTROLLER, AND THE MVME141. AT POWER-UP, THE TAPE CONTROLLER POSITIONS THE STREAMING TAPE TO LOAD POINT WHERE THE VOLUME ID CAN CORRECTLY BE READ AND USED.

IF, HOWEVER, THE MVME141 LOSES POWER BUT THE CONTROLLER DOES NOT, AND THE TAPE HAPPENS NOT TO BE AT LOAD POINT, THE SEQUENCES OF COMMANDS REQUIRED (ATTACH AND REWIND) CANNOT BE GIVEN TO THE CONTROLLER AND AUTOBOOT IS NOT SUCCESSFUL.

## 1.5 ROMboot

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

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

- a. Power must have just been applied (but the RB command can change this to also respond to any reset).
- b. The user's routine must be located within the MVME141 ROM memory map (but the RB command can change this to any other portion of the on-board memory, or even off-board VMEbus memory).
- c. The ASCII string "BOOT" must be located within the specified memory range.
- d. The user's routine must pass a checksum test, which ensures that this routine was really intended to receive control at power-up.

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

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

<u>MODULE OFFSET</u>	<u>LENGTH</u>	<u>CONTENTS</u>	<u>DESCRIPTION</u>
\$00	4 bytes	BOOT	ASCII string indicating possible routine; checksum must be zero, too.
\$04	4 bytes	Entry Address	Longword offset from "BOOT".

\$08	4 bytes	Routine Length	Longword, includes length from "BOOT" to and including checksum.
\$0C	?	Routine name	ASCII string containing routine name.

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

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

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

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

- a. Search direct address for "BOOT".
- b. Search user non-volatile RAM (first 1K bytes of battery back-up RAM).
- c. Search complete ROM map.
- d. Search local RAM (if RB command has selected to operate on any reset), at all 8K byte boundaries starting at \$00004000.
- e. 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:

- a. Outputs a (CR)(LF) sequence to the default output port.
- b. Displays the date and time from the current cursor position.
- c. Outputs two more (CR)(LF) sequences to the default output port.
- d. Returns control to 141Bug.

#### NOTE

This example assumes that the target code is temporarily loaded into the MVME141 RAM. However, an emulator such as the Motorola HDS-300 or HDS-400 could easily be used to load and modify the target code in its actual execution location.

SAMPLE ROMboot ROUTINE - Module preparation including calculation of checksum

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

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

141-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 .....
    
```

141-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
    
```

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

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

141-Bug> M 602C;B

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

0000602C 00 ?F9  
0000602D 00 ?9F.

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

Verify that the checksum is correct.

141-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 .....
141-Bug>

```

The EPROMs can now be programmed and inserted.

COLD Start

141-Bug> Searching for ROM Boot

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

## 1.6 RESTARTING THE SYSTEM

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

### 1.6.1 Reset

Pressing and releasing the MVME141 front panel RESET switch initiates a system reset. COLD and WARM reset modes are available. By default, 141Bug is in COLD mode (refer to the RESET command description). During COLD reset, a total system initialization takes place, as if the MVME141 had just been powered up. The breakpoint table and offset registers are cleared. The target registers are invalidated. Input and output character queues are cleared. Onboard devices (timer, serial ports, etc.) are reset. All static variables (including disk device and controller parameters) are restored to their default states. Serial ports are reconfigured to their default state.

During WARM reset, the 141Bug variables and tables are preserved, as well as the target state registers and breakpoints. If the particular MVME141 is the system controller, then a system reset is issued to the VMEbus and other modules in the system are reset as well.

Reset must be used if the processor ever halts (as evidenced by the MVME141 illuminated STATUS LED), for example after a double bus fault; or if the 141Bug environment is ever lost (vector table is destroyed, etc.).

### 1.6.2 Abort

Abort is invoked by pressing and releasing the ABORT switch on the MVME141 front panel. Whenever abort is invoked when executing a user program (running target code), a "snapshot" of the processor state is captured and stored in the target registers. (When working in the debugger, abort captures and stores only the program counter, status register, and format/vector information.) For this reason, abort is most appropriate when terminating a user program that is being debugged. Abort should be used to regain control if the program gets caught in a loop, etc. The target PC, stack pointers, etc., help to pinpoint the malfunction.

Abort generates a level seven interrupt (non-maskable). The target registers, reflecting the machine state at the time the ABORT switch was pushed, are displayed to the screen. Any breakpoints installed in the user code are removed and the breakpoint table remains intact. Control is returned to the debugger.

### 1.6.3 Reset and Abort - Restore Battery Backed Up RAM

Pressing both the RESET and ABORT switches at the same time and releasing the RESET pushbutton 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:

- . AUTOBOOT (Bug "normal") is turned off.
- . ROMboot (Bug "normal") is turned off.
- . Environment set for Bug "normal" mode.
- . Operating system set for SYSTEM V/68.

#### 1.6.4 Break

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

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

#### 1.7 MEMORY REQUIREMENTS

The program portion of 141Bug is approximately 256Kb of code. These EPROM sockets on the MVME141 are mapped at locations \$FFF00000 through \$FFF1FFFF. However, 141Bug code is position-independent and executes anywhere in memory.

141Bug requires a minimum of 16Kb of read/write memory to operate. This memory is usually the MVME141 onboard read/write memory, requiring stand-alone operation of the MVME141.

The first 16Kb is used for 141Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME141 is reset, the target PC is initialized to the address corresponding to the beginning of the user space and the target stack pointers are initialized to addresses within the user space, with the target ISP set to the top of the user space.

The following abbreviated memory map for the MVME141 highlights addresses that might be of particular interest to the firmware monitor user. Note that addresses are assumed to be hexadecimal throughout this manual. In text, numbers may be preceded with a dollar sign (\$) for identification as hexadecimal.

DRAM LOCATIONS	FUNCTION
00000000-000003FF	Target vector area
00000400-000007FF	Bug vector area
00000800-00000803	Multi-Processor Control Register (MPCR)
00000804-00000807	Multi-Processor Address Register (MPAR)
00000808-000037FF	Work area and stack for MVME141 debug monitor

EPROM LOCATIONS	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

FFF1FFFA-FFF1FFFB 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  
 \$FF800000 -\$FF83FFFF in sockets U78 (even), U56 (odd)

BATTERY BACKED-UP  
 RAM LOCATIONS

## FUNCTION

FFFE0000-FFFE03FF Reserved for user  
 FFFE0400-FFFE05FF Reserved for operating system use  
 FFFE0600-FFFE07F7 Reserved for bug use  
 FFFE0774-FFFE0777 End of memory + 1, set via memory sizing routine  
 FFFE0778-FFFE077A Memory sizing flag  
 FFFE077B  
 FFFE077C  
 FFFE077E  
 FFFE0782  
 FFFE0788  
 FFFE078E  
 FFFE0794  
 FFFE079A  
 FFFE07A0 Reserved  
 FFFE07A6  
 FFFE07A7  
 FFFE07C6 AUTOBOOT controller number, set via the AB command  
 FFFE07C7 AUTOBOOT device number, set via the AB command  
 FFFE07C8-FFFE07E3 AUTOBOOT string, set via the AB command  
 FFFE07E4-FFFE07E9 Off-board address, set via the OBA command  
 FFFE07EA-FFFE07EF ROMboot direct address, set via the RB command  
 FFFE07F0 AUTOBOOT enable switch, set via the [NO] AB command (Y/N)  
 FFFE07F1 AUTOBOOT at power-up switch, set via the AB command (Y/N)  
 FFFE07F2 ROMboot enable switch, set via the [NO] RB command (Y/N)  
 FFFE07F3 ROMboot from VMEbus switch, set via the RB command (Y/N)  
 FFFE07F4 ROMboot at power-up switch, set via the RB command (Y/N)  
 FFFE07F5 Reserved  
 FFFE07F6 Bug/System switch, set via the ENV command (B/S)  
 FFFE07F7 SYSTEM V/68 or VERSAdos switch, set via ENV command (S/V)  
 FFF507F8-FFF507FF Time of day clock

I/O HARDWARE  
 ADDRESSES

## FUNCTION

FFFE3002-FFFE3003 Serial port 1  
 FFFE3000-FFFE3001 Serial port 2  
 FFFE1000-FFFE102F VSB  
 FFFE2000-FFFE201F VME gate array registers

## 1.8 DISK I/O SUPPORT

141Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 141Bug consist of command-level disk operations, disk I/O system calls (only via the TRAP #15 instruction) for use by user programs, and defined data structures for disk parameters.

Parameters such as the address where the module is mapped and the type and number of devices attached to the controller module are kept in tables by 141Bug. Default values for these parameters are assigned at power-up and cold-start reset, but may be altered as described in paragraph 1.9.4.

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

### 1.8.1 Blocks Versus Sectors

The logical block defines the unit of information for disk devices. A disk is viewed by 141Bug 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. 141Bug translates this into an equivalent sector specification, which is then passed on to the controller to initiate the transfer. If the conversion from blocks to sectors yields a fractional sector count, an error is returned and no data is transferred.

### 1.8.2 Disk I/O via 141Bug Commands

These following 141Bug commands are provided for disk I/O. Detailed instructions for their use are found in Chapter 3. When a command is issued to a particular controller LUN and device LUN, these LUNs are remembered by 141Bug so that the next disk command defaults to use the same controller and device.

#### 1.8.2.1 IOP (Physical I/O to Disk)

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

**1.8.2.2 IOT (I/O Teach)**

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

**1.8.2.3 IOC (I/O Control)**

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

**1.8.2.4 BO (Bootstrap Operating System)**

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

**1.8.2.5 BH (Bootstrap and Halt)**

BH reads an operating system or control program from a specified device into memory, and then returns control to 141Bug. It is used as a debugging tool.

**1.8.3 Disk I/O via 141Bug System Calls**

All operations that actually access the disk are done directly or indirectly by 141Bug TRAP #15 system calls. (The command-level disk operations provide a convenient way of using these calls without writing and executing a program.)

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

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

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

To perform a disk operation, 141Bug must eventually present a particular disk controller module with a controller command packet which has been especially prepared for that type of controller module. (This is accomplished in the respective controller driver module.) A command packet for one type of controller module usually does not have the same format as a command packet

for a different type of module. The system call facilities which do disk I/O accept a generalized (controller-independent) packet format as an argument, and translate it into a controller-specific packet, which is then sent to the specified device. Refer to the 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 be given vary from controller to controller. The disk driver module for the particular hardware module (board) must take the standardized packet given to a trap function and create a new packet which is specifically tailored for the disk drive controller it is sent to. Refer to documentation on the particular controller module for the format of its packets, and for using the IOC command.

#### 1.8.4 Default 141Bug Controller and Device Parameters

141Bug initializes the parameter tables for a default configuration of controllers and devices (refer to Appendix E). If the system needs to be configured differently than this default configuration (for example, to use a 70Mb Winchester drive where the default is a 40Mb Winchester drive), then these tables must be changed.

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

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

#### 1.8.5 Disk I/O Error Codes

141Bug returns an error code if an attempted disk operation is unsuccessful. Refer to Appendix F for an explanation of disk I/O error codes.

### 1.9 MULTIPROCESSOR SUPPORT

The MVME141Bug uses RAM starting at address \$0 or the onboard RAM at \$FFF40000. If the onboard RAM is being used, the following MPCR method of multi-processor control does not work. Therefore, the MVME141 uses the VMEchip to implement the MPCR. It is described following the description of the MPCR method.

### 1.9.1 Multi-Processor Control Register (MPCR) Method

A processor can initiate program execution on the MVME141 by issuing a remote GO command using the MPCRO. The MPCR, located at \$800, contains one of two longwords used to control communication between processors. The MPCR contents are organized as follows:

```

$800      +-----+
           | * | N/A | N/A | N/A | (MPCR)
           +-----+
  
```

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

- . Status returned (from the monitor)
- . Status set by the bus master (job requested by some processor)

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

```

ASCII NUL (HEX 00) -- Wait. Initialization not yet complete.
ASCII R   (HEX 52) -- Ready. The firmware is ready for a command.
ASCII E   (HEX 45) -- Executing. The firmware is executing a command.
  
```

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

```

ASCII G (HEX 47) -- GD. Use the firmware command GD.
ASCII B (HEX 42) -- GO. Use the firmware command GO.
  
```

#### NOTE

The address used by the above two commands is placed in the Multi-Processor Address Register (MPAR) before the command G or B is set in the MPCR.

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

```

$804      +-----+
           | * | * | * | * | (MPAR)
           +-----+
           (Longword address)
  
```

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

The MPCR contains \$00 at power-up, indicating that initialization is not yet complete. As the initialization proceeds, the execution path comes to the "prompt" routine. Before sending the prompt, this routine places an R in the MPCR to indicate that initialization is complete. This routine prints the prompt and searches for a command string. During the command string search, if a G or B is detected, it is executed.

If no terminal is connected to the port, the MPCR is still polled to see whether an external processor requires control. If a terminal is connected, the MPCR is polled even if the user is entering commands.

An ASCII G placed in the MPCR by a remote processor indicates that the Go Direct type of transfer is requested. An ASCII B in the MPCR indicates that breakpoints are to be armed before control is transferred to the address in the MPAR. This is the same command as the GO command.

In either sequence of placing a G or B in the MPCR, an E is placed in the MPCR to indicate that execution is underway just before control is passed to the MPAR 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.

### 1.9.2 VMEchip Method

The MPCR functions the same in this mode. If the MPCR is equal to R, then the VMEchip global register number 1 bit 0 (SIGLP) is checked for being set. If not set, nothing happens. If it is set, general purpose control registers 1, 2, 3, and 4 are read and form the execution address. A B is put in the MPCR and the execution address is placed in the MPAR. The SIGLP bit is cleared and the debug monitor called to run the command. The MPCR and MPAR are then handled the same as in the MPCR method.

The VMEchip global registers are accessed in the VMEbus short I/O space (refer to the VMEchip manual for specific details). Each register is a single byte occurring at an odd address. The global register number 1 is at an offset of \$3 from the start of the VMEchip global registers. The base address is selected by header J1 pins 5 through 20 on the MVME141 module. The execution address is formed by reading General Purpose Control and Status Registers (GPR number) in the following manner:

- GPR0 - not used.
- GPR1 - used as the MSB of the address.
- GPR2 - used as the next byte of the address.
- GPR3 - used as the next byte of the address.
- GPR4 - used as the LSB of the address.

The address appears as:

```

+-----+-----+-----+-----+
| GPR1 | GPR2 | GPR3 | GPR4 |
+-----+-----+-----+-----+
(Longword address)

```

## 1.10 DIAGNOSTIC FACILITIES

Included in the 141Bug package is a complete set of hardware diagnostics intended for testing and troubleshooting of the MVME141 (refer to Chapter 6). In order to use the diagnostics, the user must switch directories to the diagnostic directory. If in the debugger directory, the user can switch to the diagnostic directory by entering the debugger command Switch Directories (SD). The diagnostic prompt ("141-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.

## 1.11 REFERENCE MANUALS

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

DOCUMENT TITLE	MOTOROLA PUBLICATION NUMBER
MVME050 System Controller Module and MVME701/MVME701A I/O Transition Module User's Manual	MVME050
MVME141 VMEmodule 32-Bit Microcomputer User's Manual	MVME141
MVME319 Intelligent Disk/Tape Controller User's Manual	MVME319
MVME320 VMEbus Disk Controller Module User's Manual	MVME320
MVME320A VMEbus Disk Controller Module User's Manual	MVME320A
MVME320B VMEbus Disk Controller Module User's Manual	MVME320B
MVME321 Intelligent Disk Controller User's Manual	MVME321
MVME321 IPC Firmware User's Guide	MVME321FW
MVME322 ESDI Disk Controller VMEmodule User's Manual	MVME322
MVME327A VMEbus to SCSI Bus Adapter User's Manual	MVME327A
MVME350 Streaming Tape Controller VMEmodule User's Manual	MVME350
MVME350 IPC Firmware User's Guide	MVME350FW

GENERAL INFORMATION

MVME360 SMD Disk Controller User's Manual	MVME360
VERSAdos to VME Hardware and Software Configuration User's Manual	MVMEVDOS
MC68030 32-Bit Microprocessor User's Manual	MC68030UM
MC68882 Floating-Point Coprocessor User's Manual	MC68882UM
MC68681 Dual Asynchronous Receiver/Transmitter (DUART) Data Book	MC68681

=====

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

V/ESDI 4201 Panther High-performance VMEbus Enhanced Small Device Interface (ESDI) Disk Controller User's Guide; Interphase Corporation, 2925 Merrell Road, Dallas, TX 75229-9990

## CHAPTER 2 - USING THE 141Bug DEBUGGER

## 2.1 ENTERING DEBUGGER COMMAND LINES

141Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt ("141-Bug>") appears on the terminal screen, then the debugger is ready to accept commands.

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

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

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

- a. The command identifier (i.e., "MD" or "md" for the Memory Display command). Note that either uppercase or lowercase is allowed.
- b. A port number if the command is set up to work with more than one port.
- c. At least one intervening space before the first argument.
- d. Any required arguments, as specified by command.
- e. An option field, set off by a semicolon (;) to specify conditions other than the default conditions of the command.

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

**NOTE**

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

- ^X (cancel line) The cursor is backspaced to the beginning of the line. If the terminal port is configured with the hardcopy or TTY option (refer to 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.
- <DEL> (delete or rubout) Performs the same function as ^H.
- ^D (redisplay) The entire command line as entered so far is redisplayed on the following line.

When observing output from any 141Bug 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 141Bug but may be changed by the user using the PF command. In the initialized (default) mode, operation is as follows:

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

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

- boldface strings** A boldface string is a literal such as a command or a program name, and is to be typed just as it appears.
- italic strings* An italic string is a "syntactic variable" and is to be replaced by one of a class of items it represents.
- | A vertical bar separating two or more items indicates that a choice is to be made; one or more of the items separated by this symbol may be selected.
- [ ] Square brackets enclose an item that is optional. The item may appear zero or one time.
- [ ] . . . Square brackets followed by an ellipsis (three dots) enclose an item that is optional/repetitive. The item may appear zero or more times.
- [ ] Boldface brackets are required characters.

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

### 2.1.1 Syntactic Variables

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

<i>del</i>	Delimiter; either a comma or a space.
<i>exp</i>	Expression (described in detail in paragraph 2.1.1.1).
<i>addr</i>	Address (described in detail in paragraph 2.1.1.2).
<i>count</i>	Count; the syntax is the same as for <i>exp</i> .
<i>range</i>	A range of memory addresses which may be specified either by <i>addr del addr</i> or by <i>addr : count</i> .
<i>text</i>	An ASCII string of up to 255 characters, delimited at each end by the single quote mark (').

#### 2.1.1.1 Expression as a Parameter

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

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

BASE	IDENTIFIER	EXAMPLES
Hexadecimal	\$	\$FFFFFFF
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      NUMERIC VALUE
LITERAL    (IN HEXADECIMAL)
=====
'A'         41
'ABC'      414243
'TEST'     54455354
=====

```

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

Valid expression examples:

```

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

```

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

#### 2.1.1.2 Address as a Parameter

Many commands use *addr* as a parameter. The syntax accepted by 141Bug is similar to the one accepted by the MC68030 one-line assembler. All control addressing modes are allowed. An "address + offset register" mode is also provided.

#### Address Formats

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

TABLE 2-1. Formats for Debugger Address Parameters

FORMAT	EXAMPLE	DESCRIPTION
N	140	Absolute address + contents of automatic offset register.
N+Rn	130+R5	Absolute address + contents of the specified offset register (not an assembler-accepted syntax).
(An)	(A1)	Address register indirect.
(d,An) or d(An)	(120,A1) 120(A1)	Address register indirect with displacement (two formats accepted).
(d,An,Xn) or d(An,Xn)	(&120,A1,D2) &120(A1,D2)	Address register indirect with index & displacement (two formats accepted).
([bd,An,Xn],od)	([C,A2,A3],&100)	Memory indirect pre-indexed.
([bd,An],Xn,od)	([12,A3],D2,&10)	Memory indirect post-indexed.

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

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

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

### Offset Registers

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

USING THE 141Bug DEGUGGER

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

**NOTE**

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

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

```

1
2
3
4
5 0 00000000 48E78080      MOVESTR  MOVEM.L  DO/A0,-(A7)
6 0 00000004 4280          CLR.L    DO
7 0 00000006 1018          MOVE.B   (A0)+,DO
8 0 00000008 5340          SUBQ.W   #1,DO
9 0 0000000A 12D8          LOOP     MOVE.B   (A0)+,(A1)+
10 0 0000000C 51C8FFFC     MOVS     DBRA    DO,LOOP
11 0 00000010 4CDF0101     MOVEM.L (A7)+,DO/AO
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:

```

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

```

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

```

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

```

For additional information about the offset registers, refer to the Offset Registers (OF) command description.

### 2.1.2 Port Numbers

Some 141Bug commands give the user the option of choosing the port which is to be used to input or output. The valid port numbers which may be used for these commands are:

- 0 - MVME141 RS-232C serial port 1)
- 1 - MVME141 RS-232C serial port 2)

#### NOTE

These logical port numbers (0 and 1) are referred to as "Serial Port 1" and "Serial Port 2", respectively, by the MVME141 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 MVME141 front panel.

## 2.2 ENTERING AND DEBUGGING PROGRAMS

There are various ways to enter a user program into system memory for execution. One way is to create the program using the Memory Modify (MM) command with the assembler/disassembler option. The program is entered by the user one source line at a time. After each source line is entered, it is assembled and the object code is loaded to memory. Refer to Chapter 4 for complete details of the 141Bug 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 141Bug MM command as outlined above and stored to the host using the Dump (DU) command. A communication link must exist between the host system and the MVME141 port B. (Refer to hardware configuration details in paragraph 1.3.) The file is downloaded from the host into system memory via the debugger Load (LO) command.

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

### 2.3 CALLING SYSTEM UTILITIES FROM USER PROGRAMS

A convenient way of doing character input/output and many other useful operations has been provided so that the user does not have to write these routines into the target code. The user has access to various 141Bug routines via the MC68030 TRAP #15 instruction. Refer to Chapter 5 for details on the various TRAP #15 utilities available and how to invoke them from within a user program.

### 2.4 PRESERVING THE DEBUGGER OPERATING ENVIRONMENT

This paragraph explains how to avoid contaminating the operating environment of the debugger. 141Bug uses certain of the MVME141 onboard resources and may also use offboard system memory to contain temporary variables, exception vectors, etc. If the user disturbs resources upon which 141Bug depends, then the debugger may function unreliably or not at all.

#### 2.4.1 141Bug Vector Table and Wordspace

As described in Chapter 1, "Memory Requirements", 141Bug needs 12Kb of read/write memory to operate and also allocates another 4Kb as user space for a total of 16Kb allocated. 141Bug 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, 141Bug reserves space for static variables and initializes these static variables to predefined default values. After the static variables, 141Bug allocates space for the system stack and then initializes the system stack pointer to the top of this area.

With the exception of the first 1024-byte vector table area, the user must be extremely careful not to use the above-mentioned areas for other purposes. The user should refer to paragraph 1.7 and to Appendix A to determine how to dictate the location of the reserved memory areas. If, for example, a user program inadvertently wrote over the static variable area containing the serial communication parameters, these parameters would be lost, resulting in a loss of communication with the system console terminal. If a user program corrupts the system stack, then an incorrect value may be loaded into the processor PC, causing a system crash.

2.4.2 Exception Vectors Used by 141Bug

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

TABLE 2-2. Exception Vectors Used by 141Bug

VECTOR OFFSET	EXCEPTION	141Bug FACILITY
\$10	Illegal instruction	Breakpoints (used by BR, GO, GN, GT)
\$24	Trace	T, TC, TT
\$BC	TRAP #15	System calls (refer to Chapter 5)
\$108	Level 7 interrupt	ABORT switch

When the debugger handles one of the exceptions listed above, the target stack pointer is left pointing past the bottom of the exception stack frame created; that is, it reflects the system stack pointer values just before the exception occurred. In this way, the operation of the debugger facility (through an exception) is transparent to the user. Example: Trace one instruction using the debugger.

```

141-Bug> RD
PC =00004000 SR =2700=TR:OFF S. 7 ..... VBR =00000000
USP =0000F830 MSP =00005C18 ISP* =00006000 SFC =0=F0
CACR =0=D:.... I:... CAAR =00000000 DFC =0=F0
D0 =00000000 D1 =00000000 D2 =00000000 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
141-Bug> T
PC =00004000 SR =2700=TR:OFF S. 7 ..... VBR =00000000
USP =0000F830 MSP =00005C18 ISP* =00006000 SFC =0=F0
CACR =0=D:.... I:... CAAR =00000000 DFC =0=F0
D0 =00000000 D1 =00000000 D2 =00000000 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
141-Bug>
    
```

Notice that the value of the target stack pointer register (A7) has not changed even though a trace exception has taken place. The user program may either use the exception vector table provided by 141Bug or it may create a separate exception vector table of its own. The two following paragraphs detail these two methods.

#### 2.4.2.1 Using 141Bug Target Vector Table

141Bug initializes and maintains a vector table area for target programs. A target program is any user program started by the bug, either manually with GO or Trace type commands or automatically with the B0ot command. The start address of this target vector table area is the base address of the MVME141, determined as described in paragraph 1.7. This address is loaded into the target-state Vector Base Register (VBR) at power-up and cold-start reset and can be observed by using the RD command to display the target-state registers immediately after power-up.

141Bug initializes the target vector table with the debugger vectors listed in Table 2-3 and fills the other vector locations with the address of a generalized exception handler (refer to paragraph 2.4.3.3). The target program may take over as many vectors as desired by simply writing its own exception vectors into the table. If the vector locations listed in Table 2-3 are overwritten, then the accompanying debugger functions are lost.

141Bug maintains a separate vector table for its own use in a 1Kb space elsewhere in the reserved memory space. In general, the user does not have to be aware of the existence of the debugger vector table. It is completely transparent to the user and the user should never make any modifications to the vectors contained in it.

#### 2.4.2.2 Creating a New Vector Table

A user program may create a separate vector table in memory to contain its exception vectors. If this is done, then the user program must change the value of the VBR to point at the new vector table. In order to use the debugger facilities, the user can copy the proper vectors from the 141Bug vector table into the corresponding vector locations in the user vector table.

The vector for the 141Bug generalized exception handler (described in detail in paragraph 2.4.3.3) may be copied from offset \$08 (Bus Error vector) in the target vector table to all locations in the user vector table where a separate exception handler is not used. This provides diagnostic support in the event that the user program is stopped by an unexpected exception. The generalized exception handler gives a formatted display of the target registers and identifies the type of exception. Example: a user routine which builds a separate vector table and then moves the VBR to point at it:

```

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

```

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

When an exception occurs which the user wants to pass on to the debugger (ABORT, for example) the user exception handler must read the vector offset from the format word of the exception stack frame. This offset is added to the address of the 141Bug target program vector table (which the user program saved), yielding the address of the 141Bug exception vector. The user program then jumps to the address stored at this vector location, which is the address of the 141Bug exception handler.

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

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

```

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

### 2.4.2.3 141Bug Generalized Exception Handler

141Bug 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. Thus, if an unexpected exception occurs during execution of a user code segment, the user is presented with the exception stack frame to help determine the cause of the exception. The following example illustrates this:

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

```

141-Bug> RD
PC =00004000 SR =2700=TR:OFF S_7_..... VBR =00000000
USP =0000F830 MSP =00005C18 ISP* =00006000 SFC =0=F0
CACR =0=D:.... I:... CAAR =00000000 DFC =0=F0
D0 =00000000 D1 =00000000 D2 =00000000 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
141-Bug> T
PC =00004000 SR =2700=TR:OFF S_7_..... VBR =00000000
USP =0000F830 MSP =00005C18 ISP* =00006000 SFC =0=F0
CACR =0=D:.... I:... CAAR =00000000 DFC =0=F0
D0 =00000000 D1 =00000000 D2 =00000000 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
141-Bug>
    
```

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

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

## 2.5 MEMORY MANAGEMENT UNIT SUPPORT

The Memory Management Unit (MMU) is supported in the 141Bug. An MMU confidence check is run at reset time to verify that registers can be accessed. It also insures that a context switch can be done successfully. Also, the commands RD, RM, MD, and MM have been extended to allow display and modification of MMU data in registers and in memory. MMU instructions can be assembled/disassembled with the DI option of the MD/MM commands. In addition, the MMU target state is saved and restored along with the processor state as required when switching between the target program and 141Bug. Finally, there is a set of diagnostics to test functionality of the MMU.

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

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

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

```
Address Translation Control (ATC) registers:
CRP          - CPU Root Pointer Register          (DL)
SRP          - Supervisor Root Pointer           (DL)
TC           - Translation Control Register       (L)
TTO         - Transparent Translation 0           (L)
TTI         - Transparent Translation 1           (L)

Status Information registers:
MMUSR       - MMU Status Register                (W)
```

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

2.6 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 were changed to allow the specification of function codes:

```

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

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

*addr^fc*

The valid function code mnemonics are:

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

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

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

```

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

## CHAPTER 3 - THE 141Bug DEBUGGER COMMAND SET

## 3.1 INTRODUCTION

This chapter contains descriptions of each debugger command, with one or more examples of each. 141Bug debugger commands are summarized in Table 3-1.

TABLE 3-1. Debugger Commands

COMMAND MNEMONIC	TITLE
AB/NOAB	Autoboot Enable/Disable
BC	Block Compare
BF	Block of Memory Fill
BH	Bootstrap Operating System and Halt
BI	Block of Memory Initialize
BM	Block of Memory Move
BO	Bootstrap Operating System
BR/NOBR	Breakpoint Insert/Delete
BS	Block of Memory Search
BV	Block of Memory Verify
CS	Checksum
DC	Data Conversion
DU	Dump S-records
EEP	EEPROM Programming
ENV	Set Environment to Bug or Operating System
GD	Go Direct (Ignore Breakpoints)
GN	Go to Next Instruction
GO	Go Execute User Program
GT	Go to Temporary Breakpoint
HE	Help
IOC	I/O Control for Disk
IOP	I/O Physical (Direct Disk Access)
IOT	I/O "TEACH" for Configuring Disk Controller
LO	Load S-records from Host
LSAD	LAN Station Address Display/Set
MA/NOMA	Macro Define/Display/Delete
MAE	Macro Edit
MAL/NOMAL	Enable/Disable Macro Expansion Listing
MAR/MAW	Save/Load Macros
MD	Memory Display
MENU	System Menu
MM	Memory Modify
MS	Memory Set
OBA	Set Memory Address from VMEbus
OF	Offset Registers Display/Modify
PA/NOPA	Printer Attach/Detach
PF/NOFF	Port Format/Detach
PS	Put RTC into Power Save Mode for Storage

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

COMMAND MNEMONIC PARAGRAPH	TITLE
RB/NORB	ROMboot Enable/Disable
RD	Register Display
REMOTE	Connect the Remote Modem to CSO
RESET	Cold/Warm Reset
RM	Register Modify
RS	Register Set
SD	Switch Directories
SET	Set Time and Date
T	Trace
TA	Terminal Attach
TC	Trace on Change of Control Flow
TIME	Display Time and Date
TM	Transparent Mode
TT	Trace to Temporary Breakpoint
VE	Verify S-records Against Memory

Each of the individual commands is described in the following pages. The command syntax is shown using the symbols explained in paragraph 2.1.

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 141Bug). The symbol **(CR)** represents the carriage return key on the user's terminal keyboard. The **(CR)** is shown only if the carriage return is the only user input.

## 3.2 AUTOBOOT ENABLE/DISABLE

AB  
NOABAB  
NOAB

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

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

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

Example:

```
141-Bug> ab                               Enable the autoboot function.
Controller LUN = 00 ? (CR)                 Select controller for boot.
Device LUN     = 00 ? (CR)                 Select device to boot from.
Default string = VME141.. ? (CR)          Select string to pass on.
Boot at power-up only [Y,N] ? Y (CR)      If the user selects N, the
                                           MVME141 boots at any board
                                           reset.
```

At power-up only:

```
Auto Boot from Controller 0, Device 0, VME141..
141-Bug> noab                             NOAB disables the autoboot
No Auto Boot from Controller 0, Device 0, VME141.. function, but does not
141-Bug>                                  change the parameters.
```

3.3 BLOCK OF MEMORY COMPARE

BC

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

options:

- B - Byte
- W - Word
- L - Longword

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

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

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

```
141-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 .....
```

```
141-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:

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

Example 2:

```
141-Bug> BC 20000:20 21000;B
Effective address: 00020000
Effective count : &32
Effective address: 00021000
141-Bug> (memory compares, nothing printed)
```

BC

Example 3:

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

(create a mismatch)

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

(mismatches are printed out)

3.4 BLOCK OF MEMORY FILL

BF

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

where:

*data* and *increment* are both expression parameters

options (length of data field):

- B - Byte
- W - Word
- L - Longword

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

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

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

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

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

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

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



## 3.5 BOOTSTRAP OPERATING SYSTEM AND HALT

BH

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

where:

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

BH is used to load an operating system or control program from disk into memory. This command works in exactly the same way as the B0 command, except that control is not given to the loaded program. After the registers are initialized, control is returned to the 141Bug debugger and the prompt reappears on the terminal screen. Because control is retained by 141Bug, all the 141Bug facilities are available for debugging the loaded program if necessary.

Examples:

```
141-Bug> bh 0,1           Boot and halt from controller 0, device LUN 1.
141-Bug>
```

```
141-Bug> bh 3,a,test2;d   Boot and halt from controller 3, device LUN $A,
141-Bug>                  and pass the string "test2;d" to the loaded
                           program.
```

Refer to the B0 command description for more detailed information about what happens during bootstrap loading.

## 3.6 BLOCK OF MEMORY INITIALIZE

BI

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

options:

B - Byte  
 W - Word  
 L - Longword

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

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

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

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

Example 1:

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

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

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

## 3.7 BLOCK OF MEMORY MOVE

BM

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

options:

B - Byte  
 W - Word  
 L - Longword

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

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

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

```
141-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 .....
```

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

```
141-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 .....
```

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

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

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

BM

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

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

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

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

```
141-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
141-Bug>
```

3

## 3.8 BOOTSTRAP OPERATING SYSTEM

B0

B0 [*controller LUN*][*del device LUN*][*del string*]

where:

- controller LUN* - is the LUN of the controller to which the above device is attached. Defaults to LUN 0.
- device LUN* - is the Logical Unit Number (LUN) of the device to boot from. Defaults to LUN 0.
- del* - is a field delimiter: Comma (,) or spaces ( ).
- string* - is a string that is passed to the operating system or control program loaded. Its syntax and use is completely defined by the loaded program.

B0 is used to load an operating system or control program from disk into memory and give control to it. Where to find the program and where in memory to load it is contained in block 0 of the device LUN specified. (Refer to Appendix D.) The device configuration information is located in block 1 (Appendix D). The controller and device configurations used when B0 is initiated can be examined and changed via the I/O Teach (IOT) command.

The following sequence of events occurs when B0 is invoked:

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

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

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

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

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

B0

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

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

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

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

PC = Entry point of loaded program (loaded from "pseudo reset vector").

SR = \$2700.

D0 = Device LUN.

D1 = Controller LUN.

D4 = Flags for IPL; 'IPLx', with x = bits	7654	3210
Reserved		00
Firmware support for TRAP #15		1
Firmware support IPL Disk I/O		1
Firmware support for SCSI streaming tape		0
Firmware support for TRAP #15 ID Packet		1
Unused (Reserved)		00

A0 = Address of disk controller.

A1 = Entry point of loaded program.

A2 = Address of media configuration block. Zero if no configuration loaded.

A5 = Start of string (after command parameters).

A6 = End of string + 1 (if no string was entered A5 = A6).

A7 = Initial stack pointer (loaded from "pseudo reset vector").

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

Examples:

141-Bug> B0                      Boot from default controller LUN and device LUN as defined by the AB command.

141-Bug> B0,3                    Boot from default controller LUN, device LUN 3.

141-Bug> bo3                     Boot from controller LUN 3, default device LUN.

141-Bug> bo 0,8,test            Boot from controller LUN 0, device LUN 8, and pass the string "test" to the booted program.

## 3.9 BREAKPOINT INSERT/DELETE

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

The BR command allows the user to set a target code instruction address as a "breakpoint address" for debugging purposes. If, during target code execution, a breakpoint with 0 count is found, the target code state is saved in the target registers and control is returned back to 141Bug. This allows the user to see the actual state of the processor at selected instructions in the code.

Up to eight breakpoints can be defined. The breakpoints are kept in a table which is displayed each time either BR or NOBR is used. If an address is specified with the BR command, that address is added to the breakpoint table. The count field specifies how many times the instruction at the breakpoint address must be fetched before a breakpoint is taken. The count, if greater than zero, is decremented with each fetch. Every time that a breakpoint with zero count is found, a breakpoint handler routine prints the CPU state on the screen and control is returned to 141Bug.

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

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

Example:

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

## 3.10 BLOCK OF MEMORY SEARCH

BS

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

or

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

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

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

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

- a. The user-entered data pattern is right-justified and leading bits are truncated or leading zeros are added as necessary to make the data pattern the specified size.
- b. A compare is made with successive bytes, words, or longwords (depending on the size in effect) within the range for a match with the user-entered data. Comparison is made only on those bits at bit positions corresponding to a "1" in the mask. If no mask is specified, then a default mask of all ones is used (all bits are compared). The size of the mask is taken to be the same size as the data.
- c. If the N (non-aligned) option has been selected, then the data is searched for on a byte-by-byte basis, rather than by words or longwords, regardless of the size of *data*. This is useful if a word (or longword) pattern is being searched for, but is not expected to lie on a word (or longword) boundary.
- d. If a match is found, then the address of the first byte of the match is output along with the memory contents. If a mask was in use, then the actual data at the memory location is displayed, rather than the data with the mask applied.

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

BS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

BS

141-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.

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

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

141-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.

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

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

141-Bug>

## 3.11 BLOCK OF MEMORY VERIFY

BV

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

where:

*data* and *increment* are both expression parameters

options:

B - Byte  
W - Word  
L - Longword

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

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

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

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

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

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



## 3.12 CHECKSUM

CS

CS *address1 address2*

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

- a. At power-up, the power-up confidence test is executed. One of the items verified is the checksum contained in the firmware monitor EPROM. If for any reason the contents of the EPROM were to change from the factory version, the checksum test is designed to detect the change and inform the user of the failure.
- b. Following a valid power-up test, 141Bug 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 paragraph 1.5. which describes the format of the routine to be executed and the interface provided upon entry.

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

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

- a. An absolute address (32-bit maximum).
- b. An expression using a displacement + relative offset register.

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

The algorithm used to calculate the checksum is as follows:

- a. \$FF is placed in each of two bytes within a register. These bytes represent the even and odd bytes as the checksum is calculated.
- b. Starting with *address1* the even and odd bytes are extracted from memory and XORed with the bytes in the register.
- c. This process is repeated, word by word, until *address2* is reached. This technique allows use of even ending addresses (\$D40000 as opposed to \$D3FFFF).

EXAMPLE

COMMENT

141-Bug> MD 20000:3F;B

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

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

141-Bug> M 20010;DI

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

00020010	41F90001 F000	LEA.L	(\$1F000).L,A0 ?(CR)
00020016	203C0000 EFFF	MOVE.L	#\$EFFF,DO ?(CR)
0002001C	1100	MOVE.B	DO,-(A0) ?(CR)
0002001E	51C8FFFC	DBF.W	DO,\$2001C ?(CR)
00020022	4E75	RTS ?	(CR)
00020024	0101	BTST.L	DO,D1 ?(CR) 0101 is revision.
00020026	0000	DC.W	\$0 ?(CR) 0000 is where checksum is to be placed.
00020028	FFFF	DC.W	\$FFFF ?(CR) FFFF is unused memory.
0002002A	FFFF	DC.W	\$FFFF ?(CR) FFFF is unused memory.
0002002C	FFFF	DC.W	\$FFFF ?(CR) FFFF is unused memory.
0002002E	FFFF	DC.W	\$FFFF ?(CR) FFFF is unused memory.
00020030	FFFF	DC.W	\$FFFF ?. FFFF is unused memory.



EXAMPLE (Using Absolute Addresses)COMMENT

141-Bug> CS 20000 20028

Request checksum of area using absolute addresses.

Effective address: 00020000

Effective address: 00020027

Even/Odd = \$5B3C

Checksum of even bytes is \$5B.  
Checksum of odd bytes is \$3C.

141-Bug> M 20026;W

Place these bytes in zeroed area used while calculating checksum.

00020026 0000 ?5B3C.

141-Bug> CS 20000 20028

Verify checksum.

Effective address: 00020000

Effective address: 00020027

Even/Odd = \$0000

Result is 0000, good checksum.

EXAMPLE (Using Relative Offset)COMMENT

141-Bug> OF R3

R3 =00000000 00000000? 20000 .

Define value of relative offset register 3.

141-Bug> CS 0+R3 28+R3

Request checksum of area using relative offset.

Effective address: 00000+R3

Effective address: 00027+R3

Even/Odd = \$5B3C

Checksum of even bytes is \$5B.  
Checksum of odd bytes is \$3C.

141-Bug> M 26+R3;W

Place these bytes in zeroed area used while checksum was calculated.

00000026+R3 0000 ?5B3C.

141-Bug> CS 0+R3 28+R3

Verify checksum.

Effective address: 00000+R3

Effective address: 00027+R3

Even/Odd = \$0000

141-Bug>

## 3.13 DATA CONVERSION

DC

DC *exp* | *addr*

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

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

Examples:

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

141-Bug> DC &10-&20
SIGNED : FFFFFFF6 = -$A = -&10
UNSIGNED: FFFFFFF6 = $FFFFFF6 = &4294967286

141-Bug> DC 123+&345+067+%1100001
      00000314 = $314 = &788

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

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

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

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

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

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

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

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

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

## 3.14 DUMP S-RECORDS

DU

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

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

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

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

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

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

**NOTE**

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

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

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

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

```
141-Bug> DU 0 30000:&10
Effective address: 00030000
Effective count : &10
S0030000FC
S20E03000026025445535466084E4F7B
S9030000FC
141-Bug>
```

DU

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

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

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

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

BREAK (Press BREAK key to get VERSAdos login
prompt.)

"
login (User must log onto VERSAdos and enter the
" catalog where FILE1.MX will reside.)
"

=UPLOADS FILE1 (At VERSAdos prompt, invoke the UPLOADS
utility and tell it to create a file
named "FILE1" for the S-records that will
be uploaded).
```

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

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

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

```
UPLOADS Allocating new file
```

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

```
=^A (When the VERSAdos prompt returns, enter
141-Bug> the escape character to return to
141Bug).
```

DU

Now enter the command for 141Bug to dump the S-records to the port.

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

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

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

The UPLOADS utility now displays some more messages like this:

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

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

```
*STATUS* No error since start of program
```

```
Upload of S-Records complete.
```

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

```
^A (Enter the escape character to return to
141-Bug)
141-Bug>
```

## 3.15 EEPROM PROGRAMMING

*not present in 141*

EEP

EEP range del addr [:W]

options:

W - Word (default)

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

Example 1: (Assumes EEPROMs installed in U16 and U18 (bank 2), and J4 configured for the right size EEPROMs. Refer to the MVME141 User's Manual for jumper details. U16 and U18 are at addresses starting at \$FFA00000 and ending at or below \$FFBFFFFF in the main memory map, with the odd-byte chip in U18 and the even-byte chip in U16. Note that 141Bug is in the EPROMs in U1 and U2 (bank 1), at \$FF800000 through \$FF83FFFF, with odd bytes in U2 and even bytes in U1.)

```
141-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    .....
```

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

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

Example 2:

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

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

## 3.16 SET ENVIRONMENT TO BUG/OPERATING SYSTEM

ENV

## ENV

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

Two Bug modes are available:

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

Two operating system modes are available:

- SYSTEM V/68 - This is the standard system mode, and is the one defaulted to if BBRAM should fail. In this mode, the MVME141 disk controller default configurations are for 512b sectors.
- VERSAdos     - In this mode, the MVME141 disk controller default configurations are for 256b sectors.

Example 1:

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

141-Bug>
```

Example 2:

```
141-Bug> ENV
Bug or System environment [B,S] = B? S             (change to system
mode of operation)
SYSTEM V/68 or VERSAdos operating system [S,V] = V? s (change to SYSTEM
V/68 operating
system and execute
memory test)
Execute /Bypass SST Memory Test [E,B] = E? (CR)
```

Firmware now takes the reset path and initializes the MVME141 for the system mode (refer to Appendix A for system mode operation details).

**EVN**

Example 3:

141-Bug> ENV

Bug or System environment [B,S] = S? B

(change to Bug mode)

SYSTEM V/68 or VERSAdos operating system [S,V] = S? V

(change to VERSAdos operating system)

Copyright Motorola Inc. 1988, All Rights Reserved

VME141 Monitor/Debugger Release 1.0 - 4/8/88

FPC passed test  
MMU passed test

Cold Start  
141-Bug>

3

## 3.17 GO DIRECT (IGNORE BREAKPOINTS)

GD

GD [*addr*]

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

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

Once execution of the target code has begun, control may be returned to 141Bug by various conditions:

- User pressed the ABORT or RESET switches on the MVME141 front panel.
- An unexpected exception occurred.
- By execution of the .RETURN TRAP #15 function.

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

```
141-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
141-Bug> RM DO
```

Initialize D0 and start target program:

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

To exit target code, press ABORT switch.

```
Exception: Abort
Format Vector = 007C
PC =0001000E SR =2711=TR:OFF S. 7 X...C
USP =0000F830 MSP =0000FC18 ISP*=00010000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =00052A9C D1 =00000000 D2 =000000FF D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00010000
0001000E 60FE                BRA.B  $1000E
141-Bug>
```

GD

Set PC to start of program and restart target code:

141-Bug> RM PC

PC =0001000E ? 10000.

141-Bug> GD

Effective address: 00010000

## 3.18 GO TO NEXT INSTRUCTION

GN

GN

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

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

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

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

```
141-Bug> MD 6000:4;DI
00006000 7003                MOVEQ.L #3,D0
00006002 7201                MOVEQ.L #1,D1
00006004 6100FFA            BSR.W $7000
00006008 2600                MOVE.L D0,D3
141-Bug>
```

The following simple routine resides at address \$7000.

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

Execute up to the BSR instruction.

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

```
141-Bug> GT 6004
Effective address: 00006004
Effective address: 00006000
At Breakpoint
PC =00006004 SR =2700=TR:OFF S. 7 .....
USP =00003830 MSP =00003C18 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =00000003 D1 =00000001 D2 =00000000 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00006004 6100FFA            BSR.W  $7000
141-Bug>
```

GN

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

```

141-Bug> GN
Effective address: 00006004
At Breakpoint
PC =00006008 SR =2700=TR:OFF S_7 .....
USP =00003830 MSP =00003C18 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
DO =00000004 D1 =00000001 D2 =00000000 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00006008 2600 MOVE.L DO,D3
141-Bug>

```

## 3.19 GO EXECUTE USER PROGRAM

GO

GO [*addr*]

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

The sequence of events is as follows:

- a. First, if an address is specified, it is loaded in the target PC.
- b. Then, if a breakpoint is set at the target PC address, the instruction at the target PC is traced (executed in trace mode).
- c. Next, all breakpoints are inserted in the target code.
- d. Finally, target code execution resumes at the target PC address.

At this point control may be returned to 141Bug by various conditions:

- a. A breakpoint with 0 count was found.
- b. User pressed the ABORT or RESET switches on the MVME141 front panel.
- c. An unexpected exception occurred.
- d. By execution of the .RETURN TRAP #15 function.

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

```
141-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
141-Bug> RM DO
```

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

```
DO =00000000 ? 52A9C.
141-Bug> BR 10000,1000E
BREAKPOINTS
00010000          0001000E
141-Bug> GO 10000
Effective address: 00010000
At Breakpoint
PC =0001000E SR =2011=TR:OFF S. 0 X...C
USP =0000F830 MSP =0000FC18 ISP*=00010000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =00052A9C D1 =00000000 D2 =000000FF D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00010000
0001000E 60FE          BRA.B $1000E
```

60

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

Continue target program execution.

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

Remove breakpoints and restart target code.

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

To exit target code, press the ABORT switch.

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

## 3.20 GO TO TEMPORARY BREAKPOINT

GT

GT *addr*

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

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

After setting the temporary breakpoint, the sequence of events is similar to that of the GO command. At this point control may be returned to 141Bug by various conditions:

- a. A breakpoint with count 0 was found.
- b. User pressed the ABORT or RESET switches on the MVME141 front panel.
- c. An unexpected exception occurred.
- d. By execution of the .RETURN TRAP #15 function.

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

```
141-Bug> MD 10000;D1
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
141-Bug> RM D0
```

Initialize D0 and set a breakpoint:

```
D0 =00000000 ? 52A9C.
```

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

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

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

GT

```

141-Bug> GT 10006
Effective address: 00010006
Effective address: 00010000
At Breakpoint
PC =00010006 SR =2711=TR:OFF S_7 X...C
USP =00003830 MSP =00003C18 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =00052A9C D1 =00000029 D2 =00000009 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010006 E289 LSR.L # $1,D1
141-Bug>

```

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

```

141-Bug> GT 10002
Effective address: 00010006
At Breakpoint
PC =0001000E SR =2711=TR:OFF S_7 X...C
USP =00003830 MSP =00003C18 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =00052A9C D1 =00000000 D2 =000000FF D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
0001000E 60FE BRA.B $1000E
141-Bug>

```

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

3.21 HELP

HE

HE [*command*]

HE command is the 141Bug help facility. HE (CR) displays the command names of all available commands along with their appropriate titles. HE *command* displays only the command name and title for that particular command.

3

## 3.22 I/O CONTROL FOR DISK

IOC

## IOC

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

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

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

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

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

## 3.23 I/O PHYSICAL (DIRECT DISK ACCESS)

IOP

## IOP

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

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

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

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

- a. Controller LUN =00?

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

- b. Device LUN =00?

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

- c. Read/Write/Format =R?

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

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

- d. Memory Address =00003000?

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

## e. Starting Block =00000000?

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

## f. Number of Blocks =0002?

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

## g. Address Modifier =00?

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

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

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

## i. File Number =0000?

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

## j. Flag Byte =00?

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

- Bit 7 File Mark flag. If 1, a file mark was detected at the end of the last operation.
- Bit 1 Ignore File Number (IFN) flag. If 0, the file number field is used to position the tape before any reads or writes are done. If 1, the file number field is ignored, and reads or writes start at the present tape position.
- Bit 0 End of File flag. If 0, reads or writes are done until the specified block count is exhausted. If 1, reads are done until the count is exhausted or until a file mark is found. If 1, writes are terminated with a file mark.

## IOP

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

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

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

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

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

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

```
141-Bug> IOP
Controller LUN =00? (CR)
Device LUN =00? 2
Read/Write/Format=R? (CR)
Memory Address =00003000? 50000
Starting Block =00000000? &370
Number of Blocks =0002? &25
Address Modifier =00? (CR)
141-Bug>
```

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

```
141-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)
141-Bug>
```

## 3.24 I/O "TEACH" FOR CONFIGURING DISK CONTROLLER

IOT

IOT [;[H][A]]

The IOT command allows the user to "teach" a new disk configuration to 141Bug for use by the TRAP #15 disk functions. IOT lets the user modify the controller and device descriptor tables used by the TRAP #15 functions for disk access. Note that because the 141Bug commands that access the disk use the TRAP #15 disk functions, changes in the descriptor tables affect all those commands. These commands include IOP, B0, BH, and also any user program that uses the TRAP #15 disk functions.

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

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

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

Example:

```
141-Bug> iot;h
```

## Disk Controllers Available

Lun	Type		Address	# dev
0	VME141	SCSI 10	\$\$\$\$\$4000	1
1	VME141	SCSI 11	\$\$\$\$\$4000	1
2	VME141	SCSI 13	\$\$\$\$\$4000	1
3	VME141	SCSI 0F	\$\$\$\$\$4000	1
4	VME141	SCSI 12	\$\$\$\$\$4000	1
5	VME141	SCSI 12	\$\$\$\$\$4000	1
8	VME320		\$\$\$\$\$B000	4
9	VME350		\$\$\$\$\$5000	1

```
141-Bug>
```

IOT may be invoked with an "A" (All) option specified. This option instructs IOT to list all the disk controllers which are currently supported in 141Bug.

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

## IOT

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

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

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

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

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

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

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

- b. Block Size:  
 0-128 1-256  
 2-512 3-1024           =01?

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

$(\text{Block Size}) * (\text{Number of Blocks}) / (\text{Physical Sector Size})$  must be an integer.

- c. Sectors/Track           =0020?

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

- d. Starting Head           =10?

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

- e. Number of Heads       =05?

This field specifies the number of heads on the drive.

## f. Number of Cylinders =0337?

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

48 TPI - 40 cylinders  
96 TPI - 80 cylinders

## g. Precomp. Cylinder =0000?

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

## h. Reduced Write Current Cylinder =0000?

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

## i. Interleave Factor =00?

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

## j. Spiral Offset =00?

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

## k. ECC Data Burst Length =0000?

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

## l. Step Rate Code =00?

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

The encoding is as follows:

Step Rate Code (Hex)	Winchester Hard Disks	5 1/4-Inch Floppy	8-Inch Floppy
00	0 msec	12 msec	6 msec
01	6 msec	6 msec	3 msec
02	10 msec	12 msec	6 msec
03	15 msec	20 msec	10 msec
04	20 msec	30 msec	15 msec

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

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

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

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

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

o. Single/Equal\_in\_all Track zero density =S (S/E)?

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

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

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

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

q. Gap 1 =07?

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

r. Gap 2 =08?

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

IOT

s. Gap 3 =00?

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

t. Gap 4 =00?

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

u. Spare Sectors Count =00?

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

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

```
141-Bug> IOT
Controller LUN      =00? 8
Device LUN         =00? 2
Sector Size:
0-128 1-256
2-512 3-1024      =01? (CR)
Block Size:
0-128 1-256
2-512 3-1024      =01? (CR)
Sectors/track      =0010? (CR)
Number of heads    =02? (CR)
Number of cylinders =0050? (CR)
Precomp. Cylinder  =0028? (CR)
Step Rate Code     =00? (CR)
Single/Double TRACK density=D (S/D)? (CR)
Single/Double DATA density =D (S/D)? (CR)
Single/Equal_in_all Track zero density =S (S/E)? (CR)
Slow/Fast Data Rate =S (S/F)? (CR)
141-Bug>
```

## IOT

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

```

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

```

## IOT

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

```

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

```

```

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

```

## 3.25 LOAD S-RECORDS FROM HOST

LO

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

The LO command is used when data in the form of a file of Motorola S-records is to be downloaded from a host system to the MVME141. The LO command accepts serial data from the host and loads it into memory.

NOTE

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

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

The optional *addr* field allows the user to enter an offset address which is to be added to the address contained in the address field of each record. This causes the records to be stored to memory at different locations 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 141Bug begins to look for S-records at the host port. This allows the user to send a command to the host device to initiate the download. This text should NOT be delimited by any kind of quote marks. Text is understood to begin immediately following the equals sign and terminate with the carriage return. If the host is operating full duplex, the string is also echoed back to the host port by the host and appears on the user's terminal screen.

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

The other options have the following effects:

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

LO

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

T option - TRAP #15 code. This option causes LO to set the target register D4 = 'LO 'x, with x = \$0C (\$4C4F200C). The ASCII string 'LO ' indicates that this is the LO command; the code \$0C indicates TRAP #15 support with stack parameter/result passing and TRAP #15 disk support. This code can be used by the downloaded program to select the appropriate calling convention when invoking debugger functions, because some Motorola debuggers use conventions different from 141Bug, and they set a different code in D4.

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

If a non-hex character is encountered within the data field of a data record, then the part of the record which had been received up to that time is printed to the screen and the 141Bug 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 141Bug AND if the checksum comparison has not been disabled via the "-C" option, then an error condition exists. A message is output stating the address of the record (as obtained from the address field of the record), the calculated checksum, and the checksum read with the record. A copy of the record is also output. This is a fatal error and causes the command to abort.

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

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

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

```

1
2
3           65040000           ORG           $65040000
4
5 65040000 7001           MOVEQ.L  #1,D0
6 65040002 D088           ADD.L  A0,D0

```

LO

```

7      65040004 4A00      TST.B   DO
8      65040006 4E75      RTS
9

```

```

***** TOTAL ERRORS      0--
***** TOTAL WARNINGS    0--

```

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

```

S00F00005445535453335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091

```

Load this file into MVME141 memory for execution at address \$40000 as follows:

```

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

BREAK                       (Press BREAK key to get VERSAdos login
                             prompt).

"
login                       (User must log onto VERSAdos and enter the
"                             proper catalog to access the file TEST.MX)
"
=^A                         (Enter escape character to return to
                             141Bug prompt).

```

```

141-Bug> LO -65000000 ;X=COPY TEST.MX,#
COPY TEST.MX,#
S00F00005445535453335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
141-Bug>

```

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

The offset address of -65000000 was added to the addresses of the records in TEST.MX and caused the program to be loaded to memory starting at \$40000. The text "COPY TEST.MX,#" is a VERSAdos command line that caused the file to be copied by VERSAdos to the host system port which is connected with the MVME141 host port.

```

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

```

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

## 3.26 LAN STATION ADDRESS DISPLAY/SET

*not present in 141*

LSAD

## LSAD

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

Every MVME141 is assigned an Ethernet station address. The address is \$08003E2XXXXX where XXXXX is the unique number assigned to the module (i.e., every MVME141 has a different value for XXXXX).

Each Ethernet station address is displayed on a label attached to the back of the MVME141 front panel. In addition, the XXXXX portion of the Ethernet station address is stored in BBRAM, location \$FFFE0778 as \$2XXXXX.

If Motorola networking software is running on an MVME141, it uses the 2XXXXX value from BBRAM to complete the Ethernet station address (\$08003E2XXXXX). The user must assure that the value of 2XXXXX is maintained in BBRAM. If the value of 2XXXXX is lost in BBRAM, the user should use number on the front panel label to restore it. Note that MVME141bug includes the "LSAD" command for examining and updating the BBRAM XXXXX value.

Example: (display Ethernet station address)

```
141-Bug> lsad
```

```
LAN Station Address = $08003E200000
```

```
To set the Station Address:
```

```
Enter the code located on the back of the front panel $1
```

```
LAN Station Address = $08003E200001
```

```
141-Bug>
```

## 3.27 MACRO DEFINE/DISPLAY/DELETE

MA  
NOMAMA [name]  
NOMA [name]

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

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

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

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

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

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

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

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

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

MA  
NOMA

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

Examples:

```
141-Bug> MA ABC
M=MD 3000
M=GO \0
M= (CR)
141-Bug>
```

Define macro ABC

```
141-Bug> MA DIS
M=MD \0:17;DI
M= (CR)
141-Bug>
```

Define macro DIS

```
141-Bug> MA
MACRO ABC
010 MD 3000
020 GO \0
MACRO DIS
010 MD \0:17;DI
141-Bug>
```

List macro definitions

```
141-Bug> MA ABC
MACRO ABC
010 MD 3000
020 GO \0
141-Bug>
```

List definitions of macro ABC

```
141-Bug> NOMA DIS
141-Bug>
```

Delete macro DIS

```
141-Bug> MA ASM
M=MM \0;DI
M= (CR)
141-Bug>
```

Define macro ASM

THE 141Bug DEBUGGER COMMAND SET

MA  
NOMA

141-Bug> MA  
MACRO ABC  
010 MD 3000  
020 GO \0  
MACRO ASM  
010 MD \0;DI  
141-Bug>

List all macros

141-Bug> NOMA  
141-Bug>

Delete all macros

141-Bug> MA  
NO MACROS DEFINED  
141-Bug>

List all macros

## 3.28 MACRO EDIT

MAE

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

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

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

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

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

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

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

Examples:

```
141-Bug> MA ABC
MACRO ABC
010 MD 3000
020 GO \0
141-Bug>
```

List definitions of macro ABC

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

Add a line to macro ABC

This line was inserted

MAE

141-Bug> MAE ABC 10 MD 10+RO  
MACRO ABC  
010 MD 10+RO  
020 RD  
030 GO \0  
141-Bug>

Replace line 10

This line was overwritten

141-Bug> MAE ABC 30  
MACRO ABC  
010 MD 10+RO  
020 RD  
141-Bug>

Delete line 30

3

**3.29 ENABLE/DISABLE MACRO EXPANSION LISTING**

**MAL**  
**NOMAL**

**MAL**  
**NOMAL**

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

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

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

## 3.30 SAVE/LOAD MACROS

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

*controller LUN* is the logical unit number of the controller to which the above device is attached. Initially defaults to LUN 0.

*device LUN* is the logical unit number of the device to save/load macros to/from. Initially defaults to LUN 0.

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

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

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

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

**NOTE**

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

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

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

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

141-Bug&gt; MAR 0,2,3

Load macros from block 3

141-Bug&gt;

THE 141Bug DEBUGGER COMMAND SET

MAW  
MAR

141-Bug> MA  
MACRO ABC  
010 MD 3000  
020 GO \0  
141-Bug>

List macros

141-Bug> MA ASM  
M=MM \0;DI  
M=(CR)  
141-Bug>

Define macro ASM

141-Bug> MA  
MACRO ABC  
010 MD 3000  
020 GO \0  
MACRO ASM  
010 MD \0;DI  
141-Bug>

List all macros

141-Bug> MAW ,,8  
WRITING TO BLOCK \$8 ON CONTROLLER \$0, DEVICE \$2  
OK to proceed (y/n)? Y  
141-Bug>

Save macros to block 8, previous device  
Carriage return not needed

## 3.31 MEMORY DISPLAY

MD

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

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

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

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

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

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

Example 1:

```
141-Bug> md 12000
00012000 2800 1942 2900 1942 2800 1842 2900 2846      (..B)..B(..B).(F
141-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

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

MD

## Example 3:

```

141-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
141-Bug>

```

## Example 4:

```

141-Bug> md 5000;d
00005000 0_3F6_44C1D0F047FC2= 2.477700000000000002 E-0003
00005008 0_423_DAEFF04800000= 1.274900000000000000 E+0011
00005010 0_000_0000000000000= 0.000000000000000000 E+0000
00005018 0_403_0000000000000= 1.600000000000000000 E+0001
00005020 1_3FF_0000000000000=-1.000000000000000000 E+0000
00005028 0_000_00000FFFFFFF= 2.1219957904712067 E+0314
00005030 0_44D_FDE9F10A8D361= 6.020000000000000000 E+0023
00005038 0_3C0_79CA10C924223= 1.599999999999999999 E+0019
141-Bug>

```

## 3.32 MENU

## MENU

## MENU

The MENU command works only if the 141Bug is in the "system" mode (refer to the ENV command). When invoked in the "system" mode, it provides a way to exit 141Bug and return to the menu.

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

141-Bug> MENU

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

Enter Menu #:

When the 141Bug is in "system" mode, a user can toggle back and forth between the menu and Bug by typing a 3 in response to the Enter Menu #: prompt when the menu is displayed. Entering the Bug and then typing MENU in response to the 141-Bug (or 141-Diag prompt) to return to the system menu.

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

## 3.33 MEMORY MODIFY

MM

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

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

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

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

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

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

- V or v The next successive memory location is opened. (This is the default. It is in effect whenever MM is invoked and remains in effect until changed by entering one of the other special characters.)
- ^ MM backs up and opens the previous memory location.
- = MM re-opens the same memory location (this is useful for examining I/O registers or memory locations that are changing over time).
- . Terminates MM command. Control returns to 141Bug.

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

Example 1:

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

MM

Example 2:

```

141-Bug> mm 10001;1a                Longword access to location 10001
00010001 CD432187? (CR)             (Alternate location accesses).
00010009 00068010? 68010+10=       Modify and reopen location.
00010009 00068020? (CR)
00010009 00068020? .               Exit MM.

```

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

- Enter (CR). This closes the present location and continues with disassembly of next instruction.
- Enter a new source instruction followed by (CR). This invokes the assembler, which assembles the instruction and generates a "listing file" of one instruction.
- Enter .(CR). This closes the present location and exits the MM command.

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

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

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

The examples below were made in the hardcopy mode.

Example 3: Assemble a new source line.

```

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

```

Example 4: New source line with error.

```

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

```

MM

Example 5: Step to next location and exit MM.

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

Example 6:

```
141-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_84782000000000000? .
141-Bug>
```

3

## 3.34 MEMORY SET

MS

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

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

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

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

Example: Assume that memory is initially cleared:

```
141-Bug> ms 25000 0123456789abcDEF 'This is ''141Bug'' 23456
141-Bug> md 25000:20;b
00025000 0123 4567 89AB CDEF   5468 6973 2069 7320   .#Eg.+MoThis is
00025010 2731 3437 4275 6727   2345 6000 0000 0000   '141Bug'#E'.....
141-Bug>
```

## 3.35 SET MEMORY ADDRESS FROM VMEbus

*not present in 141*

OBA

## OBA

The OBA (Off-Board Address) command allows the user to set the base address of the MVME141 onboard RAM, as seen from the VMEbus. (Refer to Chapter 1.) Therefore, the user should enter the hex number corresponding to the actual base address, so that the offboard external devices on the VMEbus will know where it is. The default (factory-delivered) condition is with the offboard address set to \$0.

## Example:

141-Bug&gt; oba

RAM address from VMEbus = \$0 400000

Change \$0 to \$400000.

141-Bug&gt; oba

RAM address from VMEbus = \$400000 (CR)

141-Bug

## 3.36 OFFSET REGISTERS DISPLAY/MODIFY

OF

OF [Rn[;A] ]

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

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

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

Command usage:

- OF - To display all offset registers. An asterisk indicates which register is the automatic register.
- OF Rn - To display/modify Rn. The user can scroll through the registers in a way similar to that used by the MM command.
- OF Rn;A - To display/modify Rn and set it as the automatic register. The automatic register is one that is automatically added to each absolute address argument of every command except if an offset register is explicitly added. An asterisk indicates which register is the automatic register.

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

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

Offset register rules:

- At power-up and cold-start reset, R7 is the automatic register.
- At power-up and cold-start reset, all offset registers have both base and top addresses preset to 0. This effectively disables them.
- R7 always has both base and top addresses set to 0; cannot be changed.

OF

- d. Any offset register can be set as the automatic register.
- e. The automatic register is always added to every absolute address argument of every 141Bug command where there is not an offset register explicitly called out.
- f. There is always an automatic register. A convenient way to disable the effect of the automatic register is by setting R7 as the automatic register. Note that this is the default condition.

Examples:

Display offset registers.

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

Modify some offset registers.

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

Look at location \$20000.

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

Set R0 as the automatic register.

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

To look at location \$20000.

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

To look at location 0, override the automatic offset.

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

## 3.37 PRINTER ATTACH/DETACH

PA  
NOPAPA [n]  
NOPA [n]

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

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

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

```
141-Bug> PF4
Logical unit $04 unassigned
Name of board? VME141
Name of port? PTR
Port base address = $FFFE2800? (CR)
Auto Line Feed protocol [Y,N] = N? Y.
OK to proceed (y/n)? Y
141-Bug>
```

For further details, refer to the PF command.

## Examples:

CONSOLE DISPLAY:	PRINTER OUTPUT:
141-Bug> PA4	(printer now attached)
(attaching port 4)	141-Bug>HE NOPA
141-Bug> HE NOPA	NOPA Printer detach
NOPA Printer detach	141-Bug>NOPA
141-Bug> NOPA	(printer now detached)
(detach all attached printers)	
141-Bug> NOPA	
No printer attached	
141-Bug>	

## 3.38 PORT FORMAT/DETACH

PF  
NOPFPF[n]  
NOPFn

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

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

## 3.38.1 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:

```
141-Bug> pf
Current port assignments: (Port #: Board name, Port name)
[00: VME141- "1"] [01: VME141- "2"]
141-Bug>
```

## 3.38.2 Configuring a Port

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

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

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

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

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

PF  
NOPF

- v Either uppercase or lowercase "v" causes Port Format to resume or prompting in the original order (i.e., Baud Rate, then Parity Type, V ...).
- = Entering an equal sign by itself or when following a value causes PF to issue the same prompt again. This is supported to be consistent with the operation of other debugger commands. To resume prompting in either normal or reverse order, enter the letter "v" or a caret "^", respectively.
  - . Entering a period by itself or following a value causes Port Format to exit from the interactive mode and issue the "OK to proceed (y/n)?".
- (CR) Pressing return without entering a value preserves the current value and causes the next prompt to be displayed.

Example:

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

### 3.38.3 Parameters Configurable by Port Format

Port base address:

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

Baud rate:

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

Parity type:

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

PF  
NOPF**Character width:**

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

**Number of stop bits:**

Only 1 and 2 stop bits are supported.

**Synchronization type:**

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

**Synchronization character values:**

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

**Automatic software handshake:**

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

**Software handshake character values:**

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

**3.38.4 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 VME141, VME050, etc.). Pressing the RETURN key on the console at this point causes PF to list the currently supported modules and ports. Once the name of the module (board) has been entered, a prompt is issued for the name of the port. After the port name has been entered, Port Format attempts to supply a default configuration for the new port.

PF  
NOPF

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

If the configuration of the new port is not fixed, then the interactive configuration mode is entered. Refer to paragraph 3.30.2 above regarding configuring assigned ports. If the new port does have a fixed configuration, then Port Format issues the "OK to proceed (y/n)?" prompt immediately.

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

Example: Assigning port 7 to the MVME050 printer port.

```
141-Bug> PF 7
Logical unit $07 unassigned
Name of board? (CR)      (cause PF to list supported modules (boards), ports)
Boards and ports supported:
VME141: 1,2,3,4,PTR
VME050: 1,2,PTR2
Name of board? VME050      (uppercase or lowercase accepted)
Name of port? PTR2
Port base address = $FFFF1080? (CR)
Auto Line Feed protocol [Y,N] = N? .
(interactive mode not entered because hardware has fixed configuration)
OK to proceed (y/n)? Y
141-Bug>
```

### 3.38.5 NOPF Port Detach

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

## 3.39 PUT RTC INTO POWER SAVE MODE FOR STORAGE

PS

PS

The PS command is used to turn off the oscillator in the RTC chip, MK48T02. The MVME141 module is shipped with the RTC oscillator stopped to minimize current drain from the on-chip battery. Normal cold-start of the MVME141 with the 141Bug EPROMs installed gives the RTC a "kick start" to begin oscillation. To disable the RTC, the user must enter "PS".

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

Example:

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

## 3.40 ROMBOOT ENABLE/DISABLE

RB  
NORBRB  
NORB

The RB command enables the search for and booting from a routine nominally encoded in on-board ROMs/PROMs/EPROMs/EEPROMs on the MVME141. However, the routine can be in other memory locations, as detailed in the RB command options given below. Refer also to the ROMboot function description and example in paragraph 1.5.

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

The default condition is with the ROMboot function disabled.

Examples:

141-Bug> RB

Boot at power-up only [Y,N] ? Y (CR)

Enable search of VMEbus [Y,N] ? N (CR)

Boot direct address = \$FF800000 (CR)

141-Bug> NORB

ROM boot disabled

141-Bug>

If the user types N, then boot is attempted at any board reset.

If the user types Y, the search for "BOOT", etc. starts at the end of onboard memory, in 8Kb increments.

This default address is the start of the 141Bug EPROMs, so the search here is fast.

This disables the ROMboot function but does not change any options chosen under RB.

## 3.41 REGISTER DISPLAY

RD

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

The RD command is used to display the target state, that is, the register state associated with the target program (refer to the GO command). The instruction pointed to by the target PC is disassembled and displayed also. Internally, a register mask specifies which registers are displayed when RD <CR> is executed. At reset time, this mask is set to display the MPU registers only. This register mask can be changed with the RD command. The optional arguments allow the user 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.

The arguments are as follows:

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

MPU	Microprocessor Unit
MMU	Memory Management Unit
FPC	Floating Point Coprocessor

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

- a. The qualifier is applied to the next register range only.
- b. If no qualifier is specified, a + qualifier is assumed.
- c. All device names should appear before any register names.
- d. The command line arguments are parsed from left to right, with each field being processed after parsing, thus, the sequence in which qualifiers and registers are organized has an impact on the resultant register mask.

RD

- e. When specifying a register range, *reg1* and *reg2* do not have to be of the same class.
- f. The register mask used by RD is also used by all exception handler routines, including the trace and breakpoint exception handlers.

The MPU registers in ordering sequence are:

<u>NUMBER AND TYPE OF REGISTERS</u>	<u>MNEMONICS</u>
10 System Registers (PC,SR,USP,MSP,ISP,VBR,SFC,DFC,CACR,CAAR)	
8 Data Registers (D0-D7)	
8 Address Registers (A0-A7)	

(Total: 26 Registers. Note that A7 represents the active stack pointer, which leaves 25 different registers.)

The MMU registers in ordering sequence are:

<u>NUMBER AND TYPE OF REGISTERS</u>	<u>MNEMONICS</u>
5 Address Translation/Control	(CRP,SRP,TC,TT0,TT1)
1 Status	(MMUSR)

The FPC registers in ordering sequence are:

<u>NUMBER AND TYPE OF REGISTERS</u>	<u>MNEMONICS</u>
3 System Registers	(FPCR,FPSR,FPIAR)
8 Data Registers	(FPO-FP7)

Example 1: Default display - MPU registers only.

```
141-Bug> rd
PC =00004000 SR =2700=TR:OFF S. 7 . . . . . VBR =00000000
USP =0000F830 MSP =00005C18 ISP*=00006000 SFC =0=F0
CACR=0=D: . . . . I: . . . . CAAR=00000000 DFC =0=F0
D0 =00000000 D1 =00000000 D2 =00000000 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
141-Bug>
```

#### NOTE

An asterisk following a stack pointer name indicates that it is the active stack pointer.

The status register includes a mnemonic portion to help in reading it:

TRACE BITS			
T1	TO	MNEMONIC	DESCRIPTION
0	0	TR:OFF	Trace off
0	1	TR:CHG	Trace on change of flow
1	0	TR:ALL	Trace all states
1	1	TR:INV	Invalid mode

S, M bits: The bit name appears (S,M) if the respective bit is set, otherwise a "." indicates that it is cleared.

Interrupt Mask: A number (0 to 7) indicates current processor priority level.

Condition Codes: The bit name appears (X,N,Z,V,C) if the respective bit is set, otherwise a "." indicates that it is cleared.

The source and destination function code registers (SFC, DFC) include a two character mnemonic:

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

The CACR register shows mnemonics for two bits: Enable and Freeze. The bit name (E, F) appears if the respective bit is set, otherwise a "." indicates that it is cleared.

Example 2: To display only the MMU registers.

```
141-Bug> RD =MMU
CRP =00000001_00000000      SRP =00000001_00000000
TC  =00000000_TTO =00000000  TT1 =00000000
MMUSR=0000=....._0        PSR =0000-....._0
00004000 4AFC             ILLEGAL
141-Bug>
```

The MMUSR register above includes a mnemonic portion, the bits are:

B	Bus error	bit 15
L	Limit Violation	bit 14
S	Supervisor only	bit 13
W	Write protected	bit 11
I	Invalid	bit 10
M	Modified	bit 9
T	Transparent Access	bit 6
N	Number of Levels (3 bits)	bits 2-0

Example 3: To display only the FPC registers.

```
141-Bug> RD =fpc
FPCR =00000000  FPSR =00000000-(CC=.... )  FPIAR=00000000
FP0 =0_7FFF_FFFFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFFFF_E-OFFF
FP1 =0_7FFF_FFFFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFFFF_E-OFFF
FP2 =0_7FFF_FFFFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFFFF_E-OFFF
FP3 =0_7FFF_FFFFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFFFF_E-OFFF
FP4 =0_7FFF_FFFFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFFFF_E-OFFF
FP5 =0_7FFF_FFFFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFFFF_E-OFFF
FP6 =0_7FFF_FFFFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFFFF_E-OFFF
FP7 =0_7FFF_FFFFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFFFF_E-OFFF
00004000 4AFC          ILLEGAL
141-Bug>
```

The floating point data registers are always displayed in extended precision and in scientific notation format. The floating point status register display includes a mnemonic portion for the condition codes. The bit name appears (N, X, I, NAN) if the respective bit is set, otherwise, a "." indicates that it is cleared.

Example 4: To remove D3 through D5 and A2, and add FPSR and FP0, starting with the previous display.

```
141-Bug> RD MPU/-FPC/-D3-D5/-A2/FP0/FPSR
PC =00004000  SR =2700=TR:OFF S. 7..... VBR =00000000
USP =0000F830  MSP =00005C18  ISP*=00006000  SFC =0=F0
CACR =0=D:.... I:...  CAAR=00000000  DFC =0=F0
D0 =00000000  D1 =00000000  D2 =00000000  D6 =00000000
D7 =00000000  A0 =00000000  A1 =00000000  A3 =00000000
A4 =00000000  A5 =00000000  A6 =00000000  A7 =00006000
FPSR =00000000-(CC=.... )
FP0 =0_7FFF_FFFFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFFFF_E-OFFF
00004000 4AFC          ILLEGAL
141-Bug>
```

RD

Example 5: To set the display to D6 and A3 only.

```
141-Bug> RD =D6/A3
D6   =00000000 A3   =00000000
00004000 4AFC                ILLEGAL
141-Bug>
```

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

Example 6: To restore all the MPU registers.

```
141-Bug> rd +mpu
PC   =00004000 SR   =2700=TR:OFF_S_7_..... VBR =00000000
USP  =0000F830 MSP  =00005C18 ISP*=00006000 SFC =0=F0
CACR=0=D:...._I:... CAAR=00000000 DFC =0=F0
D0   =00000000 D1   =00000000 D2   =00000000 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
141-Bug>
```

Note that an equivalent command would have been RD PC-A7.

3.42 REMOTE

REMOTE

REMOTE

The REMOTE command duplicates the remote operation modem functions available from the "system" mode menu command, entry number 4. It is only accessible when the 141Bug is in "system" mode (refer to MENU command in Appendix A for details on remote operation).

3

## 3.43 COLD/WARM RESET

## RESET

## RESET

The RESET command is used to issue a local SCSI bus reset and also allows the user to specify the level of reset operation that will be in effect when a RESET exception is detected by the processor. A reset exception can be generated by pressing the RESET switch on the MVME141 front panel, or by executing a software reset.

Two RESET levels are available:

- COLD - This is the standard level of operation, and is the one defaulted to on power-up. In this mode, all the static variables are initialized every time a reset is done.
- WARM - In this mode, all the static variables are preserved when a reset exception occurs. This is convenient for keeping breakpoints, offset register values, the target register state, and any other static variables in the system.

NOTE

If the MVME141 is the system controller, pressing the RESET switch resets all the modules in the system, including disk controllers like the MVME320 or MVME360. This may cause the disk controller configuration to be out of phase with respect to the disk configuration tables in memory.

Example 1:

141-Bug> RESET

Cold/Warm Reset [C,W] = C? W

Execute Soft Reset [Y,N] ? Y

Arm for warm start the next time  
a reset is performed. Do a  
software reset now, actually  
forcing a warm start.

Copyright Motorola Inc. 1988, All Rights Reserved

VME141 Monitor/Debugger Release 1.0 - x/xx/xx

WARM Start

141-Bug>

## 3.44 REGISTER MODIFY

RM

RM *reg*

RM command allows the user to display and change the target registers. It works in essentially the same way as the MM command, and the same special characters are used to control the display/change session (refer to the MM command).

**NOTE**

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

Example 1:

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

Example 2:

```
141-Bug> rm sfc
SFC =7=CS ? 1=             Modify register and reopen.
SFC =1=UD ? .              Exit.
141-Bug>
```

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

Example 3:

```
141-Bug> rm crp
CRP =00000001_00000000    ? (CR)
SRP =00000001_00000000    ? (CR)
TC =00000000 ? 87654321
TTO =00000000 ? 12345678
TT1 =00000000 ? 87654321
MMUSR=0000=. ...._0? .
```

RM

```

141-Bug> rd +mmu
PC =00004000 SR =2700=TR:OFF S. 7 ..... VBR =00000000
USP =00005830 MSP =00005C18 ISP*=00006000 SFC =0=FO
CACR =0=D:.... I:... CAAR=00000000 DFC =0=FO
D0 =00000000 D1 =00000000 D2 =00000000 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00006000
CRP =00000001 00000000 SRP =00000001 00000000
TC =87654321 TTO =12345678 TT1 =87654321
MMUSR=0000=....._0
00004000 4AFC ILLEGAL
141-Bug>

```

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

Example 4:

```

141-Bug> rm fpsr
FPSR =00000000-(CC=.... ) ? F000000
FPIAR=00000000 ? <CR>
FP0 =0 7FFF FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFFF? 0 1234 5
FP1 =0 7FFF FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFFF? 1.25E3
FP2 =0 7FFF FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFFF? 1 7F 3FF
FP3 =0 7FFF FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFFF? 1100_9261_3
FP4 =0 7FFF FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFFF? &564
FP5 =0 7FFF FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFFF? 0 5FF FOAB
FP6 =0 7FFF FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFFF? 3.1415
FP7 =0 7FFF FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFFF? -2.74638369E-36.
141-Bug>

```

RM

```

141-Bug> rd +fpc
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
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
141-Bug>

```

## 3.45 REGISTER SET

RS

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

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

NOTE

*reg* is the mnemonic for the particular register.

Example 1:

```
141-Bug> RS D5 123455678           Change MPU register.
D5  =12345678
141-Bug>
```

Example 2:

```
141-Bug> rs tt0 87654321          Change MMU register.
TT0 =87654321
141-Bug>
```

Example 3:

```
141-Bug> rs FPO 0 1234 5           Change FPC register.
FPO =0_1234_50000000000000000000= 6.6258385370745493_E-3530
141-Bug>
```

## 3.46 SWITCH DIRECTORIES

SD

## SD

The SD command is used to change from the debugger directory to the diagnostic directory or from the diagnostic directory to the debugger directory.

The commands in the current directory (the directory that the user is in at the particular time) may be listed using the Help (HE) command.

The way the directories are structured, the debugger commands are available from either directory but the diagnostic commands are only available from the diagnostic directory.

Example 1:

```
141-Bug> SD
141-Diag>          (The user has changed from the debugger )
                   (directory to the diagnostic directory, )
                   (as can be seen by the "141-Diag"> )
                   (prompt. )
```

Example 2:

```
141-Diag> SD
141-Bug>          (The user is now back in the debugger )
                   (directory. )
```

## 3.47 SET TIME AND DATE

SET

## SET

The SET command is interactive and begins with the user entering SET followed by a carriage return. At this time, a prompt asking for MM/DD/YY is displayed. The user may change the displayed date by typing a new date followed by (CR), or may simply enter (CR), which leaves the displayed date unchanged. When the correct date matches the data entered, the user should press the carriage return to establish the current value in the time-of-day clock.

Note that an incorrect entry may be corrected by backspacing or deleting the entire line as long as the carriage return has not been entered.

After the initial prompt and entry, another prompt is presented asking for a calibration value. This value slows down (- value) or speeds up (+ value) the RTC in the MK48T02 chip. Refer to the MK48T02 data sheet (as mentioned in Chapter 1, herein) for details.

Next, a prompt is presented asking for HH:MM:SS. The user may change the displayed time by typing a new time followed by (CR), or may simply enter (CR), which leaves the displayed time unchanged.

To display the current date and time of day, refer to the TIME command.

Example: To SET a date and time of May 11, 1988 2:05:32 PM the command is as follows:

```
141-Bug> SET
```

```
Weekday xx/xx/xx xx:xx:xx
```

```
Present calibration = -0
```

```
Enter date as MM/DD/YY
```

```
05/11/88
```

```
Enter Calibration value +/- (0 to 31)
```

```
Enter time as HH:MM:SS (24 hour clock)
```

```
14:05:32
```

```
141-Bug>
```

This will start a stopped clock.

(Refer to the PS command.)

This can speed up (+) or slow down (-) the RTC oscillator.

## 3.48 TRACE

T

T [*count*]

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 141Bug.

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 141Bug.

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

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

```
141-Bug> MD 10000;DI
00010000 2200          MOVE.L DO,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
141-Bug>
```

Initialize PC and D0:

```
141-Bug> RM PC
PC =00008000 ? 10000.
141-Bug> RM D0
D0 =00000000 ? 8F41C.
```

Display target registers and trace one instruction:

```
141-Bug> RD
PC =00010000 SR =2700=TR:OFF S. 7 .....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =0008F41C D1 =00000000 D2 =00000000 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010000 2200          MOVE.L DO,D1
```

T

```

141-Bug> T
PC =00010002 SR =2700=TR:OFF S. 7 .....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =0008F41C D1 =0008F41C D2 =00000000 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010002 4282 CLR.L D2
141-Bug>

```

Trace next instruction:

```

141-Bug> (CR)
PC =00010004 SR =2704=TR:OFF S. 7 ..Z..
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =0008F41C D1 =0008F41C D2 =00000000 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010004 D401 ADD.B D1,D2
141-Bug>

```

Trace the next two instructions:

```

141-Bug> T 2
PC =00010006 SR =2700=TR:OFF S. 7 .....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =0008F41C D1 =0008F41C D2 =0000001C D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010006 E289 LSR.L #1,D1
PC =00010008 SR =2700=TR:OFF S. 7 .....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =0008F41C D1 =00047A0E D2 =0000001C D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010008 66FA BNE.B $10004
141-Bug>

```

## 3.49 TERMINAL ATTACH

TA

TA [*port*]

TA command allows the user to assign any serial port to be the console. The port specified must already be assigned (refer to the Port Format (PF) command).

Example 1: Selecting port 2 (logical unit #02) as console.

141-Bug> TA 2 (No prompt appears unless port 2 was already the console.)

Example 2: Restoring console to port selected at power-up.

141-Bug> TA (Prompt now appears at terminal connected to port 0.)

## 3.50 TRACE ON CHANGE OF CONTROL FLOW

TC

TC [count]

TC command starts execution at the address in the target PC and begins tracing upon the detection of an instruction that causes a change of control flow, such as JSR, BSR, RTS, etc. This means that execution is in real-time until a change of flow instruction is encountered. The optional count field (which defaults to 1 if none entered) specifies the number of change of flow instructions to be traced before returning control to 141Bug.

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 141Bug.

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

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

```
141-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
141-Bug>
```

Initialize PC and D0:

```
141-Bug> RM PC
PC =00008000 ? 10000.
141-Bug> RM D0
D0 =00000000 ? 8F41C.
```

Trace on change of flow:

```
141-Bug> TC
00010008 66FA          BNE.B  $10004          ( Note that this )
PC =00010004 SR =2700=TR:OFF S. 7 ..... ( display also )
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000 ( shows the )
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000 ( change of flow )
D0 =0008F41C D1 =00047A0E D2 =0000001C D3 =00000000 ( instruction. )
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010004 D401          ADD.B  D1,D2
141-Bug>
```

3.51 DISPLAY TIME AND DATE

TIME

TIME

The TIME command presents the date and time in ASCII characters to the console.

To initialize the time-of-day clock, refer to the SET command.

Example: A date and time of Wednesday, May 11, 1988 2:05:32 would be displayed as:

```
141-Bug> TIME
Wednesday 5/11/88 14:05:32
141-Bug>
```

## 3.52 TRANSPARENT MODE

TM

TM [*n*] [*escape*]

TM command essentially connects the console serial port and the host port together, allowing the user to communicate with a host computer. A message displayed by TM shows the current escape character, i.e., the character used to exit the transparent mode. The two ports remain "connected" until the escape character is received by the console port. The escape character is not transmitted to the host, and at power up or reset it is initialized to \$01=^A.

The optional port number *n* allows the user to specify which port is the "host" port. If omitted, port 1 is assumed.

The ports do not have to be at the same baud rate, but the terminal port baud rate should be equal to or greater than the host port baud rate for reliable operation. To change the baud rate use the Port Format (PF) command.

The optional escape argument allows the user to specify the character to be used as the exit character. This can be entered in three different formats:

ASCII code	:	\$03	Set escape character to ^C
control character:		^C	Set escape character to ^C
ASCII character	:	'c	Set escape character to c

If the port number is omitted and the escape argument is entered as a numeric value, precede the escape argument with a comma to distinguish it from a port number.

## Example 1:

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

^A                    Exit transparent mode.
```

## Example 2:

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

^G                    Exit transparent mode.
141-Bug>
```

## 3.53 TRACE TO TEMPORARY BREAKPOINT

TT

TT *addr*

TT command sets a temporary breakpoint at the specified address and traces until a breakpoint with 0 count is encountered. The temporary breakpoint is then removed (TT is analogous to the GT command) and control is returned to 141Bug. 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 141Bug.

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

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

```
141-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
141-Bug>
```

Initialize PC and D0:

```
141-Bug>RM PC
PC =00008000 ? 10000.
141-Bug>RM D0
D0 =00000000 ? 8F41C.
```

Trace to temporary breakpoint:

```
141-Bug>TT 10006
PC =00010002 SR =2700=TR:OFF S. 7 .....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =0008F41C D1 =0008F41C D2 =00000000 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010002 4282          CLR.L  D2
```

TT

```

PC =00010004 SR =2704=TR:OFF S. 7...Z..
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =0008F41C D1 =0008F41C D2 =00000000 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010004 D401 ADD.B D1,D2
At Breakpoint
PC =00010006 SR =2700=TR:OFF S. 7.....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX DFC =0=XX CACR=0=.. CAAR=00000000
D0 =0008F41C D1 =0008F41C D2 =0000001C D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00004000
00010006 E289 LSR.L #1,D1
141-Bug>

```

3

## 3.54 VERIFY S-RECORDS AGAINST MEMORY

VE

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

The VE command is identical to the LO command with the exception that data is not stored to memory but merely compared to the contents of memory.

This command accepts serial data from a host system in the form of a file of Motorola S-records and compares it to data already in the MVME141 memory. If the data does not compare, then the user is alerted via information sent to the terminal screen.

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

The optional *addr* field allows the user to enter an offset address which is to be added to the address contained in the address field of each record. This causes the records to be compared to memory at different locations than would normally occur. The contents of the automatic offset register are not added to the S-record addresses. (Appendix C has information on S-records.) If the address is in the range \$0 to \$1F and the port number is omitted, precede the address with a comma to distinguish it from a port number.

The optional *text* field, entered after the equal sign (=), is sent to the host before 141Bug begins to look for S-records at the host port. This allows the user to send a command to the host device to initiate the download. This text should NOT be delimited by any kind of quote marks. Text is understood to begin immediately following the equals sign and terminate with the carriage return. If the host is operating full duplex, the string is also echoed back to the host port by the host and appears on the user's terminal screen.

In order to accommodate host systems that echo all received characters, the above-mentioned text string is sent to the host one character at a time and characters received from the host are read one at a time. After the entire command has been sent to the host, VE keeps looking for an LF character from the host, signifying the end of the echoed command. No data records are processed until this LF is received. If the host system does not echo characters, VE still keeps looking for an LF character before data records are processed. For this reason, it is required in situations where the host system does not echo characters, that the first record transferred by the host system be a header record. The header record is not used, but the LF after the header record serves to break VE out of the loop so that data records are processed.

The other options have the following effects:

## -C Ignore checksum.

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

VE

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

During a verify operation, data from an S-record is compared to memory beginning with the address contained in the S-record address field (plus the offset address, if it was specified). If the verification fails, then the non-comparing record is set aside until the verify is complete and then it is printed out to the screen. If three non-comparing records are encountered in the course of a verify operation, then the command is aborted.

If a non-hex character is encountered within the data field of a data record, then the part of the record which had been received up to that time is printed to the screen and the 141Bug error handler is invoked to point to the faulty character.

If the embedded checksum of a record does not agree with the checksum calculated by 141Bug AND if the checksum comparison has not been disabled via the "-C" option, then an error condition exists. A message is output stating the address of the record (as obtained from the address field of the record), the calculated checksum, and the checksum read with the record. A copy of the record is also output. This is a fatal error and causes the command to abort.

Examples:

This short program was developed on a host system.

```

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

***** TOTAL ERRORS      0--
***** TOTAL WARNINGS   0--

```

Then the program was converted into an S-record file named TEST.MX that looks like this:

```

S00F0000544553545335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091

```

VE

This file was downloaded into memory at address \$40000. The program may be examined in memory using the Memory Display (MD) command.

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

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

```
141-Bug> VE -65000000;X=COPY TEST.MX,#
S00F00005445535453335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
Verify passes.
141-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.

```
141-Bug> M 40002
00040002 D088? D089.

141-Bug> VE -65000000;X=COPY TEST.MX,#
S00F00005445535453335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
```

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

```
S30D65040000-----88-----B3
```

```
141-Bug>
```

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

## CHAPTER 4 - USING THE ONE-LINE ASSEMBLER/DISASSEMBLER

## 4.1 INTRODUCTION

Included as part of the 141Bug 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 141Bug 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.

## 4.1.1 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

## 4.1.1.1 Machine-Instruction Operation Codes

That part of the assembly language that provides the mnemonic machine-instruction operation codes for the MC68030/MC68882 machine instructions is described in the MC68030 and MC68882 User's Manuals, MC68030UM and MC68881UM. Refer to these manuals for any question concerning operation codes.

## 4.1.1.2 Directives

Normally, assembly language can contain mnemonic directives which specify auxiliary actions to be performed by the assembler.

The 141Bug assembler recognizes only two directives called define constant (DC.W) and SYSCALL. These two directives are used to define data within the program and to make calls to 141Bug utilities. Refer to paragraphs 4.2.3 and 4.2.4, respectively.

#### 4.1.2 Comparison with MC68030 Resident Structured Assembler

There are several major differences between the 141Bug 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 141Bug assembler processes each line of a program as an individual unit. Due mainly to this basic functional difference, the capabilities of the 141Bug assembler are more restricted:

- a. Label and line numbers are not used. Labels are used to reference other lines and locations in a program. The one-line assembler has no knowledge of other lines and, therefore, cannot make the required association between a label and the label definition located on a separate line.
- b. Source lines are not saved. In order to read back a program after it has been entered, the machine code is disassembled and then displayed as mnemonic and operands.
- c. Only two directives (DC.W and SYSCALL) are accepted.
- d. No macro operation capability is included.
- e. No conditional assembly is used.
- f. Several symbols recognized by the resident assembler are not included in the 141Bug assembler character set. These symbols include > and <. Three other symbols have multiple meaning to the resident assembler, depending on the context (refer to paragraph 4.2.2). These are:
  - Asterisk (\*) -- Multiply or current PC.
  - Slash (/) -- Divide or delimiter in a register list.
  - Ampersand (&) -- AND or decimal number.

Although functional differences exist between the two assemblers, the one-line assembler is a true subset of the resident assembler. The format and syntax used with the 141Bug assembler are acceptable to the resident assembler except as described above.

## 4.2 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.

### 4.2.1 Source Line Format

Each source statement is a combination of operation and, as required, operand fields. Line numbers, labels, and comments are not used.

#### 4.2.1.1 Operation Field

Because there is no label field, the operation field may begin in the first available column. It may also follow one or more spaces. Entries can consist of one of three categories:

- a. Operation codes which correspond to the MC68030/MC68882 instruction set.
- b. Define Constant directive: DC.W is recognized to define a constant in a word location.
- c. System Call directive: SYSCALL is used to call 141Bug system utilities.

The size of the data field affected by an instruction is determined by the data size codes. Some instructions and directives can operate on more than one data size. For these operations, the data size code must be specified or a default size applicable to that instruction will be assumed. The size code need not be specified if only one data size is permitted by the operation. The data size code is specified by a period (.), appended to the operation field, and followed by B, W, or L, where:

- B = Byte (8-bit data).
- W = Word (the usual default size; 16-bit data).
- L = Longword (32-bit data).

The data size code is not permitted, however, when the instruction or directive does not have a data size attribute.

#### Examples (legal):

- LEA (A0),A1 Longword size is assumed (.B, .W not allowed); this instruction loads effective address of the first operand into A1.
- ADD.B (A0),D0 This instruction adds the byte whose address is (A0) to the lowest order byte in D0.
- ADD D1,D2 This instruction adds the low order word of D1 to the low order word of D2. (W is the default size code.)
- ADD.L A3,D3 This instruction adds the entire 32-bit (longword) contents of A3 to D3.

#### Example (illegal):

- SUBA.B #5,A1 Illegal size specification (.B not allowed on SUBA). This instruction would have subtracted the value 5 from the low order byte of A1; byte operations on address registers are not allowed.

#### 4.2.1.2 Operand Field

If present, the operand field follows the operation field and is separated from the operation field by at least one space. When two or more operand subfields appear within a statement, they must be separated by a comma. In an instruction like 'ADD D1,D2', the first subfield (D1) is called the source effective address field, and the second subfield (D2) is called the destination effective address field. Thus, the contents of D1 are added to the contents of D2 and the result is saved in register D2. In the instruction 'MOVE D1,D2', the first subfield (D1) is the sending field and the second subfield (D2) is the receiving field. In other words, for most two-operand instructions, the general format '*opcode source, destination*' applies.

#### 4.2.1.3 Disassembled Source Line

The disassembled source line may not look identical to the source line entered. The disassembler makes a decision on how it interprets the numbers used. If the number is an offset off of an address register, it is treated as a signed hexadecimal offset. Otherwise, it is treated as a straight unsigned hexadecimal. For example,

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

disassembles to:

```
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. As examples:

- a. BRA is returned for BT
- b. DBF is returned for DBRA

#### NOTE

The assembler recognizes two forms of mnemonics for two branch instructions. The BT form (branch conditionally true) has the same opcode as the BRA instruction. Also, DBRA (decrement and branch always) and DBF (never true, decrement, and branch) mnemonics are different forms for the same instruction. In each case, the assembler will accept both forms.

#### 4.2.1.4 Mnemonics and Delimiters

The assembler recognizes all MC68030 instruction mnemonics. Numbers are recognized as binary, octal, decimal, and hexadecimal, with hexadecimal the default case.

- a. Decimal - is a string of decimal digits (0 through 9) preceded by an ampersand (&). Examples are: &12334, -&987654321.
- b. Hexadecimal - is a string of hexadecimal digits (0 through 9, A through F) preceded by an optional dollar sign (\$). An example is: \$AFES.

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.

```
00003000 21FC0000 12345678      MOVE.L  #$1234,(5678).W
005000   0053                DC.W    'S'
005002   223C41424344       MOVE.L  #'ABCD',D1
005008   3536                DC.W    '56'
```

The assembler/disassembler recognizes/references these register mnemonics:

Pseudo Registers

R0-R7      User Offset Registers

Main Processor Registers

PC            Program Counter. Used only in forcing PC-relative addressing.  
 SR            Status Register.  
 CCR          Condition Codes Register (Lower eight bits of SR).  
 USP          User Stack Pointer.  
 MSP          Master Stack Pointer.  
 ISP          Interrupt Stack Pointer.  
 VBR          Vector Base Register.  
 SFC          Source Function Code Register.  
 DFC          Destination Function Code Register.  
 CACR         Cache Control Register.  
 CAAR         Cache Address Register.  
 D0-D7        Data registers.  
 A0-A7        Address Registers. A7 represents the active system stack pointer, (one of USP, MSP, or ISP), as specified by M and S bits of status register (SR).

Memory Management Unit Registers

MMUSR        Status Register  
 CRP         CPU Root Pointer  
 SRP         Supervisor Root Pointer  
 TC          Translation Control Register  
 TTO         Transparent Translation 0  
 TT1         Transparent Translation 1

=====

Floating Point Coprocessor Registers

=====

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

=====

4.2.1.5 Character Set

The character set recognized by the 141Bug assembler is a subset of ASCII, and these are listed as follows:

- a. The letters A through Z (uppercase and lowercase)
- b. The integers 0 through 9
- c. Arithmetic operators: + - \* / << >> ! &
- d. Parentheses ( )
- e. Characters used as special prefixes:
  - # (pound sign) specifies the immediate form of addressing.
  - \$ (dollar sign) specifies a hexadecimal number.
  - & (ampersand) specifies a decimal number.
  - @ (commercial at sign) specifies an octal number.
  - % (percent sign) specifies a binary number.
  - ' (apostrophe) specifies an ASCII literal character string.
- f. Five separating characters:
  - Space
  - , (comma)
  - . (period)
  - / (slash)
  - (dash)
- g. The character \* (asterisk) indicates current location.

4.2.2 Addressing Modes

Effective addressing modes, combined with operation codes, define the particular function to be performed by a given instruction. Effective addressing and data organization are described in detail in Section 2, "Data Organization and Addressing Capabilities", of the MC68030 User's Manual].

The addressing modes of the MC68030 which are accepted by the 141Bug one-line assembler are summarized in Table 4-1.

TABLE 4-1. 141Bug Assembler Addressing Modes

FORMAT	DESCRIPTION
Dn	Data register direct.
An	Address register direct.
(An)	Address register indirect.
(An)+	Address register indirect with postincrement.
-(An)	Address register indirect with predecrement.
d(An)	Address register indirect with displacement.
d(An,Xi)	Address register indirect with index, 8-bit displacement.
(bd,An,Xi)	Address register indirect with index, base displacement.
[[bd,An],Xi,od)	Address register memory indirect postindexed.
[[bd,An,Xi],od)	Address register memory indirect pre-indexed.
(d16,PC)	Program Counter indirect with displacement.
(d8,PC,Xi)	Program Counter indirect with index, 8-bit displacement.
(bd,PC,Xi)	Program Counter indirect with index, base displacement.
[[bd,PC],Xi,od)	Program Counter memory indirect postindexed.
[[bd,PC,Xi],od)	Program Counter memory indirect pre-indexed.
(xxxx).W	Absolute word address.
(xxxx).L	Absolute long address.
#xxxx	Immediate data.

The user may use an expression in any numeric field of these addressing modes. The assembler has a built-in expression evaluator. It supports the following operands types:

- a. Binary numbers (%10 )
- b. Octal numbers (@765..0)
- c. Decimal numbers (&987..0)
- d. Hexadecimal numbers (\$FED..0)
- e. String literals ('CHAR')
- f. Offset registers (R0-R7 )
- g. Program counter (\*)

Allowed operators are:

- a. Addition +
- b. Subtraction -
- c. Multiply \*
- d. Divide /
- e. Shift left <<
- f. Shift right >>
- g. Bitwise OR !
- h. Bitwise AND &

The order of evaluation is strictly left to right with no precedence granted to some operators over others. The only exception to this is when the user forces the order of precedence through the use of parentheses.

Possible points of confusion:

- a. The user should keep in mind that where a number is intended and it could be confused with a register, it must be differentiated in some way. For example:

```

CLR      DO      means CLR.W register DO.  On the other hand,

CLR      $DO
CLR      ODO
CLR      +DO
CLR      DO+0    all mean CLR.W memory location $DO.
```

- b. With the use of '\*' to represent both multiply and program counter, how does the assembler know when to use which definition?

For parsing algebraic expressions, the order of parsing is:

*operand operator operand operator. . .*

with a possible left or right parenthesis.

Given the above order, the assembler can distinguish by placement which definition to use. For example:

```

***      means PC  x  PC
*+*      means PC  +  PC
2**      means 2   *  PC
*&&16    means PC  AND &16
```

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

- a. Address register indirect with index, base displacement.
- b. Address register memory indirect postindexed.
- c. Address register memory indirect pre-indexed.
- d. Program counter indirect with index, base displacement.
- e. Program counter memory indirect postindexed.
- f. Program counter memory indirect pre-indexed.

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

- a. The user may terminate the operand at any time by specifying ')'. Example:

```

CLR      ( )      or
CLR      (,,)     is equivalent to

CLR      (0.N,ZAO,ZDO.W*1)
```

b. The user may skip a field by "stepping past" it with a comma. Example:

CLR (D7) is equivalent to

CLR (\$D7,ZA0,ZD0.W\*1)

but

CLR (,,D7) is equivalent to

CLR (0.N,ZA0,D7.W\*1)

c. If the user does not specify the base register, the default 'ZA0' is forced.

d. If the user does not specify the index register, the default 'ZD0.W\*1' is forced.

e. Any unspecified displacements are defaulted to '0.N'.

The rules for parsing the memory indirect addressing modes are the same as above with the following additions.

a. The subfield that begins with '[' must be terminated with a matching ']'.

b. If the text given is insufficient to distinguish between the pre-indexed or postindexed addressing modes, the default is the pre-indexed form.

#### 4.2.3 DC.W Define Constant Directive

The format for the DC.W directive is: DC.W *operand*

The function of this directive is to define a constant in memory. The DC.W directive can have only one operand (16-bit value) which can contain the actual value (decimal, hexadecimal, or ASCII). Alternatively, the operand can be an expression which can be assigned a numeric value by the assembler. The constant is aligned on a word boundary as word (.W) size is specified. An ASCII string is recognized when characters are enclosed inside single quotes (' '). Each character (seven bits) is assigned to a byte of memory, with the eighth bit (MSB) always equal to zero. If only one byte is entered, the byte is right justified. A maximum of two ASCII characters may be entered for each DC.W directive. Examples are:

00010022	04D2	DC.W	&1234	Decimal number
00010024	AAFE	DC.W	AAFE	Hexadecimal number
00010026	4142	DC.W	'AB'	ASCII string
00010028	5443	DC.W	'TB'+1	Expression
0001002A	0043	DC.W	'C'	ASCII character is right justified

#### 4.2.4 SYSCALL System Call Directive

The function of this directive is to aid the user in making the TRAP #15 calls to system functions. The format for this directive is:

```
SYSCALL function name
```

For example, the following two pieces of code produce identical results.

```
TRAP #15  
DC.W 0
```

or

```
SYSCALL .INCHR
```

Refer to Chapter 5 (SYSTEM CALLS) for a complete listing of all the functions provided.

#### 4.3 ENTERING AND MODIFYING SOURCE PROGRAMS

User programs are entered into the memory using the one-line assembler/disassembler. The program is entered in assembly language statements on a line-by-line basis. The source code is not saved as it is converted immediately to machine code upon entry. This imposes several restrictions on the type of source line that can be entered.

Symbols and labels, other than the defined instruction mnemonics, are not allowed. The assembler has no means to store the associated values of the symbols and labels in lookup tables. This forces the programmer to use memory addresses and to enter data directly rather than use labels.

Also, editing is accomplished by retyping the entire new source line. Lines can be added or deleted by moving a block of memory data to free up or delete the appropriate number of locations (refer to Block Move (BM) command).

##### 4.3.1 Invoking the Assembler/Disassembler

The assembler/disassembler is invoked using the ;DI option of the Memory Modify (MM) and Memory Display (MD) commands:

```
MM addr ;DI
```

```
      where   (CR)      sequences to next instruction  
            .(CR)      exits command
```

and

```
MD[S] addr[:count | addr];DI
```

The MM (;DI option) is used for program entry and modification. When this command is used, the memory contents at the specified location are disassembled and displayed. A new or modified line can be entered if desired.

The disassembled line can be an MC68030 instruction, a SYSCALL, or a DC.W directive. If the disassembler recognizes a valid form of some instruction, the instruction will be returned; if not (random data occurs), the DC.W \$XXXX (always hex) is returned. Because the disassembler gives precedence to instructions, a word of data that corresponds to a valid instruction will be returned as the instruction.

#### 4.3.2 Entering a Source Line

A new source line is entered immediately following the disassembled line, using the format discussed in paragraph 4.2.1:

```
141-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:

```
141-Bug> MM 10000;DI
00010000 528B                ADDQ.L #1,A3
00010002 4282                CLR.L D2 ?
```

If a hardcopy terminal is being used, the above example will look as follows:

```
141-Bug> MM 10000;DI
00010000 2600                MOVE.L D0,D3 ?  ADDQ.L #1,A3
00010000 528B                ADDQ.L #1,A3
00010002 4282                CLR.L D2 ?
```

Another program line can now be entered. Program entry continues in like manner until all lines have been entered. A period is used to exit the MM command.

If an error is encountered during assembly of the new line, the assembler displays the line unassembled with a "^" under the field suspected of causing the error and an error message is displayed. The location being accessed is redisplayed:

```
141-Bug> m 10000;di
00010000 528B                ADDQ.L #1,A3 ? lea.l 5(a0,d8),a4
00010000                LEA.L 5(A0,D8),A4
-----^
*** Unknown Field ***
00010000 528B                ADDQ.L #1,A3 ?
```

### 4.3.3 Entering Branch and Jump Addresses

When entering a source line containing a branch instruction (BRA, BGT, BEQ, etc.), do not enter the offset to the branch destination in the operand field of the instruction. The offset is calculated by the assembler. The user must append the appropriate size extension to the branch instruction.

To reference a current location in an operand expression, the character "\*" (asterisk) can be used. Examples are:

```

00030000  60004094          BRA *+$4096
00030000  60FE              BRA.B *
00030000  4EF90003 0000     JMP *
00030000  4EF00130 00030000 JMP (*,A0,DO)
    
```

In the case of forward branches or jumps, the absolute address of the destination may not be known as the program is being entered. The user may temporarily enter an "\*" for branch to self in order to reserve space. After the actual address is discovered, the line containing the branch instruction can be re-entered using the correct value.

#### NOTE

Branch sizes must be entered as ".B" or ".W" as opposed to ".S" and ".L".

### 4.3.4 Assembler Output/Program Listings

A listing of the program is obtained using the Memory Display (MD) command with the ;DI option. The MD command requires both the starting address and the line count to be entered in the command line. When the ;DI option is invoked, the number of instructions disassembled and displayed is equal to the line count.

To obtain a hard copy listing of a program, use the Printer Attach (PA) command to activate the printer port. An MD to the terminal then causes a listing on the terminal and on the printer.

Note again, that the listing may not correspond exactly to the program as entered. As discussed in paragraph 4.2.1.3, the disassembler displays in signed hexadecimal any number it interprets as an offset off of an address register; all other numbers are displayed in unsigned hexadecimal.

## CHAPTER 5 - SYSTEM CALLS

## 5.1 INTRODUCTION

This chapter describes the 141Bug TRAP #15 handler, which allows system calls from user programs. The system calls can be used to access selected functional routines contained within 141Bug, including input and output routines. TRAP #15 may also be used to transfer control to 141Bug at the end of a user program (refer to the .RETURN function in this chapter).

In the descriptions of some input and output functions, reference is made to the "default input port" or the "default output port". After power-up or reset, the default input and output port is initialized to be port 0 (the MVME141 serial port 1). The defaults may be changed, however, using the .REDIR\_I and .REDIR\_O functions, as described in this chapter.

## 5.1.1 Invoking System Calls Through TRAP #15

To invoke a system call from a user program, simply insert a TRAP #15 instruction into the source program. The code corresponding to the particular system routine is specified in the word following the TRAP opcode, as shown in the following example.

Format in user program:

```
TRAP #15   System call to 141Bug
DC.W $xxxx Routine being requested (xxxx = code)
```

In some of the examples shown in the following descriptions, a SYSCALL macro is used. This macro automatically assembles the TRAP #15 call followed by the Define Constant for the function code. For clarity, the SYSCALL macro is as follows:

```
SYSCALL      MACRO
              TRAP      #15
              DC.W      \1
              ENDM
```

Using the SYSCALL macro, the system call would appear in the user program as follows:

```
SYSCALL      routine name
```

It is, of course, necessary to create an equate file with the routine names equated to their respective codes.

When using the 141Bug one-line assembler/disassembler, the SYSCALL macro and the equates are predefined. Simply write in "SYSCALL" followed by a space and the function, then carriage return.

Example:

```
141-Bug> M 3000;DI
0000 3000 00000000   ORI.B #$.D0? SYSCALL .OUTLN
0000 3000 4E4F0022   SYSCALL .OUTLN
0000 3004 00000000   ORI.B #$.D0? .
141-Bug>
```

### 5.1.2 String Formats for I/O

Within the context of the TRAP #15 handler there are two formats for strings:

Pointer/Pointer Format - The string is defined by a pointer to the first character and a pointer to the last character + 1.

Pointer/Count Format - The string is defined by a pointer to a count byte, which contains the count of characters in the string, followed by the string itself.

A line is defined as a string followed by a carriage return and a line feed: (CR)(LF).

## 5.2 SYSTEM CALL ROUTINES

The TRAP #15 functions are summarized in Table 5-1. Refer to the write-ups on the utilities for specific use information.

TABLE 5-1. 141Bug System Call Routines

CODE	FUNCTION	DESCRIPTION
\$0000	.INCHR	Input character
\$0001	.INSTAT	Input serial port status
\$0002	.INLN	Input line (pointer/pointer format)
\$0003	.READSTR	Input string (pointer/count format)
\$0004	.READLN	Input line (pointer/count format)
\$0005	.CHKBRK	Check for break
\$0010	.DSKRD	Disk read
\$0011	.DSKWR	Disk write
\$0012	.DSKCFG	Disk configure
\$0014	.DSKFMT	Disk format
\$0015	.DSKCTRL	Disk control
\$0020	.OUTCHR	Output character
\$0021	.OUTSTR	Output string (pointer/pointer format)
\$0022	.OUTLN	Output line (pointer/pointer format)
\$0023	.WRITE	Output string (pointer/count format)
\$0024	.WRITELN	Output line (pointer/count format)
\$0025	.WRITDLN	Output line with data (pointer/count format)
\$0026	.PCRLF	Output carriage return and line feed
\$0027	.ERASLN	Erase line
\$0028	.WRITD	Output string with data (pointer/count format)
\$0029	.SNDBRK	Send break
\$0040	.TM_INI	Timer initialization routine
\$0041	.TM_STRO	Start timer at T=0
\$0042	.TM_RD	Timer read function
\$0043	.DELAY	Timer delay function
\$0050	.RTC_TM	Time initialization for RTC
\$0051	.RTC_DT	Date initialization for RTC
\$0052	.RTC_DSP	Display time from RTC
\$0053	.RTC_RD	Read the RTC registers
\$0060	.REDIR	Redirect I/O of a TRAP #15 function
\$0061	.REDIR_I	Redirect input
\$0062	.REDIR_O	Redirect output
\$0063	.RETURN	Return to 141Bug
\$0064	.BINDEC	Convert binary to Binary Coded Decimal (BCD)
\$0067	.CHANGEV	Parse value
\$0068	.STRCMP	Compare two strings (pointer/count format)
\$0069	.MULU32	Multiply two 32-bit unsigned integers
\$006A	.DIVU32	Divide two 32-bit unsigned integers
\$006B	.CHK_SUM	Generate checksum
\$0070	.BRD_ID	Return pointer to board ID packet

## 5.2.1 .INCHR Function

.INCHR

TRAP FUNCTION: .INCHR - Input character routine

CODE: \$0000

DESCRIPTION: Reads a character from the default input port. The character is returned in the stack.

ENTRY CONDITIONS:

SP ==> Space for character (byte)  
 Word fill (byte)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Character (byte)  
 Word fill (byte)

EXAMPLE:	SUBQ.L	#2,SP	Allocate space for result.
	SYSCALL	.INCHR	Call .INCHR.
	MOVE.B	(SP)+,D0	Load character in D0.

## 5.2.2 .INSTAT Function

.INSTAT

TRAP FUNCTION: .INSTAT - Input serial port status

CODE: \$0001

DESCRIPTION: Used to see if there are characters in the default input port buffer. The condition codes are set to indicate the result of the operation.

## ENTRY CONDITIONS:

No arguments or stack allocation required

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

Z(ero) = 1 if the receiver buffer is empty

EXAMPLE:	LOOP	SYSCALL	.INSTAT	Any characters?
		BEQ.S	EMPTY	No, branch
		SUBQ.L	#2,A7	Yes, then
		SYSCALL	.INCHR	Read them
		MOVE.B	(SP)+,(A0)+	In buffer
		BRA.S	LOOP	Check for more
	EMPTY			

## 5.2.3 .INLN Function

.INLN

TRAP FUNCTION: .INLN - Input line routine

CODE: \$0002

DESCRIPTION: Used to read a line from the default input port. The buffer size should be at least 256 bytes.

ENTRY CONDITIONS:

SP ==&gt; Address of string buffer (longword)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==&gt; Address of last character in the string+1 (longword)

EXAMPLE: If A0 contains the address where the string is to go;

SUBQ.L	#4,A7	Allocate space for result.
PEA.L	(A0)	Push pointer to destination.
TRAP	#15	(May also invoke by SYSCALL
DC.W	2	macro ("SYSCALL .INLN".)
MOVE.L	(A7)+,A1	Retrieve address of last character+1.

NOTES: A line is a string of characters terminated by (CR). The maximum allowed size is 254 characters. The terminating (CR) is not considered part of the string, but it is returned in the buffer, that is, the returned pointer points to it. Control character processing as described in paragraph 1.8, Terminal Input/Output Control, is in effect.

## 5.2.4 .READSTR Function

.READSTR

TRAP FUNCTION: .READSTR - Read string into variable-length buffer

CODE: \$0003

DESCRIPTION: Used to read a string of characters from the default input port into a buffer. On entry, the first byte in the buffer indicates the maximum number of characters that can be placed in the buffer. The buffer size should at least be equal to that number+2. The maximum number of characters that can be placed in a buffer is 254 characters. On exit, the count byte indicates the number of characters in the buffer. Input terminates when a (CR) is received. The (CR) character appears in the buffer, although it is not included in the string count. All printable characters are echoed to the default output port. The (CR) is not echoed. Some control character processing is done:

^G	Bell	Echoed.
^X	Cancel line	Line is erased.
^H	Backspace	Last character is erased.
(DEL)	Same as backspace	Last character is erased.
(LF)	Line Feed	Echoed.
(CR)	Carriage Return	Terminates input.

All other control characters are ignored.

## ENTRY CONDITIONS:

SP ==> Address of input buffer (longword)

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack

The count byte contains the number of bytes in the buffer.

EXAMPLE: If A0 contains the string buffer address;

MOVE.B	#75,(A0)	Set maximum string size.
PEA.L	(A0)	Push buffer address.
TRAP	#15	(May also invoke by SYSCALL
DC.W	3	macro ("SYSCALL .READSTR").)
MOVE.B	(A0),D0	Read actual string size.

NOTES: This routine allows the caller to dictate the maximum length of input to be less than 254 characters. If more characters are entered, then the buffer input is truncated. Control character processing as described in paragraph 1.8, Terminal Input/Output Control, is in effect.

## 5.2.5 .READLN Function

.READLN

TRAP FUNCTION: .READLN - Read line to fixed-length buffer

CODE: \$0004

DESCRIPTION: Used to read a string of characters from the default input port. Characters are echoed to the default output port. A string consists of a count byte followed by the characters read from the input. The count byte indicates the number of characters in the input string, excluding (CR)(LF). A string may be up to 254 characters.

## ENTRY CONDITIONS:

SP ==&gt; Address of input buffer (longword)

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==&gt; Top of stack

The first byte in the buffer indicates the string length.

EXAMPLE: If A0 points to a 256-byte buffer;

PEA.L	(A0)	Load buffer address
SYSCALL	.READLN	And read a line from default input port.

NOTES: The caller must allocate 256 bytes for a buffer. Input may be up to 254 characters. (CR)(LF) is sent to default output following echo of input. Control character processing as described in paragraph 1.8, Terminal Input/Output Control, is in effect.

## 5.2.6 .CHKBRK Function

.CHKBRK

TRAP FUNCTION: .CHKBRK - Check for break

CODE: \$0005

DESCRIPTION: Returns "Zero" status in the condition code register if break status is detected at the default input port.

ENTRY CONDITIONS:

No arguments or stack allocation required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

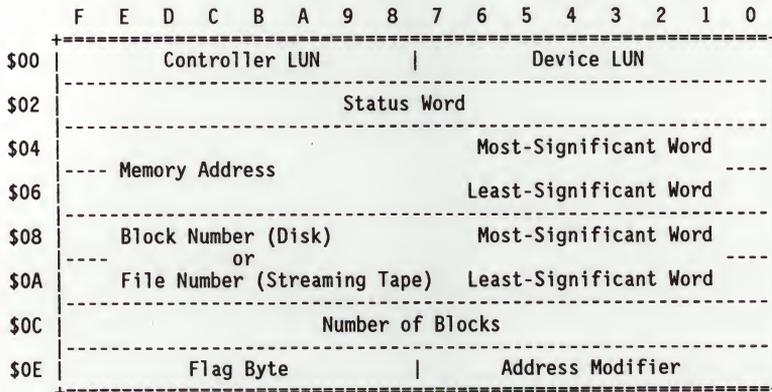
Z flag in CCR if break detected

EXAMPLE:   SYSCALL .CHKBRK  
          BEQ       BREAK

## 5.2.7 .DSKRD, .DSKWR Functions

.DSKRD  
.DSKWRTRAP FUNCTIONS: .DSKRD - Disk read function  
.DSKWR - Disk write functionCODES: \$0010  
\$0011

DESCRIPTION: These functions are used to read and write blocks of data from/to the specified disk or tape device. Information about the data transfer is passed in a command packet which has been built somewhere in memory. (The user program must first manually prepare the packet.) The address of the packet is passed as an argument to the function. The same command packet format is used for .DSKRD and .DSKWR. It is eight words in length and is arranged as follows:



## Field descriptions:

- Controller LUN - Logical Unit Number (LUN) of controller to use.
- Device LUN - Logical Unit Number of device to use.
- Status Word - This status word reflects the result of the operation. It is zero if the command completed without errors. Refer to Appendix F for meanings of returned error codes.
- Memory Address - Address of buffer in memory. On a disk read, data is written starting at this address. On a disk write, data is read starting at this address.

.DSKRD  
.DSKWR

- Block Number** - For disk devices, this is the block number where the transfer starts. On a disk read, data is read starting at this block. On a disk write, data is written starting at this block.
- File Number** - For streaming tape devices, this is the file number where the transfer starts. This field is used if the IFN bit in the Flag Byte is cleared (refer to the Flag Byte description).
- Number of Blocks** - This field indicates the number of blocks to read from the disk (.DSKRD) or to write to the disk (.DSKWR). For streaming tape devices, the actual number of blocks transferred is returned in this field.
- Flag Byte** - The flag byte is used to specify variations of the same command, and to receive special status information. Bits 0 through 3 are used as command bits, and bits 4 through 7 are used as status bits. For disk devices, this field must be set to zero. For streaming tape devices, the following bits are defined:
- Bit 7 File Mark flag. If 1, a file mark 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 flag. If 0, reads or writes are done until the specified block count is exhausted. If 1, reads are done until the count is exhausted or until a file mark is found. If 1, writes are terminated with a filemark.
- Address Modifier** - VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If nonzero, the specified value is used.

.DSKRD  
.DSKWR

## ENTRY CONDITIONS:

SP ==> Address (longword)      Address of command packet

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
Status word of command packet is updated.  
Data is written into memory as a result of .DSKRD function.  
Data is written to disk as a result of .DSKWR function.  
Z(ero) = Set to 1 if no errors.

EXAMPLE: If A0, A1 point to packets formatted as specified above;

	PEA.L	(A0)	
	SYSCALL	.DSKRD	Read from disk
	BNE	ERROR	Branch if error
	PEA.L	(A1)	
	SYSCALL	.DSKWR	Write to disk
	BNE	ERROR	Branch if error
	.		
	.		
ERROR	xxxxx	xxx	Handle error
	xxxxx	xxx	

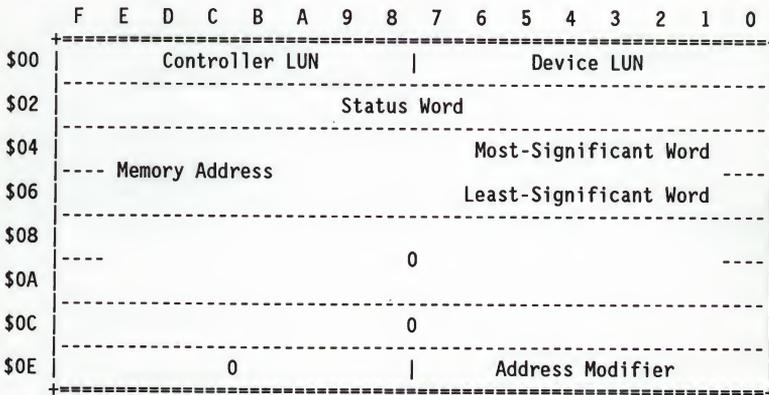
5.2.8 .DSKCFIG Function

.DSKCFIG

TRAP FUNCTION: .DSKCFIG - Disk configure function

CODE: \$0012

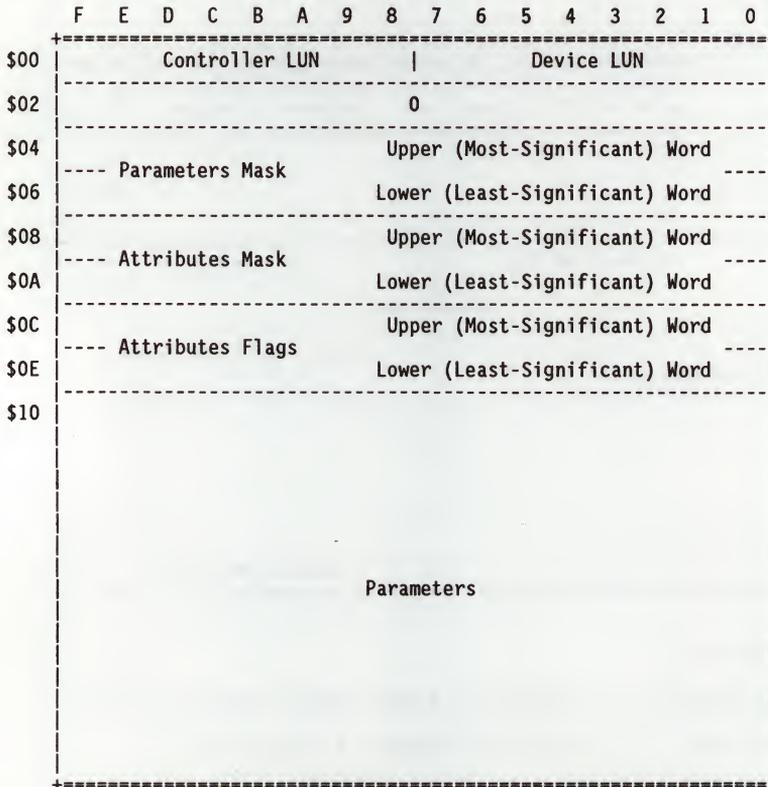
DESCRIPTION: This function allows the user to change the configuration of the specified device. It effectively performs an "IOT under program control". All the required parameters are passed in a command packet which has been built somewhere in memory. The address of the packet is passed as an argument to the function. The packet format is as follows:



Field descriptions:

- Controller LUN - Logical Unit Number (LUN) of controller to use.
- Device LUN - Logical Unit Number of device to use.
- Status Word - This status word reflects the result of the operation. It is zero if the command completed without errors. Refer to Appendix F for meanings of returned error codes.
- Memory Address - Contains a pointer to a Device Descriptor Packet that contains the configuration information to be changed.
- Address Modifier - VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If nonzero, the specified value is used.

The Device Descriptor Packet is as follows:



Most of the fields in the device descriptor packet are equivalent to the fields defined in the CFGA configuration area block, as described in Appendix D. In the field descriptions following, reference is made to the equivalent field in the CFGA whenever possible. For additional information on these fields, refer to Appendix D.

- Controller LUN - Same as in command packet.
- Device LUN - Same as in command packet.

## .DSKCFIG

- Parameters Mask - Equivalent to the IOSPRM and IOSEPRM fields, with the lower word equivalent to IOSPRM, and the upper word equivalent to IOSEPRM.
- Attributes Mask - Equivalent to the IOSATM and IOSEATM fields, with the lower word equivalent to IOSATM, and the upper word equivalent to IOSEATM.
- Attributes Flags - Equivalent to the IOSATW and IOSEATW fields, with the lower word equivalent to IOSATW, and the upper word equivalent to IOSEATW.
- Parameters - The parameters used for device reconfiguration are specified in this area. Most parameters have an exact CFGA equivalent. The following list shows the field name, offset from start of packet, length, equivalent CFGA field, and short description of each field. Those parameters that do not have an exact equivalent are indicated with "\*\*", and are explained after the list.

FIELD NAME	OFFSET (BYTES)	LENGTH (BYTES)	CFGA EQUIV.	DESCRIPTION
P_DDS*	\$10	1	-	Device descriptor size
P_DSR	\$11	1	IOSSR	Step rate
P_DSS*	\$12	1	IOSPSM	Sector size (encoded)
P_DBS*	\$13	1	IOSREC	Block size (encoded)
P_DST*	\$14	2	IOSSPT	Sectors/track
P_DIF	\$16	1	IOSILV	Interleave factor
P_DSO	\$17	1	IOSSOF	Spiral offset
P_DSH*	\$18	1	IOSSHD	Starting head
P_DNH	\$19	1	IOSHDS	Number of heads
P_DNCYL	\$1A	2	IOSTRK	Number of cylinders
P_DPCYL	\$1C	2	IOSPCOM	Precompensation cylinder
P_DRWCYL	\$1E	2	IOSRWCC	Reduced write current cylinder
P_DECCB	\$20	2	IOSECC	ECC data burst length
P_DGAP1	\$22	1	IOSGPB1	Gap 1 size
P_DGAP2	\$23	1	IOSGPB2	Gap 2 size
P_DGAP3	\$24	1	IOSGPB3	Gap 3 size
P_DGAP4	\$25	1	IOSGPB4	Gap 4 size
P_DSSC	\$26	1	IOSSSC	Spare sectors count
P_DRUNIT	\$27	1	IOSRUNIT	Reserved area units
P_DRCALT	\$28	2	IOSRSVC1	Reserved count for alternates
P_DRCCTR	\$2A	2	IOSRSVC2	Reserved count for controller

**.DSKCFIG**

## List Notes:

P\_DDS This field is for internal use only, and does not have an equivalent CFGA field. It should be set to 0.

P\_DSS This is a one byte encoded field, whereas the IOSPSM field is a two byte unencoded field containing the actual number of bytes per sector. The P\_DSS field is encoded as follows:

\$00	128 bytes
\$01	256 bytes
\$02	512 bytes
\$03	1024 bytes
\$04-\$FF	Reserved encodings

P\_DBS This is a one byte encoded field, whereas the IOSREC field is a two byte unencoded field containing the actual number of bytes per record (block). The P\_DBS field is encoded as follows:

\$00	128 bytes
\$01	256 bytes
\$02	512 bytes
\$03	1024 bytes
\$04-\$FF	Reserved encodings

P\_DST This is a two byte field, whereas the IOSSPT field is one byte.

P\_DSH This is a one byte field, whereas the IOSSHD field is two bytes. This field is equivalent to the lower byte of IOSSHD.

## ENTRY CONDITIONS:

SP ==> Address (longword)      Address of command packet

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
 Status word of command packet is updated.  
 The device configuration is changed.  
 Z(ero) = Set to 1 if no errors.

EXAMPLE: If A0 points to a packet formatted as specified above;

	PEA.L	(A0)	Load command packet
	SYSCALL	.DSKCFIG	Reconfigure device
	BNE	ERROR	Branch if error
	.		
	.		
	.		
ERROR	xxxxx	xxx	Handle error
	xxxxx	xxx	

## 5.2.9 .DSKFMT Function

.DSKFMT

TRAP FUNCTION: .DSKFMT - Disk format function

CODE: \$0014

DESCRIPTION: This function allows the user to send a format command to the specified device. The parameters required for the command are passed in a command packet which has been built somewhere in memory. The address of the packet is passed as an argument to the function. The format of the packet is as follows:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
\$00	Controller LUN							Device LUN								
\$02	Status Word															
\$04	Memory Address										Most-Significant Word					
\$06											Least-Significant Word					
\$08	Block Number (Disk)										Most-Significant Word					
\$0A											Least-Significant Word					
\$0C	0															
\$0E	Flag Byte							Address Modifier								

## Field descriptions:

- Controller LUN - Logical Unit Number (LUN) of controller to use.
- Device LUN - Logical Unit Number of device to use.
- Status Word - This status word reflects the result of the operation. It is zero if the command completed without errors. Refer to Appendix F for meanings of returned error codes.
- Memory Address - Address of buffer in memory. On a disk read, data is written starting at this address. On a disk write, data is read starting at this address.
- Block Number - For disk devices, when doing a format track, the track that contains this block number is formatted. This field is ignored for streaming tape devices.

.DSKFMT

Flag Byte - Contains additional information. Bit 0 is interpreted as follows for disk devices:

If 0, it indicates a "Format Track" operation. The track that contains the specified block is formatted.

If 1, it indicates a "Format Disk" operation. All the tracks on the disk are formatted.

For streaming tapes, bit 0 is interpreted as follows:

If 0, it selects a "Retension tape" operation. 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.

If 1, it selects an "Erase Tape" operation. This completely clears the tape of previous data and at the same time retensions the tape.

Address Modifier - VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If nonzero, the specified value is used.

ENTRY CONDITIONS:

SP ==> Address (longword)      Address of command packet

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
 Status word of command packet is updated.  
 Z(ero) = Set to 1 if no errors.

EXAMPLE: If A0 points to a packet formatted as specified above;

	PEA.L	(A0)	Load command packet
	SYSCALL	.DSKFMT	format device
	BNE	ERROR	If errors, branch
	.		
	.		
	.		
ERROR	xxxxx	xxx	Handle error
	xxxxx	xxx	

## 5.2.10 .DSKCTRL Function

.DSKCTRL

TRAP FUNCTION: .DSKCTRL - Disk control function

CODE: \$0015

DESCRIPTION: This function is used to implement any special device control functions that cannot be accommodated easily with any of the other disk functions. At the present, the only defined function is SEND packet, which allows the user to send a packet in the specified format of the controller. The required parameters are passed in a command packet which has been built somewhere in memory. The address of the packet is passed as an argument to the function.

The packet is as follows:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
\$00	Controller LUN							Device LUN								
\$02	Status Word															
\$04	Memory Address								Most-Significant Word							
\$06									Least-Significant Word							
\$08																
\$0A	0															
\$0C	0															
\$0E	0							Address Modifier								

## Field descriptions:

- Controller LUN - Logical Unit Number (LUN) of controller to use.
- Device LUN - Logical Unit Number of device to use.
- Status Word - This status word reflects the result of the operation. It is zero if the command completed without errors. Refer to Appendix F for meanings of returned error codes.
- Memory Address - Contains a pointer to the controller packet to send. Note that the controller packet to send (as opposed to the command packet) is controller and device dependent. Information about this packet should be found in the user's manual for the controller and device being accessed.

**.DSKCTRL**

Address Modifier - VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If nonzero, the specified value is used.

## ENTRY CONDITIONS:

SP ==> Address (longword)      Address of command packet

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
 Status word of command packet is updated.  
 Additional side effects depend on the packet sent to the controller.  
 Z(ero) = Set to 1 if no errors.

EXAMPLE: If A1 points to a packet formatted as specified above;

	PEA.L	(A1)	Load command packet
	SYSCALL	.DSKCTRL	invoke control function
	BNE	ERROR	If errors, branch
	.		
	.		
ERROR	xxxxx	xxx	Handle error
	xxxxx	xxx	

## 5.2.11 .OUTCHR Function

.OUTCHR

TRAP FUNCTION: .OUTCHR - Output character routine

CODE: \$0020

DESCRIPTION: This function outputs a character to the default output port.

ENTRY CONDITIONS:

SP ==> Character (byte)  
 Word fill (byte) (Placed automatically by MPU)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
 Character is sent to the default I/O port.

EXAMPLE:	MOVE.B	D0,-(SP)	Send character in D0
	SYSCALL	.OUTCHR	To default output port.

## 5.2.12 .OUTSTR, .OUTLN Functions

.OUTSTR  
.OUTLN

TRAP FUNCTIONS: .OUTSTR - Output string to default output port  
.OUTLN - Output string along with (CR)(LF)

CODES: \$0021  
\$0022

DESCRIPTION: .OUTSTR outputs a string of characters to the default output port. .OUTLN outputs a string of characters followed by a (CR)(LF) sequence.

## ENTRY CONDITIONS:

SP ==> Address of first character (longword)  
+4 Address of last character+1 (longword)

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack

EXAMPLE: If A0 = start of string  
A1 = end of string+1

```
MOVEM.L A0/A1,-(SP)  Load pointers to string
SYSCALL .OUTSTR      And print it.
```

## 5.2.13 .WRITE, .WRITELN Functions

.WRITE  
.WRITELN

TRAP FUNCTIONS: .WRITE - Output string with no CR or LF  
.WRITELN - Output string with CR and LF

CODES: \$0023  
\$0024

DESCRIPTION: These output functions are designed to output strings formatted with a count byte followed by the characters of the string. The user passes the starting address of the string. The output goes to the default output port.

## ENTRY CONDITIONS:

Four bytes of parameter positioned in stack as follows:

SP ==> Address of string (longword)

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
Parameter stack space will have been deallocated.

EXAMPLE: For example, the following section of code ...

```

MESSAGE1 DC.B 9, 'MOTOROLA '
MESSAGE2 DC.B 9, 'QUALITY !'
      .
      .
      .
      PEA.L MESSAGE1(PC) Push address of string.
      SYSCALL .WRITE Use TRAP #15 macro.
      PEA.L MESSAGE2(PC) Push address of other string.
      SYSCALL .WRITE Invoke function again.

```

... would print out the following message:

MOTOROLA QUALITY!

Using function .WRITELN, however, instead of function .WRITE would output the following message:

MOTOROLA  
QUALITY!

NOTES: The string must be formatted such that the first byte (the byte pointed to by the passed address) contains the count (in bytes) of the string. There is no special character at the end of the string as a delimiter.

## 5.2.14 .PCRLF Function

.PCRLF

TRAP FUNCTION: .PCRLF - Print (CR)(LF) sequence

CODE: \$0026

DESCRIPTION: Sends a (CR)(LF) sequence to the default output port.

ENTRY CONDITIONS:

No arguments or stack allocation required.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

None

EXAMPLE:           SYSCALL .PCRLF           Output CRLF

## 5.2.15 .ERASLN Function

.ERASLN

TRAP FUNCTION: .ERASLN - Erase Line

CODE: \$0027

DESCRIPTION: Used to erase the line at the present cursor position. If the terminal type flag is set for hardcopy mode, a (CR)(LF) is issued instead.

## ENTRY CONDITIONS:

No arguments required.

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

The cursor is positioned at the beginning of a blank line.

EXAMPLE:           SYSCALL .ERASLN

## 5.2.16 .WRITD, .WRITDLN Functions

.WRITD  
.WRITDLN

TRAP FUNCTIONS: .WRITD - Output string with data  
.WRITDLN - Output string with data and (CR)(LF)

CODES: \$0028  
\$0025

DESCRIPTION: These trap functions take advantage of the monitor I/O routine which outputs a user string containing embedded variable fields. The user passes the starting address of the string and the address of a data stack containing the data which is inserted into the string. The output goes to the default output port.

## ENTRY CONDITIONS:

Eight bytes of parameter positioned in stack as follows:

SP ==> Address of string (longword)  
Data list pointer (longword)

A separate data stack or data list arranged as follows:

Data list pointer => Data for 1st variable in string (longword)  
Data for next variable (longword)  
Data for next variable (longword)  
etc.

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
Parameter stack space will have been deallocated.

EXAMPLE: For example, the following section of code ...

```
ERRMESSG   DC.B      $14,'ERROR CODE = |10,8Z|'

          MOVE.L    #3,-(A5)      Push error code on data stack.
          PEA.L     (A5)          Push data stack location.
          PEA.L     ERRMESSG(PC)  Push address of string.
          SYSCALL   .WRITDLN     Invoke function.
          TST.L    (A5)+         Deallocate data from data stack.
```

... would print out the following message:

ERROR CODE = 3

NOTES: 1. The string must be formatted such that the first byte (the byte pointed to by the passed address) contains the count (in bytes) of the string (including the data field specifiers, described in NOTE 2. following).

.WRITD  
.WRITDLN

2. Any data fields within the string must be represented as follows: "*| radix,fieldwidth[Z]|*" where *radix* is the base that the data is to be displayed in (in hexadecimal, for example, "A" is base 10, "10" is base 16, etc.) and *fieldwidth* is the number of characters this data is to occupy in the output. The data is right justified, and left-most characters are removed to make the data fit. The "Z" is included if it is desired to suppress leading zeros in output.
3. All data is to be placed in the stack as longwords. Each time a data field is encountered in the user string, a longword is read from the data stack to be displayed.
4. The data stack is not destroyed by this routine. If it is necessary that the space in the data stack be deallocated, then this must be done by the calling routine, as shown in the preceding example.

## 5.2.17 .SNDBRK Function

.SNDBRK

TRAP FUNCTION: .SNDBRK - Send break

CODE: \$0029

DESCRIPTION: Used to send a break to the default output port(s).

ENTRY CONDITIONS:

No arguments or stack allocation required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Each serial port specified by current default port list has sent "break".

EXAMPLE: SYSCALL .SNDBRK

## 5.2.18 .TM\_INI Function

.TM\_INI

TRAP FUNCTION: .TM\_INI - Timer initialization routine

CODE: \$0040

DESCRIPTION: This routine initializes the onboard timer, and also calculates a calibration factor used by the other timer functions. This routine should be used the first time the timer functions are used.

## ENTRY CONDITIONS:

No arguments required.

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

TM.CAL1(A5) loaded with calibration factor.

## EXAMPLE:

SYSCALL .TM\_INI            Initialize timer

## 5.2.19 .TM\_STRO Function

.TM\_STRO

TRAP FUNCTION: .TM\_STRO - Start timer at T=0

CODE: \$0041

DESCRIPTION: This routine first resets the timer to 0 and then starts it.

ENTRY CONDITIONS:

No arguments required.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Timer is started.

EXAMPLE:

SYSCALL .TM\_STRO

5.2.20 `.TM_RD` Function`.TM_RD`TRAP FUNCTION: `.TM_RD` - Timer read function

CODE: \$0042

DESCRIPTION: This routine is used to read the value of the timer (microseconds) Timer updates are interrupt driven.

ENTRY CONDITIONS:

SP ==&gt; Space for result (longword)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==&gt; Time in microseconds (longword)

EXAMPLE:

<code>SUBQ.L</code>	<code>#4,A7</code>	Allocate space for result
<code>SYSCALL</code>	<code>.TM_RD</code>	Read timer
<code>MOVE.L</code>	<code>(SP)+,D0</code>	Load time in microseconds

NOTE: Timer is stopped when a read is done. `TM_STRO` must be invoked to start timer at 0.

## 5.2.21 .DELAY Function

.DELAY

TRAP FUNCTION: .DELAY - Timer delay function

CODE: \$0043

DESCRIPTION: Used to generate accurate timing delays that are independent of the processor frequency and instruction execution rate. This function uses the onboard timer for operation. The user specifies the desired delay count in milliseconds. .DELAY returns to the caller after the specified delay count is exhausted. The onboard timer has to be initialized once before this function is called by invoking the .TM\_INI trap function.

ENTRY CONDITIONS:

SP ==&gt; Delay time in milliseconds (longword)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==&gt; Top of stack

EXAMPLE:

```

SYSCALL .TM_INI      Initialize timer
PEA.L   &15000      Load a 15 second delay
SYSCALL .DELAY
*
*
*
PEA.L   &50          Load a 50 millisecond delay
SYSCALL .DELAY

```

## 5.2.22 .RTC\_TM Function

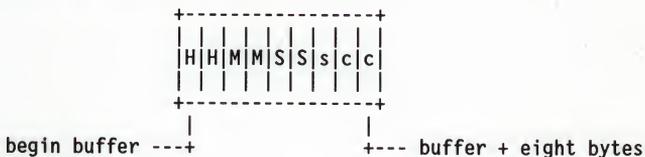
.RTC\_TM

TRAP FUNCTION: .RTC\_TM - Time initialization for RTC

CODE: \$0050

DESCRIPTION: Initializes the MK48T02 Real-Time Clock with the time that is located in a user-specified buffer.

The data input format can be either ASCII or unpacked BCD. The order of the data in the buffer is:



ENTRY CONDITIONS:

SP ==&gt; Time initialization buffer (address)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
Parameter is deallocated from stack.

EXAMPLE: Time is to be initialized to 2:05:32 PM with a calibration factor of -15 (s=sign, cc=value).

Data in BUFFER is 3134 3035 3332 2D 3135 or  
x1x4 x0x5 x3x2 2D x1x5. (x = don't care)PEA.L BUFFER(PC) Put buffer address on stack  
SYSCALL .RTC\_TM Initialize time and start clock

## 5.2.23 .RTC\_DT Function

.RTC\_DT

TRAP FUNCTION: .RTC\_DT - Date initialization for RTC

CODE: \$0051

DESCRIPTION: Initializes the MK48T02 Real-Time Clock with the date that is located in a user-specified buffer.

The data input format can be either ASCII or unpacked BCD. The order of the data in the buffer is:



## ENTRY CONDITIONS:

SP ==&gt; Date initialization buffer (address)

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
 Parameter is deallocated from stack.

EXAMPLE: Date is to be initialized to Monday, Nov. 18, 1988 (d = day of week).

Data in BUFFER is 3838 3131 3138 32 or  
 x8x8 x1x1 x1x8 x2. (x = don't care)

PEA.L	BUFFER(PC)	Put buffer address on stack
SYSCALL	.RTC_DT	Initialize date and start clock

## 5.2.24 .RTC\_DSP Function

.RTC\_DSP

TRAP FUNCTION: .RTC\_DSP - Display time from RTC

CODE: \$0052

DESCRIPTION: Displays the day of the week, date, and time in the following format:

(Day of week) MM/DD/YY hh:mm:ss

ENTRY CONDITIONS:

No arguments or stack allocation required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

The cursor is left at the end of the string.

EXAMPLE: SYSCALL .RTC\_DSP Displays the day, date, and time on the console.

## 5.2.25 .RTC\_RD Function

.RTC\_RD

TRAP FUNCTION: .RTC\_RD - Read the RTC registers

CODE: \$0053

DESCRIPTION: Used to read the Real-Time Clock registers. The data returned is in BCD. The last byte of the returned data is the calibration value (c); bit #5 is a sign bit (1 indicates positive, 0 indicates negative).

The order of the data in the buffer is:



## ENTRY CONDITIONS:

SP ==> Buffer address where RTC data is to be returned <long>

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
Buffer now contains date and time in BCD format.

EXAMPLE: A date and time of Saturday, May 11, 1988 2:05:32 PM are be returned in the buffer (d = day of week, c = calibration value).

Data in buffer = 88 05 11 07 14 05 32 xx (xx = unknown)

PEA.L	BUFFER(PC)	Put buffer address on stack
SYSCALL	.RTC_RD	Read timer

## 5.2.26 .REDIR Function

.REDIR

TRAP FUNCTION: .REDIR - Redirect I/O function

CODE: \$0060

DESCRIPTION: Used to select an I/O port and at the same time invoke a particular I/O function. The invoked I/O function reads or writes to the selected port.

## ENTRY CONDITIONS:

SP ==> Port	(word)
I/O function to call	(word)
Parameters of I/O function	(size specified by function)
Space for results	(size specified by function)

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Result	(size specified by function)
---------------	------------------------------

EXAMPLE: None

NOTES: To use .REDIR, the caller should first allocate space and push the parameters required by the desired I/O function onto the stack:

```

SUBQ.L    #2,A7        Allocate space (no parameters
                       required by .INCHR)

```

Then the parameters required by .REDIR should be pushed and the call made to .REDIR:

```

MOVE.W    #.INCHR,-(SP) Load function code
MOVE.W    #1,-(SP)      Load port number
SYSCALL   .REDIR        Redirect I/O function

```

Finally, the results should be popped from the stack:

```

MOVE.B    (SP)+,D0      Read character

```

The above example reads a character from port 1 using .REDIR.

## 5.2.27 .REDIR\_I, .REDIR\_O Functions

.REDIR\_I  
.REDIR\_OTRAP FUNCTIONS: .REDIR\_I - Redirect input  
.REDIR\_O - Redirect outputCODES: \$0061  
\$0062

DESCRIPTION: The .REDIR\_I and .REDIR\_O functions are used to change the default port number of the input and output ports, respectively. This is a permanent change, that is, it remains in effect until a new .REDIR command is issued.

ENTRY CONDITIONS:

SP ==&gt; Port Number &lt;word&gt;

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack  
.SIO\_IN - Loaded with a new mask if .REDIR\_I called  
.SIO\_OUT - Loaded with a new mask if .REDIR\_O calledEXAMPLE: MOVE.W #1,-(SP) Load port number  
SYSCALL .REDIR\_I Set it as the new default

## 5.2.28 .RETURN Function

**.RETURN**

TRAP FUNCTION: .RETURN - Return to 141Bug

CODE: \$0063

DESCRIPTION: Used to return control to 141Bug from the target program in an orderly manner. First, any breakpoints inserted in the target code are removed. Then, the target state is saved in the register image area. Finally, the routine returns to 141Bug.

## ENTRY CONDITIONS:

No arguments required.

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

Control is returned to 141Bug.

EXAMPLE: SYSCALL .RETURN Return to 141Bug.

NOTES: .RETURN must be used only by code that was started using 141Bug.

5

## 5.2.29 .BINDEC Function

.BINDEC

TRAP FUNCTION: .BINDEC - Calculate BCD equivalent of the binary number specified

CODE: \$0064

DESCRIPTION: Takes a 32-bit unsigned binary number and changes it to an equivalent BCD number.

ENTRY CONDITIONS:

SP ==> Argument:Hex number (longword)  
 Space for result (2 longword)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Decimal number (2 most significant DIGITS) (longword)  
 (8 least significant DIGITS) (longword)

EXAMPLE:	SUBQ.L	#8,A7	Allocate space for result
	MOVE.L	D0,-(SP)	Load hex number
	SYSCALL	.BINDEC	Call .BINDEC
	MOVE.L	(SP)+,D1/D2	Load result

## 5.2.30 .CHANGEV Function

.CHANGEV

TRAP FUNCTION: .CHANGEV - Parse value, assign to variable

CODE: \$0067

DESCRIPTION: Attempt to parse value in user specified buffer. If user's buffer is empty, prompt user for new value, otherwise update integer offset into buffer to skip "value". Display new value and assign to variable unless user's input is an empty string.

## ENTRY CONDITIONS:

SP ==> Address of 32-bit offset into user's buffer  
 Address of user's buffer (pointer/count format string)  
 Address of 32-bit integer variable to "change"  
 Address of string to use in prompting and displaying value

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack

EXAMPLE:	PROMPT	DC.B	\$14,'COUNT =  10,8 '	
	GETCOUNT	PEA.L	PROMPT(PC)	Point to prompt string
		PEA.L	COUNT	Point to variable to change
		PEA.L	POINT	Point to offset into buffer
		PEA.L	BUFFER	Point to buffer
		SYSCALL	.CHANGEV	Make the system call
		RTS		COUNT changed, return

NOTE: If the above code was called with BUFFER containing "1 3" in pointer/count format and POINT containing 2 (longword), then COUNT would be assigned the value 3, and POINT would contain 4 (pointing to first character past "3"). Note that POINT is the offset from the start address of the buffer (not the address of the first character in the buffer!) to the next character to process. In this case, a value of 2 in POINT indicates that the space between "1" and "3" is the next character to be processed. After calling .CHANGEV, the screen displays the following line:

COUNT = 3

If the above code was called again, nothing could be parsed from BUFFER, so a prompt would be issued. For purpose of example, the string "5" is entered in response to the prompt.

COUNT = 3? 5  
 COUNT = 5

If in the previous example nothing had been entered at the prompt, COUNT would retain its prior value.

COUNT = 3? (CR)  
 COUNT = 3

## 5.2.31 .STRCMP Function

.STRCMP

TRAP FUNCTION: .STRCMP - Compare two strings (pointer/count)

CODE: \$0068

DESCRIPTION: Comparison for equality is made and boolean flag is returned to caller. The flag is \$00 if the strings are not identical; otherwise it is \$FF.

ENTRY CONDITIONS:

SP ==> Address of string 1  
 Address of string 2  
 Three bytes (unused)  
 Byte to receive string comparison result

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Three bytes (unused)  
 Byte to receive string comparison result

EXAMPLE: If A1 and A2 contain addresses of the two strings;

SUBQ.L	#4,SP	Allocate longword to receive result.
PEA.L	(A1)	Push address of one string.
PEA.L	(A2)	Push address of the other string.
SYSCALL	.STRCMP	Compare the strings.
MOVE.L	(SP)+,D0	Pop boolean flag into data register.
TST.B	D0	Check boolean flag.
BNE	ARE_SAME	Branch if strings are identical.

## 5.2.32 .MULU32 Function

.MULU32

TRAP FUNCTION: .MULU32 - Unsigned 32-bit x 32-bit multiply

CODE: \$0069

DESCRIPTION: Two 32-bit unsigned integers are multiplied and the product is returned on the stack as a 32-bit unsigned integer. No overflow checking is performed.

## ENTRY CONDITIONS:

SP ==> 32-bit multiplier  
 32-bit multiplicand  
 32-bit space for result

## EXIT CONDITION DIFFERENT FROM ENTRY:

SP ==> 32-bit product (result from multiplication)

EXAMPLE: Multiply D0 by D1; load result into D2.

[5~

SUBQ.L	#4,SP	Allocate space for result.
MOVE.L	D0,-(SP)	Push multiplicand.
MOVE.L	D1,-(SP)	Push multiplier.
SYSCALL	.MULU32	Multiply D0 by D1.
MOVE.L	(SP)+,D2	Get product.

## 5.2.33 .DIVU32 Function

.DIVU32

TRAP FUNCTION: .DIVU32 - Unsigned 32-bit x 32-bit divide

CODE: \$006A

DESCRIPTION: Unsigned division is performed on two 32-bit integers and the quotient is returned on the stack as a 32-bit unsigned integer. The case of division by zero is handled by returning the maximum unsigned value \$FFFFFFF.

ENTRY CONDITIONS:

SP ==>	32-bit divisor	(value to divide by)
	32-bit dividend	(value to divide)
	32-bit space for result	

EXIT CONDITION DIFFERENT FROM ENTRY:

SP ==&gt; 32-bit quotient (result from division)

EXAMPLE: Divide D0 by D1; load result into D2.

SUBQ.L	#4,SP	Allocate space for result.
MOVE.L	D0,-(SP)	Push dividend.
MOVE.L	D1,-(SP)	Push divisor.
SYSCALL	.DIVU32	Divide D0 by D1.
MOVE.L	(SP)+,D2	Get quotient.

## 5.2.34 .CHK\_SUM Function

.CHK\_SUM

TRAP FUNCTION: .CHK\_SUM - Generate checksum for address range

CODE: \$006B

DESCRIPTION: This function generates a checksum for an address range that is passed in as arguments.

ENTRY CONDITIONS:

SP ==> Beginning address	(longword)
Ending address + 1	(longword)
Space for checksum	(word)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Checksum	(word)
-----------------	--------

EXAMPLE:	CLR.W	-(SP)	Make room for the checksum.
	PEA.L	A1	Push pointer to ending address + 1.
	PEA.L	A0	Push pointer to starting address.
	SYSCALL	.CHK_SUM	Invoke TRAP #15 call.
	MOVE.W	(SP)+,D0	Load D0.W with checksum (EE00)
			MSB=even LSB=odd

- NOTES:
1. If a Bus Error results from this routine, then the bus error exception handler is invoked and the calling routine is also aborted.
  2. The calling routine must insure that the beginning and ending addresses are on word boundaries or the integrity of the checksum cannot be guaranteed.

## 5.2.35 .BRD\_ID Function

.BRD\_ID

TRAP FUNCTION: .BRD\_ID - Return pointer to board ID packet.

CODE: \$0070

DESCRIPTION: This routine returns a pointer on the stack to the 'board identification' packet. The packet is built at initialization time and contains information about the board and the peripherals it supports.

The format of the board identification packet is shown below:

\$00	Eye Catcher			
\$04	Rev	Month	Day	Year
\$08	Packet Size		Reserved	
\$0C	Board Number		Board Suffix	
\$10	Options (coprocessor, etc.)		Family	CPU
\$14	Controller LUN		Device LUN	
\$18	Device Type		Device Number	
\$1C	Bug/SSID Modem Structure Pointer			

Eye Catcher            Long word containing ASCII string "|ID|".

Rev                    Byte contains Bug revision (in BCD).

Month, Day, Year      3 bytes contain date (in BCD) Bug was frozen.

Packet Size            Word contains the size of the packet.

Reserved               Reserved for future use.

Board Number          Word contains the board number (in BCD).

Board Suffix          Word contains the ASCII board suffix (XT, A, 20).

Options:

bits 0-3                Four bits contain CPU type:

                         CPU = 1 ; MC68010 present

                         CPU = 2 ; MC68020 present

                         CPU = 3 ; MC68030 present

                         CPU = 4 ; MC68040 present

bits 4-6            Three bits contain the Family type:  
                     Fam = 0 ; 68xxx family  
                     Fam = 1 ; 88xxx family

bits 7-31          The remaining bits define various coprocessors presently  
                     defined:  
                     Bit 7 set = FPC present  
                     Bit 8 set = PMMU present  
                     Bit 9 set = MMB present

SSID Pointer        Long word contains a pointer to the modem structure for  
                     Bug and SSID.

## ENTRY CONDITIONS:

SP ==> Result <long>            Allocate space for ID packet address.

## EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Address <long>           Starting address of ID packet.

## EXAMPLE:

CLR.L	-(SP)	Make room for returned pointer.
SYSCALL	.BRD_ID	Board ID trap call.
MOVE.L	(SP)+,A0	Get pointer off stack.

THIS PAGE INTENTIONALLY LEFT BLANK.

5

## CHAPTER 6 - 141Bug DIAGNOSTIC FIRMWARE GUIDE

## 6.1 SCOPE

This diagnostic guide contains information about the operation and use of the MVME141 Diagnostic Firmware Package, hereafter referred to as "the diagnostics". Paragraphs 6.3 and 6.4 give the user guidance in setting up the system and invoking the various utilities and tests. 6.5 describes utilities available to the user. 6.6 through 6.14 are guides to using each test.

## 6.2 OVERVIEW OF DIAGNOSTIC FIRMWARE

The MVME141 diagnostic firmware package is part of two 128K x 8 EPROMs which are installed on the MVME141. These two EPROMs (which also contain 141Bug) contain a complete diagnostic monitor along with a battery of utilities and tests for exercise, test, and debug of hardware in the MVME141 environment.

The diagnostics are menu-driven for ease of use. The Help (HE) command displays a menu of all available diagnostic functions (i.e., the tests and utilities). Several tests have a subtest menu which may be called using the HE command. In addition, some utilities have subfunctions, and as such have subfunction menus.

## 6.3 SYSTEM START-UP

Even though the MVME141Bug EPROMs are installed on the MVME141 module, for 141Bug to operate properly with the MVME141, follow this set-up procedure.

**CAUTION**

**INSERTING OR REMOVING MODULES WHILE POWER IS APPLIED COULD DAMAGE MODULE COMPONENTS.**

1. Turn all equipment power OFF. Refer to the MVME141 User's Manual and configure the header jumpers on the module as required for the user's particular application. There are no jumper configurations specifically dictated by 141Bug. The Bug works correctly with all jumpers in the as-shipped factory configuration.
2. Refer to the MVME141 User's Manual and configure header J1 for the user's particular application. J1 enables or disables the system controller function of the MVME141, and also sets the group address of the global CSR for the VME controller chip.

3. Be sure that the two 128K x 8 141Bug EPROMs are installed in sockets U56 (odd bytes) and U78 (even bytes) on the MVME141 module.
4. Refer to the set-up procedure for the user's particular chassis or system for details concerning the installation of the MVME141.
5. Connect the terminal which is to be used as the 141Bug system console to connector J5 (port 1) on the MVME141 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 MVME141 ports at power-up.

After power-up, the baud rate of the J5 port (port 1) can be reconfigured by using the Port Format (PF) command of the 141Bug debugger.

#### NOTE

In order for high-baud rate serial communication between 141Bug and the terminal to work, the terminal must do some handshaking. If the terminal being used does not do hardware handshaking via the CTS line, then it must do XON/XOFF handshaking. If the user gets garbled messages and missing characters, then he should check the terminal to make sure XON/XOFF handshaking is enabled.

6. If it is desired to connect device(s) (such as a host computer system or a serial printer) to port 2, connect the appropriate cable to connector J4 and configure the port as detailed in the MVME141 User's Manual. After power-up, this port can be reconfigured by using the PF command of the 141Bug debugger.
7. Power up the system. 141Bug executes self-checks and displays the debugger prompt "141-Bug>".

If after a delay, the 141Bug begins to display test result messages on the bottom line of the screen in rapid succession, the MVME141 is in the Bug "system" mode. If this is not the desired mode of operation, then press the ABORT switch on the front panel of the MVME141. When the MENU is displayed, enter a 3 to go to the system debugger. The environment may be changed by using the set environment (ENV) command. Refer to the Bug operation in the system mode in this manual.

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

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

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

The firmware monitor comes up with the FAIL LED on.

## 6.4 DIAGNOSTIC MONITOR

The tests described herein are called via a common diagnostic monitor, hereafter called monitor. This monitor is command-line driven and provides input/output facilities, command parsing, error reporting, interrupt handling, and a multi-level directory.

### 6.4.1 Monitor Start-Up

When the monitor is first brought up, following power-up or pushbutton switch RESET, the following is displayed on the diagnostic video display terminal (port 1 terminal):

```
Copyright Motorola Inc. 1988, All Rights Reserved
```

```
VME141 Monitor/Debugger Release 1.0 - x/xx/xx
```

```
COLD Start
141-Bug>
```

If after a delay, the 141Bug begins to display test result messages on the bottom line of the screen in rapid succession, the MVME141 is in the Bug "system" mode. If this is not the desired mode of operation, then press the ABORT switch. When the menu id displayed, enter a 3 to go to the system debugger. The environment may be changed by using the Set Environment (ENV) command. Refer to Appendix A for details of Bug operation in the system mode.

At the prompt, enter SD to switch to the diagnostics directory. The Switch Directories (SD) command is described elsewhere in this chapter. The prompt should now read "141-Diag>".

### 6.4.2 Command Entry and Directories

Entry of commands is made when the prompt "141-Diag>" appears. The name (mnemonic) for the command is entered before pressing the carriage return (CR). Multiple commands may be entered. If a command expects parameters and another command is to follow it, separate the two with an exclamation point (!). For instance, to invoke the command MT B after the command MT A, the command line would read MT A ! MT B. Spaces are not required but are shown here for legibility. Several commands may be combined on one line.

Several commands consist of a command name that is listed in a main (root) directory and a subcommand that is listed in the directory for that particular command. In the main directory are commands like MPU and CA30. These commands are used to refer to a set of lower level commands.

To call up a particular MPU test, enter (on the same line) MPU A. This command causes the monitor to find the MPU subdirectory, and then to execute the command "A" from that subdirectory.

Examples:

Single-Level Commands	HE	Help
	DE	Display Error Counters
Two-Level Commands	MPU	MPU Tests for the MC68030
	A	Register Test
	CA30	MC68030 Onchip Cache Tests
	G	Unlike Instruction Function Codes

### 6.4.3 Help - Command HE

Online documentation has been provided in the form of a Help command (syntax: HE [*command name*]). This command displays a menu of the top level directory if no parameters are entered, or a menu of each subdirectory if the name of that subdirectory is entered. (The top level directory lists (Dir) after the name of each command that has a subdirectory.) For example, to bring up a menu of all the memory tests, enter HE MT. When a menu is too long to fit on the screen, it pauses until the operator presses the carriage return, (CR), again.

### 6.4.4 Self Test - Prefix/Command ST

The monitor provides an automated test mechanism called self test. Entering ST + *command* causes the monitor to run only the tests included in that command. Entering ST - *command* runs all the tests included in an internal self-test directory except the command listed. ST without any parameters runs the entire directory, which contains most of the MVME141 diagnostics.

Each test for that particular command is listed in the paragraph pertaining to the command.

When in "system" mode, and Bug has been invoked, the suite of extended confidence tests that are run at system mode start up can be executed from Bug. This is done with the SST command. This is useful for debugging board failures that may require toggling between the test suite and Bug. Upon completion of running the test suite, the Bug prompt is displayed, ready for other commands. For details on extended confidence test operation, refer to Appendix A, Bug System Mode Operation.

#### 6.4.5 Switch Directories - Command SD

To leave the diagnostic directory (and disable the diagnostic tests), enter SD. At this point, only the commands for 141Bug function. When in the 141Bug directory, the prompt reads 141-Bug>. To return to the diagnostic directory, the command SD is entered again. When in the diagnostic directory, the prompt reads 141-Diag>. The purpose of this feature is to allow the user to access 141Bug without the diagnostics being visible.

#### 6.4.6 Loop-On-Error Mode - Prefix LE

Occasionally, when an oscilloscope or logic analyzer is in use, it becomes desirable to endlessly repeat a test at the point where an error is detected. LE accomplishes that for most of the tests. To invoke LE, enter it before the test that is to run in loop-on-error mode.

#### 6.4.7 Stop-On-Error Mode - Prefix SE

It is sometimes desirable to stop a test or series of tests at the point where an error is detected. SE accomplishes that for most of the tests. To invoke SE, enter it before the test or series of tests that is to run in stop-on-error mode.

#### 6.4.8 Loop-Continue Mode - Prefix LC

To endlessly repeat a test or series of tests, the prefix LC is entered. This loop includes everything on the command line. To break the loop, press the BREAK key on the diagnostic video display terminal. Certain tests disable the BREAK key interrupt, so pressing the ABORT or RESET switches on the MVME141 front panel may become necessary.

#### 6.4.9 Non-Verbose Mode - Prefix NV

Upon detecting an error, the tests included for the MVME141 display a substantial amount of data. To avoid the necessity of watching the scrolling display, a mode is provided that suppresses all messages except PASSED or FAILED. This mode is called non-verbose and is invoked prior to calling a command by entering NV. NV ST MT would cause the monitor to run the MT self-test, but show only the names of the subtests and the results (pass/fail).

#### 6.4.10 Display Error Counters - Command DE

Each test or command in the diagnostic monitor has an individual error counter. As errors are encountered in a particular test, that error counter is incremented. If one were to run a self-test or just a series of tests, the results could be broken down as to which tests passed by examining the error counters. DE displays the results of a particular test if the name of that test follows DE. Only nonzero values are displayed.

#### 6.4.11 Clear (Zero) Error Counters - Command ZE

The error counters originally come up with the value of zero, but it is occasionally desirable to reset them to zero at a later time. This command resets all of the error counters to zero. The error counters can be individually reset by entering the specific test name following the command. Example: ZE MPU A clears the error counter associated with MPU A.

#### 6.4.12 Display Pass Count - Command DP

A count of the number of passes in loop-continue mode is kept by the monitor. This count is displayed with other information at the conclusion of each pass. To display this information without using LC, enter DP.

#### 6.4.13 Zero Pass Count - Command ZP

Invoking this command resets the pass counter DP to zero. This is frequently desirable before typing in a command that invokes the loop-continue mode. Entering this command on the same line as LC results in the pass counter being reset every pass.

### 6.5 UTILITIES

The monitor is supplemented by several utilities that are separate and distinct from the monitor itself and the diagnostics.

### 6.5.1 Write Loop - Command WL.size

The WL.size command invokes a streamlined write of specified size to the specified memory location. This command is intended as a technician aid for debug once specific fault areas are identified. The write loop is very short in execution so that measuring devices such as oscilloscopes may be utilized in tracking failures. Pressing the BREAK key does not stop the command, but pressing the ABORT switch or RESET switch does.

Command size must be specified as B for byte, W for word, or L for longword.

The command requires two parameters: target address and data to be written. The address and data are both hexadecimal values and must be preceded by a \$ if the first digit is other than 0-9 (i.e., \$FF would be entered as \$FF). To write \$00 out to address \$00010000, enter WL.B \$00010000 00. Omission of either or both parameters causes prompting for the missing values.

### 6.5.2 Read Loop - Command RL.size

The RL.size command invokes a streamlined read of specified size from the specified memory location. This command is intended as a technician aid for debug once specific fault areas are identified. The read loop is very short in execution so that measuring devices such as oscilloscopes may be utilized in tracking failures. Pressing the BREAK key does not stop the command, but pressing the ABORT switch or RESET switch does.

Command size must be specified as B for byte, W for word, or L for longword.

The command requires one parameter: target address. The address is a hexadecimal value. To read from address \$00010000, enter RL.B \$00010000. Omission of the parameter causes prompting for the missing value.

### 6.5.3 Write/Read Loop - Command WR.size

The WR.size command invokes a streamlined write and read of specified size to the specified memory location. This command is intended as a technician aid for debug once specific fault areas are identified. The write/read loop is very short in execution so that measuring devices such as oscilloscopes may be utilized in tracking failures. Pressing the BREAK key does not stop the command, but pressing the ABORT switch or RESET switch does.

Command size must be specified as B for byte, W for word, or L for longword.

The command requires two parameters: target address and data to be written. The address and data are both hexadecimal values and must be preceded by a \$ if the first digit is other than 0-9 (i.e., \$FF would be entered as \$FF). To write \$00 out to address \$00010000, enter WR.B \$00010000 00. Omission of either or both parameters causes prompting for the missing values.

## 6.6 MPU TESTS FOR THE MC68030 - Command MPU

MPU

The following paragraphs describe the MPU tests for the MC68030.

## General Description

This paragraph details the diagnostics provided to test the MC68030 MPU, as listed in Table 6-1.

TABLE 6-1. MC68030 MPU Diagnostic Tests

MONITOR COMMAND	TITLE
MPU A	Register Test
MPU B	Instruction Test
MPU C	Address Mode Test
MPU D	Exception Processing Test

The normal procedure for fixing an MC68030 MPU error is to replace the MPU.

## Hardware Configuration

The following hardware is required to perform these tests:

- MVME141 - Module being tested
- VME chassis
- Video display terminal

**6.6.1 MPU A - Register Test****MPU A**

The following paragraphs describe the MPU A register test.

**Description**

This command does a thorough test of all the registers in the MC68030 chip, including checking for bits stuck high or low.

**Command Input**

```
141-Diag> MPU A
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
A   MPU Register test.....Running ----->
```

If any part of the test fails, then the display appears as follows.

```
A   MPU Register test.....Running ----->..... FAILED
(error message)
```

Here, (error message) is one of the following:

```
Failed D0-D7 register check
Failed SR register check
Failed USP/VBR/CAAR register check
Failed CACR register check
Failed A0-A4 register check
Failed A5-A7 register check
```

If all parts of the test are completed correctly, then the test passes.

```
A   MPU Register test.....Running -----> PASSED
```

## 6.6.2 MPU B - Instruction Test

MPU B

The MPU B instruction test is described in the following paragraphs.

**Description**

This command tests various data movement, integer arithmetic, logical, shift and rotate, and bit manipulation instructions of the MC68030 chip.

**Command Input**

```
141-Diag> MPU B
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
B    MPU Instruction Test .....Running ----->
```

If any part of the test fails, then the display appears as follows.

```
B    MPU Instruction Test.....Running ----->..... FAILED
(error message)
```

Here, (error message) is one of the following:

```
Failed AND/OR/NOT/EOR instruction check
Failed DBF instruction check
Failed ADD or SUB instruction check
Failed MULU or DIVU instruction check
Failed BSET or BCLR instruction check
Failed LSR instruction check
Failed LSL instruction check
Failed BFSET or BFCLR instruction check
Failed BFCHG or BFINS instruction check
Failed BFEXTU instruction check
```

If all parts of the test are completed correctly, then the test passes.

```
B    MPU Instruction Test.....Running -----> PASSED
```

## 6.6.3 MPU C - Address Mode Test

MPU C

The MPU C address mode test is described in the following paragraphs.

**Description**

This command tests the various addressing modes of the MC68030 chip. These include absolute address, address indirect, address indirect with postincrement, and address indirect with index modes.

**Command Input**

```
141-Diag> MPU C
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
C    MPU Address Mode test.....Running ----->
```

If any part of the test fails, then the display appears as follows.

```
C    MPU Address Mode test.....Running ----->..... FAILED
(error message)
```

Here, (error message) is one of the following:

```
Failed Absolute Addressing check
Failed Indirect Addressing check
Failed Post increment check
Failed Pre decrement check
Failed Indirect Addressing with Index check
Unexpected Bus Error at $XXXXXXXX
```

If all parts of the test are completed correctly, then the test passes.

```
C    MPU Address Mode test.....Running -----> PASSED
```

## 6.6.4 MPU D - Exception Processing Test

MPU D

The MPU D exception processing test is described in the following paragraphs.

**Description**

This command tests many of the exception processing routines of the MC68030, but not the interrupt auto vectors or any of the floating point coprocessor vectors.

**Command Input**

```
141-Diag> MPU D
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
D    MPU Exception Processing Test.....Running ----->
```

If any part of the test fails, then the display appears as follows.

```
D    MPU Exception Processing Test.....Running ----->..... FAILED
Test Failed Vector # XXX
```

Here # XXX is the hexadecimal exception vector offset, as explained in the MC68030 User's Manual, MC68030UM.

However, if the failure involves taking an exception different from that being tested, the display is:

```
D    MPU Exception Processing Test.....Running ----->..... FAILED
Unexpected exception taken to Vector # XXX
```

If all parts of the test are completed correctly, then the test passes.

```
D    MPU Exception Processing Test.....Running -----> PASSED
```

## 6.7 MC68030 ONCHIP CACHE TESTS - Command CA30

The MC68030 onchip cache tests are described in the following paragraphs.

### General Description

This paragraph details the diagnostics provided to test the MC68030 cache, as listed in Table 6-2.

TABLE 6-2. MC68030 Cache Diagnostic Tests

MONITOR COMMAND	TITLE
CA30 A	Basic data caching test
CA30 B	D cache tag RAM test
CA30 C	D cache data RAM test
CA30 D	D cache valid flags test
CA30 E	D cache burst fill test
CA30 F	Basic instruction caching test
CA30 G	Unlike instruction function codes test
CA30 H	I cache disable test
CA30 I	I cache clear test

The normal procedure for fixing an MC68030 cache error is to replace the MPU.

### Hardware Configuration

The following hardware is required to perform these tests:

MVME141 - Module being tested  
 VME chassis  
 Video display terminal.

## 6.7.1 CA30 A - Basic Data Caching Test

CA30 A

The CA30 A basic data caching test is described in the following paragraphs.

**Description**

This test checks out the basic caching function by deliberately causing stale data, then reading the corresponding locations. Failure is declared if the cache misses or if any value is read that is not what is expected to be in the cache.

The test is meant only to provide a gross functional check of the cache. Its purpose is to verify that each entry in the cache latches and holds data independently of the other entries. It does not check for bad bits in the tag RAM, valid flags, or data RAM other than what is required to cache a simple pattern.

**Command Input**

141-Diag> CA30 A

**Response/Messages**

After the command has been issued, the following line is printed:

```
A   Basic data caching .....Running ----->
```

If there are any cache misses, then the test fails and the display appears as follows.

```
A   Basic data caching .....Running ----->
CACHE MISSED! ..... FAILED
```

If there are no cache misses, then the test passes.

```
A   Basic data caching .....Running -----> PASSED
```

**6.7.2 CA30 B - Data Cache Tag RAM Test****CA30 B**

The CA30 B data cache tag RAM test is described in the following paragraphs.

**Description**

This test verifies the data cache tag RAM by caching accesses to locations that cause a variety of values to be written into the tag RAM. The test addresses and function codes are translated by the onchip MMU to select locations physically in the onboard RAM. The criterion for passing the test is hitting in the cache on a read from a test location following a cacheable write access to that same location. The data in the cache has been made stale between the cacheable write and the following read to provide the distinction between hits and misses. The failure of the tag RAM to properly latch a tag should cause the cache to miss when it should hit.

The MMU is used to allow a wide variety of values to be used for the tags without having read/write memory at each location corresponding to those tags. This relies on the cache being logical as opposed to physical.

To allow the test to be run either out of ROM or RAM, the onboard resources and the bottom 16Mb of the addressing space are mapped transparently for supervisor mode accesses. This allows debugging of the test with full access to the monitor/debugger facilities, too. This requires that the test addresses that fall in these two ranges not bear supervisor function codes.

The translation of the test addresses and function codes is implemented by setting the CRP descriptor type to "early termination" and loading an offset into the CRP table address field. This offset is the distance from the test address to a location in the test area and is added to every logical address that does not qualify for transparent translation (refer to following paragraph).

The 16Mb transparently mapped areas are described by the Transparent Translation (TT) registers. TT0 matches addresses \$00XXXXXX while TT1 matches addresses \$FFXXXXXX. Both are qualified such that the function code must specify supervisor mode. The purpose in this is to force test addresses that fall in either range to be offset instead of transparently mapped. The test addresses have been selected to avoid matching both address and function code with either TT register.

**Command Input**

141-Diag> CA30 B

**Response/Messages**

After the command has been issued, the following line is printed:

B D cache tag RAM .....Running ----->

If there are any cache misses, then the test fails and the display appears as follows.

B D cache tag RAM .....Running ----->  
CACHE MISSED! FAILED

If there are no cache misses, then the test passes.

B D cache tag RAM .....Running -----> PASSED

6

## 6.7.3 CA30 C - Data Cache Data RAM Test

CA30 C

The CA30 C data cache data RAM test is described in the following paragraphs.

**Description**

This is essentially a memory test. For each entry in the cache, several values are cached, made stale, then read back. Failure is declared if the value written (and supposedly cached) differs from that read later from the same location. No distinction is made between cache misses and genuine bad data; the only concern here is the latching of the data.

**Command Input**

```
141-Diag> CA30 C
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
C    D cache data RAM test .....Running ----->
```

If there are any cache misses, then the test fails and the display appears as follows.

```
C    D cache data RAM test .....Running ----->
CACHE MISSED!      FAILED
```

If there are no cache misses, then the test passes.

```
C    D cache data RAM test .....Running -----> PASSED
```

## 6.7.4 CA30 D - Data Cache Valid Flags Test

CA30 D

The CA30 D data cache valid flags test is described in the following paragraphs.

**Description**

This test verifies that each valid flag is set when its entry is valid and cleared either when the entire cache is flushed or an individual line is cleared. This test does check for side effects on other valid flags.

This test is actually two inline subtests: V FLAG CLEAR and V FLAG SET. The former checks that each valid flag can be individually cleared while the latter checks for the opposite.

**Command Input**

```
141-Diag> CA30 D
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
D    D cache valid flags test .....Running ----->
```

If there are any cache hits when clearing valid flags, then the test fails and the display appears as follows:

```
D    D cache valid flags test .....Running ----->
VALID BIT CLEAR SUBTEST - EXPECTED MISS ..... FAILED
```

If there are any cache misses when setting valid flags, then the test fails and the display appears as follows:

```
D    D cache valid flags test .....Running ----->
VALID BIT SET  SUBTEST - EXPECTED HIT ..... FAILED
```

If all flags are valid, then the test passes.

```
D    D cache valid flags test .....Running -----> PASSED
```

## 6.7.5 CA30 E - Data Cache Burst Fill Test

CA30 E

The CA30 E data cache burst fill test is described in the following paragraphs.

**Description**

This test checks the burst fill mechanism. The criterion for passing this test is hitting in the cache for all entries in a line after one entry in that line has been cached. Each line is tested. The entry read to cause the filling is varied from line to line.

**Command Input**

```
141-Diag> CA30 E
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
E   D cache burst fill test .....Running ----->
```

If there are any cache misses, then the test fails and the display appears as follows.

```
E   D cache burst fill test .....Running ----->
CACHE MISSED!      FAILED
```

If there are no cache misses, then the test passes.

```
E   D cache burst fill test .....Running -----> PASSED
```

## 6.7.6 CA30 F - Basic Instruction Caching Test

CA30 F

The CA30 F basic instruction caching test is described in the following paragraphs.

**Description**

This command tests the basic caching function of the MC68030 microprocessor. The test caches a program segment that resides in RAM, freezes the cache, changes the program segment in RAM, then reruns the program segment. If the cache is functioning correctly, the cached instructions are executed. Failure is detected if the MC68030 executes the instructions that reside in RAM; any cache misses cause an error.

The process is first attempted in supervisor mode for both the initial pass through the program segment and the second pass. It is then repeated, using user mode for the initial pass and the second pass. A bit is included in each cache entry for distinguishing between supervisor and user mode. If this bit is stuck or inaccessible, the cache misses during one of these two tests.

**Command Input**

```
141-Diag> CA30 F
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
F   Basic instr. caching .....Running ----->
```

If there are any cache misses during the second pass through the program segment, then the test fails and the display appears as follows.

```
F   Basic instr. caching .....Running ----->..... FAILED
2 CACHE MISSES!
CACHED IN SUPY MODE, RERAN IN SUPY MODE
```

If there are no cache misses during the second pass, then the test passes.

```
F   Basic instr. caching .....Running -----> PASSED
```

## 6.7.7 CA30 G - Unlike Instruction Function Codes Test

CA30 G

The CA30 G unlike instruction function codes test is described in the following paragraphs.

**Description**

This command tests the ability of the onchip cache to recognize instruction function codes. Bit 2 of the function code is included in the tag for each entry. This provides a distinction between supervisor and user modes for the cached instructions. To test this mechanism, a program segment that resides in RAM is cached in supervisor mode. The cache is frozen, then the program segment in RAM is changed. When the program segment is executed a second time in user mode, there should be no cache hits due to the different function codes. Failure is detected if the MC68030 executes the cached instructions.

After the program segment has been cached in supervisor mode and rerun in user mode, the process is repeated, caching in user mode and rerunning in supervisor mode. Again, the cache should miss during the second pass through the program segment.

**Command Input**

```
141-Diag> CA30 G
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
G Unlike instr. fn. codes .....Running ----->
```

If there are any cache hits during the second pass through the program segment, then the test fails and the display appears as follows.

```
G Unlike instr. fn. codes .....Running ----->..... FAILED
5 CACHE HITS!
CACHED IN SUPY MODE, RERAN IN USER MODE
```

If there are no cache hits during the second pass, then the test passes.

```
G Unlike instr. fn. codes .....Running -----> PASSED
```

## 6.7.8 CA30 H - Disable Test

CA30 H

The CA30 H disable test is described in the following paragraphs.

**Description**

In the MC68030 Cache Control Register (CACR) a control bit is provided to enable the cache. When this bit is clear, the cache should never hit, regardless of whether the address and function codes match a tag. To test this mechanism, a program segment is cached from RAM. The cache is frozen to preserve its contents, then the enable bit is cleared. The program segment in RAM is then changed and rerun. There should be no cache hits with the enable bit clear. Failure is declared if the cache does hit.

**Command Input**

141-Diag> CA30 H

**Response/Messages**

After the command has been issued, the following line is printed:

```
H    I cache disable test .....Running ----->
```

If there are any cache hits during the second pass through the program segment, then the test fails and the display appears as follows.

```
H    I cache disable test .....Running ----->..... FAILED
I CACHE HIT!
CACHED IN SUPY MODE, RERAN IN SUPY MODE
```

If there are no cache hits during the second pass, then the test passes.

```
H    I cache disable test .....Running -----> PASSED
```

## 6.7.9 CA30 I - Clear Test

CA30 I

The CA30 I clear test is described in the following paragraphs.

**Description**

A control bit is included in the MC68030 CACR to clear the cache. Writing a one to this bit invalidates every entry in the onchip cache. To test this function, a program segment in RAM is cached and then frozen there to preserve it long enough to activate the cache clear control bit. The program segment in RAM is then modified and rerun with the cache enabled. If the cache hits, the clear is incomplete and failure is declared.

**Command Input**

```
141-Diag> CA30 I
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
I   I cache clear test .....Running ----->
```

If there are any cache hits during the second pass through the program segment, then the test fails and the display appears as follows.

```
I   I cache clear test .....Running ----->..... FAILED
58 CACHE HITS!
CACHED IN SUPY MODE, RERAN IN SUPY MODE
```

If there are no cache hits during the second pass, then the test passes.

```
I   I cache clear test .....Running -----> PASSED
```

## 6.8 MEMORY TESTS - Command MT

MT

The memory tests are described in the following paragraphs.

## General Description

This set of tests accesses random access memory (read/write) that may or may not reside on the MVME141 module. Default is the onboard RAM. To test offboard RAM, change Start and Stop Addresses per MT B and MT C as described in the following paragraphs. Memory tests are listed in Table 6-3.

**NOTE**

If one or more memory tests are attempted at an address where there is no memory, a bus error message appears, giving the details of the problem.

TABLE 6-3. Memory Diagnostic Tests

MONITOR COMMAND	TITLE
MT FP	MEM Bd: Fast Pattern Test
MT FA	MEM Bd: Fast Addr. Test
MT FV	MEM Bd: Fast VMEbus W/R Test
MT A	Set Function Code
MT B	Set Start Address
MT C	Set Stop Address
MT D	Set Bus Data Width
MT E	March Address Test
MT F	Walk a Bit Test
MT G	Refresh Test
MT H	Random Byte Test
MT I	Program Test
MT J	TAS Test
MT K	Brief Parity Test
MT L	Extended Parity Test

## Hardware Configuration

The following hardware is required to perform these tests.

- MVME141 - Module being tested
- VME chassis
- Video display terminal
- Optional offboard memory.

## 6.8.1 MT A - Set Function Code

MT A

The set function code command (MT A) is described in the following paragraphs.

**Description**

This command allows the user to select the function code used in most of the memory tests. The exceptions to this are Program Test and TAS Test.

**Command Input**

```
141-Diag> MT A [new value]
```

**Response/Messages**

If the user supplied the optional new value, then the display appears as follows:

```
141-Diag> MT A [new value]  
Function Code=new value  
141-Diag>
```

If a new value was not specified by the user, then the old value is displayed and the user is allowed to enter a new value.

**NOTE**

The default is Function Code=5, which is for onboard RAM.

```
141-Diag> MT A  
Function Code=current value ?[new value]  
Function Code=new value  
141-Diag>
```

This command may be used to display the current value without changing it by pressing a carriage return (CR) without entering the new value.

```
141-Diag> MT A  
Function Code=current value ?(CR)  
Function Code=current value  
141-Diag>
```

## 6.8.2 MT B - Set Start Address

MT B

The set start address command (MT B) is described in the following paragraphs.

## Description

This command allows the user to select the start address used by all of the memory tests. For the MVME141, it is suggested that address \$00004000 be used. Other addresses may be used, but extreme caution should be used when attempting to test memory below this address.

## Command Input

```
141-Diag> MT B [new value]
```

## Response/Messages

If the user supplied the optional new value, then the display appears as follows:

```
141-Diag> MT B [new value]  
Start Addr.=new value  
141-Diag>
```

If a new value was not specified by the user, then the old value is displayed and the user is allowed to enter a new value.

**NOTE**

The default is Start Addr.=00004000, which is for onboard RAM.

```
141-Diag> MT B  
Start Addr.=current value ?[new value]  
Start Addr.=new value  
141-Diag>
```

This command may be used to display the current value without changing it by pressing a carriage return (CR) without entering the new value.

```
141-Diag> MT B  
Start Addr.=current value ?(CR)  
Start Addr.=current value  
141-Diag>
```

**NOTE**

If a new value is specified, it is truncated to a longword boundary and, if greater than the value of the stop address, replaces the stop address. The start address is never allowed to be higher in memory than the stop address. These changes occur before another command is processed by the monitor.

## 6.8.3 MT C - Set Stop Address

MT C

The set stop address command (MT C) is described in the following paragraphs.

**Description**

This command allows the user to select the stop address used by all of the memory tests.

**Command Input**

```
141-Diag> MT C [new value]
```

**Response/Messages**

If the user supplied the optional new value, then the display appears as follows:

```
141-Diag> MT C [new value]  
Stop Addr.=new value  
141-Diag>
```

If a new value was not specified by the user, then the old value is displayed and the user is allowed to enter a new value.

**NOTE**

The default is Stop Addr.=DRAMsize-4, which is the end of onboard RAM.

```
141-Diag> MT C  
Stop Addr.=current value ?[new value]  
Stop Addr.=new value  
141-Diag>
```

This command may be used to display the current value without changing it by pressing a carriage return <CR> without entering the new value.

```
141-Diag> MT C  
Start Addr.=current value ?(CR)  
Start Addr.=current value  
141-Diag>
```

**NOTE**

If a new value is specified, it is truncated to a longword boundary and, if less than the value of the start address, is replaced by the start address. The stop address is never allowed to be lower in memory than the start address. These changes occur before another command is processed by the monitor.

## 6.8.4 MT D - Set Bus Data Width

MT D

The set bus data width command (MT D) is described in the following paragraphs.

## Description

This command is used to select either 16-bit or 32-bit bus data accesses during the MVME141Bug MT memory tests. The width is selected by entering zero for 16 bits or one for 32 bits.

## Command Input

141-Diag> MT D [*new value: 0 for 16, 1 for 32*]

## Response/Messages

If the user supplied the optional new value, then the display appears as follows:

```
141-Diag> MT D [new value]
Bus Width (32=1/16=0) =new value
141-Diag>
```

If a new value was not specified by the user, then the old value is displayed and the user is allowed to enter a new value.

NOTES: 1. The default value is Bus Width (32=1/16=0) =0.

```
141-Diag> MT D
Bus Width (32=1/16=0) =current value ?[new value]
Bus Width (32=1/16=0) =new value
141-Diag>
```

This command may be used to display the current value without changing it by pressing a carriage return (CR) without entering the new value.

```
141-Diag> MT D
Bus Width (32=1/16=0) =current value ?(CR)
Bus Width (32=1/16=0) =current value
141-Diag>
```

**6.8.5 MT E - March Address Test****MT E**

The march address test command (MT E) is described in the following paragraphs.

**Description**

This command performs a march address test from Start Address to Stop Address.

The march address test has been implemented in the following manner:

- Step 1. All memory locations from Start Address up to Stop Address are cleared to 0.
- Step 2. Beginning at Stop Address and proceeding downward to Start Address, each memory location is checked for bits that did not clear and then the contents are changed to all F's (all the bits are set). This process reveals address lines that are stuck high.
- Step 3. Beginning at Start Address and proceeding upward to Stop Address, each memory location is checked for bits that did not set and then the memory location is again cleared to 0. This process reveals address lines that are stuck low.

**Command Input**

141-Diag> MT E

**Response/Messages**

After the command is entered, the display should appear as follows:

```
E   MT March Addr. Test.....Running ----->
```

If an error is encountered, then the memory location and other related information are displayed.

```
E   MT March Addr. Test.....Running ----->..... FAILED
(error-related information)
```

If no errors are encountered, then the display appears as follows:

```
E   MT March Addr. Test.....Running -----> PASSED
```

## 6.8.6 MT F - Walk a Bit Test

MT F

The walk a bit test command (MT F) is described in the following paragraphs.

**Description**

This command performs a walking bit test from start address to stop address.

The walking bit test has been implemented in the following manner:

Step 1. For each memory location, do the following:

Write out a 32-bit value with only the lower bit set.

Read it back and verify that the value written equals the one read. Report any errors.

Shift the 32-bit value to move the bit up one position.

Repeat the procedure (write, read, and verify) for all 32-bit positions.

**Command Input**

```
141-Diag> MT F
```

**Response/Messages**

After the command is entered, the display should appear as follows:

```
F    MT Walk a bit Test .....Running ----->
```

If an error is encountered, then the memory location and other related information are displayed.

```
F    MT Walk a bit Test .....Running ----->..... FAILED
(error-related information)
```

If no errors are encountered, then the display appears as follows:

```
F    MT Walk a bit Test .....Running -----> PASSED
```

## 6.8.7 MT G - Refresh Test

MT G

The refresh test command (MT G) is described in the following paragraphs.

**Description**

This command performs a refresh test from Start Address to Stop Address.

The refresh test has been implemented in the following manner:

Step 1. For each memory location:

Write out value \$FC84B730.

Verify that the location contains \$FC84B730.

Proceed to next memory location.

Step 2. Delay for 500 milliseconds (1/2 second).

Step 3. For each memory location:

Verify that the location contains \$FC84B730.

Write out the complement of \$FC84B730 (\$037B48CF).

Verify that the location contains \$037B48CF.

Proceed to next memory location.

Step 4. Delay for 500 milliseconds.

Step 5. For each memory location:

Verify that the location contains \$037B48CF.

Write out value \$FC84B730.

Verify that the location contains \$FC84B730.

Proceed to next memory location.

**Command Input**

141-Diag> MT G

**Response/Messages**

After the command is entered, the display should appear as follows:

G MT Refresh Test.....Running ----->

If an error is encountered, then the memory location and other related information are displayed.

G MT Refresh Test.....Running ----->..... FAILED  
(error-related information)

If no errors are encountered, then the display appears as follows:

G MT Refresh Test.....Running -----> PASSED

**6.8.8 MT H - Random Byte Test****MT H**

The random byte test command (MT H) is described in the following paragraphs.

**Description**

This command performs a random byte test from Start Address to Stop Address.

The random byte test has been implemented in the following manner:

Step 1. A register is loaded with the value \$ECA86420.

Step 2. For each memory location:

Copy the contents of the register to the memory location, one byte at a time.

Add \$02468ACE to the contents of the register.

Proceed to next memory location.

Step 3. Reload \$ECA86420 into the register.

Step 4. For each memory location:

Compare the contents of the memory to the register to verify that the contents are good, one byte at a time.

Add \$02468ACE to the contents of the register.

Proceed to next memory location.

**Command Input**

```
141-Diag> MT H
```

**Response/Messages**

After the command is entered, the display should appear as follows:

```
H   MT Random Byte Test.....Running ----->
```

If an error occurs, then the memory location and other related information are displayed.

H MT Random Byte Test.....Running ----->.... FAILED  
(error-related information)

If no errors occur, then the display appears as follows:

H MT Random Byte Test.....Running -----> PASSED

6

## 6.8.9 MT I - Program Test

MT I

The program test command (MT I) is described in the following paragraphs.

**Description**

This command moves a program segment into RAM and executes it. The implementation of this is as follows:

- Step 1. The program is moved into the RAM, repeating it as many times as necessary to fill the available RAM (i.e., from Start Address to Stop Address-8). Only complete segments of the program are moved. The space remaining from the last program segment copied into the RAM to Stop Address-8 is filled with NOP instructions. Attempting to run this test without sufficient memory (around 400 bytes) for at least one complete program segment to be copied causes an error message to be printed out: INSUFFICIENT MEMORY.
- Step 2. The last location, Stop Address, receives an RTS instruction.
- Step 3. Finally, the test performs a JSR to location Start Address.
- Step 4. The program itself performs a wide variety of operations, with the results frequently checked and a count of the errors maintained. Errant locations are reported in the same fashion as any memory test failure (refer to paragraph 6.8.13).

**Command Input**

141-Diag> MT I

**Response/Messages**

After the command is entered, the display should appear as follows:

```
I    MT Program Test.....Running ----->
```

If the operator has not allowed enough memory for at least one program segment to be copied into the target RAM, then the following error message is printed. To avoid this, make sure that the Stop Address is at least 388 bytes (\$00000184) greater than the Start Address.

```
I    MT Program Test.....Running ----->
      Insufficient Memory
      PASSED
```

If the program (in RAM) detects any errors, then the location of the error and other information is displayed.

I MT Program Test.....Running ----->..... FAILED  
(error-related information)

If no errors occur, then the display appears as follows:

I MT Program Test.....Running -----> PASSED

5

## 6.8.10 MT J - TAS Test

MT J

The TAS test command (MT J) is described in the following paragraphs.

**Description**

This command performs a Test and Set (TAS) test from Start Address to Stop Address.

The test is implemented as follows:

Step 1. For each memory location:

Clear the memory location to 0.

Test And Set the location (should set upper bit only).

Verify that the location now contains \$80.

Proceed to next location (next byte).

**Command Input**

```
141-Diag> MT J
```

**Response/Messages**

After the command is entered, the display should appear as follows:

```
J   MT TAS Test.....Running ----->
```

If an error occurs, then the memory location and other related information are displayed.

```
J   MT TAS Test.....Running ----->..... FAILED
(error-related information)
```

If no errors occur, then the display appears as follows:

```
J   MT TAS Test.....Running -----> PASSED
```

## 6.8.11 MT K - Brief Parity Test

MT K

The brief parity test command (MT K) is described in the following paragraphs.

**Description**

Test parity checking ability on longwords only from Start Address to Stop Address.

The brief parity test is implemented in the following manner:

Step 1. For each longword memory location:

Copy the contents of the memory location to a register, without parity enabled.

Enable parity checking and incorrect party generation.

Copy the contents of the register to the memory location, generating incorrect parity.

Disable incorrect parity generation.

Copy the contents of the memory location to a second register, generating a parity error.

Copy the contents of the register to the memory location, generating correct parity.

Copy the contents of the memory location to a second register, no parity error should occur.

Proceed to next memory location.

Step 2. Disable parity checking.

**Command Input**

141-Diag> MT K

**Response/Messages**

After the command is entered, the display should appear as follows:

K MT Brief parity test .....Running ----->

If an error occurs, then the memory location and other related information are displayed.

K MT Brief parity test .....Running ----->..... FAILED  
(error-related information)

If no errors occur, then the display appears as follows:

K MT Brief parity test .....Running -----> PASSED

## 6.8.12 MT L - Extended Parity Test

MT L

The extended parity test command (MT L) is described in the following paragraphs.

**Description**

Test parity checking ability on byte boundaries from Start Address to Stop Address.

The extended parity test is implemented in the following manner:

## Step 1. For each byte memory location:

Copy the contents of the memory location to a register, without parity enabled.

Enable parity checking and incorrect parity generation.

Copy the contents of the register to the memory location, generating incorrect parity.

Disable incorrect parity generation.

Copy the contents of the memory location to a second register, generating a parity error.

Copy the contents of the register to the memory location, generating correct parity.

Copy the contents of the memory location to a second register, no parity error should occur.

Proceed to next memory location.

## Step 2. Disable parity checking.

**Command Input**

```
141-Diag> MT L
```

**Response/Messages**

After the command is entered, the display should appear as follows:

```
L   MT Extended parity test .....Running ----->
```

If an error occurs, then the memory location and other related information are displayed.

```
L   MT Extended parity test .....Running ----->..... FAILED
(error-related information)
```

If no errors occur, then the display appears as follows:

```
L   MT Extended parity test .....Running -----> PASSED
```

### 6.8.13 Description of Memory Error Display Format

This paragraph is included to describe the format used to display errors during memory test E through L.

The error reporting code is designed to conform to two rules:

- a. The first time an error occurs, headings are printed out prior to the printing of the values.
- b. Upon 20 memory errors, the printing of error messages ceases for the remainder of the test.

The following is an example of the display format:

FC	TEST ADDR	10987654321098765432109876543210	EXPECTED	READ
5	00010000	-----X-----	00000100	00000000
5	00010004	-----X-----X----	FFFFFFF	FFFFFFEF

Each line displayed consists of five items: function code, test address, graphic bit report, expected data, and read data. The test address, expected data, and read data are displayed in hexadecimal. The graphic bit report shows a letter X at each errant bit position and a dash (-) at each good bit position.

The heading used for the graphic bit report is intended to make the bit position easy to determine. Each numeral in the heading is the one's digit of the bit position. For example, the leftmost bad bit at test address \$10004 has the numeral 2 over it. Because this is the second 2 from the right, the bit position is read 12 in decimal (base 10).

## 6.9 MEMORY MANAGEMENT UNIT TESTS - Command MMU

MMU

The memory management unit tests are described in the following paragraphs.

## General Description

This paragraph details the diagnostics provided to test the memory management hardware in the system. The tests are listed in Table 6-4.

TABLE 6-4. Memory Management Unit Diagnostic Tests

MONITOR COMMAND	TITLE
MMU A	RP Register
MMU B	TC Register
MMU C	Super_Prog Space
MMU D	Super_Data Space
MMU E	Write/Mapped-Read Pages
MMU F	Read Mapped ROM
MMU G	Fully Filled ATC
MMU H	User_Data Space
MMU I	User_Prog Space
MMU J	Indirect Page
MMU K	Page-Desc Used-Bit
MMU L	Page-Desc Modify-Bit
MMU M	Segment-Desc Used-Bit
MMU P	Invalid Page
MMU Q	Invalid Segment
MMU R	Write-Protect Page
MMU S	Write-Protect Segment
MMU V	Upper-Limit Violation
MMU W	Lower-Limit Violation
MMU X	Prefetch On Invalid-Page Boundary
MMU Y	Modify-Bit and Index
MMU Z 0, 1, or 2	Sixteen-Bit Bus
MMU O	Read/Modify/Write Cycle

## Hardware Configuration

The following hardware is required to perform these tests.

MVME141 - module being tested  
 VME chassis  
 Video display terminal

## 6.9.1 MMU A - RP Register

MMU A

The root pointer (RP) register command (MMU A) is described in the following paragraphs.

**Description**

This command tests the RP register by doing a walking bit through it.

**Command Input**

```
141-Diag> MMU A
```

**Response/Messages**

After entering this command, the display should read as follows:

```
A   RP Register .....Running ----->
```

If the root pointer fails to latch correctly, then the test fails and the following error message is printed out:

```
A   RP Register .....Running ----->
    Expect=00000010   Read=FFFFFFFF
    .... FAILED
```

If the walk-a-bit test is successful, then the root pointer register test passes.

```
A   RP Register .....Running -----> PASSED
```

6.9.2 MMU B - TC Register

MMU B

The translation control (TC) register test command (MMU B) is described in the following paragraphs.

Description

This command tests the TC register by attempting to clear and then set the Initial Shift (IS) bit.

Command Input

141-Diag> MMU B

Response/Messages

After entering this command, the display should read as follows:

B TC Register .....Running ----->

If the bit cannot be cleared and set, then the test fails.

B TC Register .....Running ----->  
Expect=00008010 Read=00000000  
.... FAILED

If the bit gets cleared and set, then the test passes.

B TC Register .....Running -----> PASSED

6

## 6.9.3 MMU C - Super\_Prog Space

MMU C

The super\_prog space test command (MMU C) is described in the following paragraphs.

**Description**

This command enables the MMU and lets it do a table walk in supervisor program space. The test is implemented in the following manner:

- Step 1. Put the function code in the MC68030 DFC register.
- Step 2. Load the address of function code table into the root pointer register.
- Step 3. Set the E bit in the TC register.
- Step 4. Let the MMU do a table walk for the next instruction, a NOP.
- Step 5. Clear the E bit in the TC register.

**Command Input**

```
141-Diag> MMU C
```

**Response/Messages**

After entering this command, the display should read as follows:

```
C   Super_Prog Space .....Running ----->
```

If a bus error occurs, then the test fails.

```
C   Super_Prog Space .....Running ----->
(bus error: CPU registers dumped to screen here)
.... FAILED
```

If the table walk does not cause a bus error, then the test passes.

```
C   Super_Prog Space .....Running -----> PASSED
```

## 6.9.4 MMU D - Super\_Data Space

## MMU D

The super\_data space test command (MMU D) is described in the following paragraphs.

## Description

This command enables the MMU and lets it do a table walk twice in supervisor program space, then once in supervisor data space. The two walks in supervisor program space are necessary to fetch the instruction that accesses the supervisor data space and prefetch the next one. The test is implemented in the following manner:

- Step 1. Put the function code in the MC68030 DFC register.
- Step 2. Load the address of function code table into the root pointer register.
- Step 3. Set the E bit in the TC register.
- Step 4. Let the MMU do a table walk for the next instruction, which causes the access to supervisor data space via a read (TST.B). Prefetching causes access to the next location, in supervisor program space.
- Step 5. Clear the E bit in the TC register.

## Command Input

141-Diag> MMU D

## Response/Messages

After entering this command, the display should read as follows:

```
D Super_Data Space .....Running ----->
```

If a bus error occurs, then the test fails.

```
D Super_Data Space .....Running ----->
(bus error: CPU registers dumped to screen here)
.... FAILED
```

If the table walk does not cause a bus error(s), then the test passes.

```
D Super_Data Space .....Running -----> PASSED
```

## 6.9.5 MMU E - Write/Mapped-Read Pages

MMU E

The write/mapped-read pages test command (MMU E) is described in the following paragraphs.

**Description**

This command is a test that writes data with the MMU disabled, then verifies that the data can be read with the MMU enabled. The test is implemented in the following manner:

Step 1. Fill two pages with a pattern.

Step 2. Enable the MMU.

Step 3. Check RAM where the two pages were written. Report all discrepancies.

**Command Input**

```
141-Diag> MMU E
```

**Response/Messages**

After entering this command, the display should read as follows:

```
E   Write/Mapped-Read Pages .....Running ----->
```

If a bus error occurs or data read does not equal that expected, then the test fails. A bus error generates a dump of the MC68030 MPU registers' contents. Data corruption generates a message that indicates where the error occurred, then a map of the table walk is displayed.

```
E   Write/Mapped-Read Pages .....Running ----->
      Addr=00000000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the reading of the two pages does not generate a bus error, and the patterns read match the expected, then the test passes.

```
E   Write/Mapped-Read Pages .....Running -----> PASSED
```

## 6.9.6 MMU F - Read Mapped ROM

MMU F

The read mapped ROM test command (MMU F) is described in the following paragraphs.

**Description**

This command tests some of the upper MMU address lines by attempting to access the ROM. Both supervisor program and supervisor data function codes are used to test two separate paths through the translation table. The test is implemented in the following manner:

- Step 1. Set up a pointer to the ROM that is PC relative. All PC relative accesses use supervisor program space and are software transparent.
- Step 2. Set up a pointer to the ROM that uses virtual addressing. Accesses using this pointer are to supervisor data space.
- Step 3. Enable the MMU.
- Step 4. For each location in the ROM, read the ROM via both pointers. The data read should be identical.

**NOTE**

The table walking for the supervisor data space takes a much different path than that for supervisor program space.

If the data does not match, then the test fails. Display the physical address, the expected, and read data.

- Step 5. Disable the MMU.

**Command Input**

```
141-Diag> MMU F
```

**Response/Messages**

After entering this command, the display should read as follows:

```
F   Read Mapped ROM .....Running ----->
```

If the data read via the two pointers ever differs, then the test fails.

```
F   Read Mapped ROM .....Running ----->
    Addr=00100000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the test is able to read every ROM location via both paths, then it passes.

```
F   Read Mapped ROM .....Running -----> PASSED
```

## 6.9.7 MMU G - Fully Filled ATC

MMU G

The fully filled address translation cache (ATC) test command (MMU G) is described in the following paragraphs.

**Description**

This command tests the ATC by verifying that all entries in the translation cache can hold a page descriptor.

For the MMU, this is done by filling the ATC with locked descriptors, and then verifying that each descriptor is resident in the cache. This is implemented as follows:

- Step 1. The lock bit is set in the first 63 page descriptors.
- Step 2. The first word in each of those pages is read, creating an entry for each page in the ATC.
- Step 3. The Lock Warning (LW) bit in the PCSR register is checked, and if it is not set, an error is flagged.
- Step 4. The MMU PTEST instruction is used to verify that the page descriptors for each of the 63 pages reside in the ATC.

**Command Input**

```
141-Diag> MMU G
```

**Response/Messages**

After entering this command, the display should read as follows:

```
G Fully Filled ATC .....Running ----->
```

If a word in the list does not match the corresponding word at the beginning of a page, then the test fails.

```
G Fully Filled ATC .....Running ----->
  Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here
.... FAILED
```

If every word in the list matches the first word of each page, then the test passes.

```
G Fully Filled ATC .....Running -----> PASSED
```

## 6.9.8 MMU H - User Data Space

MMU H

The user data space test command (MMU H) is described in the following paragraphs.

**Description**

This command tests the function code signal lines connecting into the MMU by accessing user data space. This causes the MMU to read the function code and do a table walk as a part of its translation. The test is implemented in the following manner:

- Step 1. Write a pattern out to an area that is mapped to user data space for diagnostic purposes.
- Step 2. Enable the MMU.
- Step 3. Read the area where the pattern was written to, using the function code for user data space. The test fails if the pattern does not match that written out.

**Command Input**

```
141-Diag> MMU H
```

**Response/Messages**

After entering this command, the display should read as follows:

```
H   User_Data Space .....Running ----->
```

If the pattern written out does not match that read, the test fails.

```
H   User_Data Space .....Running ----->
    Addr=00000000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the pattern written out matches the one read, the test passes.

```
H   User_Data Space .....Running -----> PASSED
```

## 6.9.9 MMU I - User\_Prog Space

MMU I

The user\_program space test command (MMU I) is described in the following paragraphs.

**Description**

This command tests the function code signal lines connecting into the MMU by accessing user program space. This causes the MMU to read the function code and do a table walk as a part of its translation. The test is implemented in the following manner:

- Step 1. Write a pattern out to an area that is mapped to user program space for diagnostic purposes.
- Step 2. Enable the MMU.
- Step 3. Read the area where the pattern was written to, using the function code for user program space. The test fails if the pattern does not match that written out.

**Command Input**

```
141-Diag> MMU I
```

**Response/Messages**

After entering this command, the display should read as follows:

```
I   User_Prog Space .....Running ----->
```

If the pattern written out does not match that read, the test fails.

```
I   User_Prog Space .....Running ----->
    Addr=00000000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the pattern written out matches the one read, the test passes.

```
I   User_Prog Space .....Running -----> PASSED
```

## 6.9.10 MMU J - Indirect Page

MMU J

The indirect page test command (MMU J) is described in the following paragraphs.

**Description**

This command tests the ability of the MMU to handle an indirect descriptor. The test is implemented in the following manner:

- Step 1. Modify the descriptor for the first RAM page to point to the descriptor for the next RAM page.
- Step 2. Write a known value into the first location of the second RAM page and the complement of that value into the first location of the first RAM page.
- Step 3. Enable the MMU.
- Step 4. Read the first location of the first virtual RAM page. This should address the first location in the second physical RAM page due to the indirect.
- Step 5. If the value read is not the value written out to the second RAM page in Step 2, then the test fails.
- Step 6. Disable the MMU.

**Command Input**

```
141-Diag> MMU J
```

**Response/Messages**

After entering this command, the display should read as follows:

```
J   Indirect Page .....Running ----->
```

If the value that was supposedly read from the first virtual page in Step 4 does not match the value written in Step 2 to the second physical page, then the test fails.

```
J   Indirect Page .....Running ----->
    Addr=00000000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the value matches, then the indirect mechanism has functioned correctly and the test passes.

```
J   Indirect Page .....Running -----> PASSED
```

## 6.9.11 MMU K - Page-Desc Used-Bit

MMU K

The page-descriptor used-bit test command (MMU K) is described in the following paragraphs.

**Description**

This command tests the ability of the MMU to set the Used-bit in a page descriptor when the page gets accessed. The test is implemented in the following manner:

- Step 1. Clear the Used-bit in a page descriptor.
- Step 2. Enable the MMU.
- Step 3. Read from the page.
- Step 4. Examine the page descriptor. If the Used-bit is not set, then the test fails.

**Command Input**

141-Diag> MMU K

**Response/Messages**

After entering this command, the display should read as follows:

```
K   Page-Desc Used-Bit .....Running ----->
```

If the Used-bit does not get set by the access in Step 3, then the test fails.

```
K   Page-Desc Used-Bit .....Running ----->
    Addr=00000000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the Used-bit does get set, then the test passes.

```
K   Page-Desc Used-Bit .....Running -----> PASSED
```

## 6.9.12 MMU L - Page-Desc Modify-Bit

MMU L

The page-descriptor modify-bit test command (MMU L) is described in the following paragraphs.

**Description**

This command tests the ability of the MMU to set the Modify-bit in a page descriptor when the page is written. The test is implemented in the following manner:

- Step 1. Clear the Modify-bit in a page descriptor.
- Step 2. Enable the MMU.
- Step 3. Write from the page.
- Step 4. Examine the page descriptor. If the Modify-bit is not set, then the test fails.

**Command Input**

```
141-Diag> MMU L
```

**Response/Messages**

After entering this command, the display should read as follows:

```
L   Page-Desc Modify-Bit .....Running ----->
```

If the Modify-bit does not get set by the access in Step 3, then the test fails.

```
L   Page-Desc Modify-Bit .....Running ----->
    Addr=00000000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the Modify-bit does get set, then the test passes.

```
L   Page-Desc Modify-Bit .....Running -----> PASSED
```

## 6.9.13 MMU M - Segment-Desc Used-Bit

MMU M

The segment-descriptor used-bit test command (MMU M) is described in the following paragraphs.

## Description

This command tests the ability of the MMU to set the Used-bit in a segment descriptor when the corresponding segment is accessed. The test is implemented in the following manner:

- Step 1. Clear the Used-bit in a segment descriptor.
- Step 2. Enable the MMU.
- Step 3. Read from an address mapped to that segment.
- Step 4. Check the Used-bit in the segment descriptor. If it has not been set, the test fails.

## Command Input

```
141-Diag> MMU M
```

## Response/Messages

After entering this command, the display should read as follows:

```
M Segment-Desc Used-Bit .....Running ----->
```

If the Used-bit does not get set by the access in Step 3, then the test fails.

```
M Segment-Desc Used-Bit .....Running ----->
  Addr=00000000 Expect=00000001 Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the Used-bit does get set, then the test passes.

```
M Segment-Desc Used-Bit .....Running -----> PASSED
```

## 6.9.14 MMU P - Invalid Page

MMU P

The invalid page test command (MMU P) is described in the following paragraphs.

## Description

This command tests the ability of the MMU to detect an invalid page and generate bus error when access is attempted to that page. The invalid page is intentionally declared that way for test purposes. The test is implemented in the following manner:

- Step 1. Modify the descriptor for a RAM page to make it invalid.
- Step 2. Enable the MMU.
- Step 3. Attempt to read from the page. This should generate a bus error.
- Step 4. If no bus error occurred, then the test fails.

## Command Input

```
141-Diag> MMU P
```

## Response/Messages

After entering this command, the display should read as follows:

```
P   Invalid Page .....Running ----->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
P   Invalid Page .....Running ----->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
P   Invalid Page .....Running -----> PASSED
```

## 6.9.15 MMU Q - Invalid Segment

MMU Q

The invalid segment test command (MMU Q) is described in the following paragraphs.

**Description**

This command tests the ability of the MMU to detect an invalid segment and generate bus error when access is attempted to that segment. The invalid segment is intentionally declared that way for test purposes. The test is implemented in the following manner:

- Step 1. Modify the descriptor for a RAM segment to make it invalid.
- Step 2. Enable the MMU.
- Step 3. Attempt to read from the page in the segment. This should generate a bus error.
- Step 4. If no bus error occurred, then the test fails.

**Command Input**

141-Diag> MMU Q

**Response/Messages**

After entering this command, the display should read as follows:

```
Q   Invalid Segment .....Running ----->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
Q   Invalid Segment .....Running ----->
    Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
Q   Invalid Segment .....Running -----> PASSED
```

## 6.9.16 MMU R - Write-Protect Page

MMU R

The write-protect page test command (MMU R) is described in the following paragraphs.

**Description**

This command tests the page write-protect mechanism in the MMU. If the MMU is functioning correctly, then attempting a write to a protected page causes a bus error. The test is implemented in the following manner:

- Step 1. Set the WP bit in the descriptor for the first RAM page.
- Step 2. Enable the MMU.
- Step 3. Attempt to write to the protected page.
- Step 4. If a bus error does not occur, then the test fails.

**Command Input**

```
141-Diag> MMU R
```

**Response/Messages**

After entering this command, the display should read as follows:

```
R   Write-Protect Page .....Running ----->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
R   Write-Protect Page .....Running ----->
    Addr=00000000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
R   Write-Protect Page .....Running -----> PASSED
```

## 6.9.17 MMU S - Write-Protect Segment

MMU S

The write-protect segment test command (MMU S) is described in the following paragraphs.

## Description

This command tests the segment write-protect mechanism in the MMU. If the MMU is functioning correctly, then attempting a write to a protected segment will cause a bus error. The test is implemented in the following manner:

- Step 1. Set the WP bit in the descriptor for the first RAM segment.
- Step 2. Enable the MMU.
- Step 3. Attempt to write to a page in the protected segment.
- Step 4. If a bus error does not occur, then the test fails.

## Command Input

```
141-Diag> MMU S
```

## Response/Messages

After entering this command, the display should read as follows:

```
S   Write-Protect Segment .....Running ----->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
S   Write-Protect Segment .....Running ----->
    Addr=00000000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
S   Write-Protect Segment .....Running -----> PASSED
```

**6.9.18 MMU V - Upper-Limit Violation****MMU V**

The upper-limit violation test command (MMU V) is described in the following paragraphs.

**Description**

This command tests the capability of the MMU to detect when a logical address exceeds the upper limit of a segment. This condition is called an upper limit violation and should cause a bus error. The test is implemented in the following manner:

- Step 1. Modify the descriptor for a segment to lower the upper limit to where it permits access to only the first page.
- Step 2. Attempt access to the second page.
- Step 3. This should cause a bus error. If no bus error occurs, then the test fails.

**Command Input**

```
141-Diag> MMU V
```

**Response/Messages**

After entering this command, the display should read as follows:

```
V   Upper-Limit Violation .....Running ----->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
V   Upper-Limit Violation .....Running ----->
    Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
V   Upper-Limit Violation .....Running -----> PASSED
```

## 6.9.19 MMU W - Lower-Limit Violation

MMU W

The lower-limit violation test command (MMU W) is described in the following paragraphs.

**Description**

This command tests the capability of the MMU to detect when a logical address exceeds the lower limit of a segment. This condition is called a lower limit violation and should cause a bus error. The test is implemented in the following manner:

- Step 1. Modify the descriptor for a segment to set the lower limit to where it permits access to only the second page.
- Step 2. Attempt access to the first page.
- Step 3. This should cause a bus error. If no bus error occurs, then the test fails.

**Command Input**

```
141-Diag> MMU W
```

**Response/Messages**

After entering this command, the display should read as follows:

```
W   Lower-Limit Violation .....Running ----->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
W   Lower-Limit Violation .....Running ----->
    Addr=00000000   Expect=00000001   Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
W   Lower-Limit Violation .....Running -----> PASSED
```

## 6.9.20 MMU X - Prefetch on Invalid-Page Boundary

MMU X

The prefetch on invalid-page boundary test command (MMU X) is described in the following paragraphs.

**Description**

This command tests to see if the MC68030 ignores a bus error that occurs as a result of a prefetch. The MMU signals a bus error if a prefetch operation crosses a page boundary into an invalid page. The MC68030 is to ignore such bus errors. The test is implemented in the following manner:

- Step 1. Invalidate the second page mapped to user program space. This page is shared with user data space.
- Step 2. Insert a trap instruction at the last location of the previous page (this page is still valid).
- Step 3. Point to a special trap handler that checks for the bus error by examining some flags.
- Step 4. Enable the MMU.
- Step 5. Branch to the address in the first page of the trap instruction, leaving supervisor mode and entering user mode.
- Step 6. The MC68030 should fetch the operating word at the end of the valid page, then attempt to prefetch the next word, which crosses the page boundary into the invalid page.
- Step 7. If the MC68030 takes a bus error exception, then the test fails. Once bus error exception processing completes, control passes to the special trap handler.
- Step 8. Once in the special trap handler, the stack is cleaned up (leaving the MC68030 in supervisor mode), and a test is performed to determine if the MC68030 executed a bus error exception.
- Step 9. If the bus error occurred, then the test fails.

**Command Input**

```
141-Diag> MMU X
```

**Response/Messages**

After entering this command, the display should read as follows:

```
X Prefetch On Inv-Page .....Running ----->
```

If a bus error occurs, then the test fails.

```
X   Prefetch On Inv-Page .....Running ----->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here
.... FAILED
```

If the prefetching does not cause the MC68030 to take a bus error exception, then the test passes.

```
X   Prefetch On Inv-Page .....Running -----> PASSED
```

## 6.9.21 MMU Y - Modify-Bit and Index

MMU Y

The modify-bit and index test command (MMU Y) is described in the following paragraphs.

**Description**

This command tests the capability of the MMU to set the Modify-Bit in a page descriptor of a page which has an index field greater than 0 (not a page-0) when the page is written.

**Command Input**

```
141-Diag> MMU Y
```

**Response/Messages**

After entering this command, the display should read as follows:

```
Y   Modify-Bit & Index .....Running ----->
```

If the Modify-Bit does not get set by the write, then the test fails.

```
Y   Modify-Bit & Index .....Running ----->
    Addr=00F00000   Expect=00000010   Read=00000000
(map of table walk displayed here
.... FAILED
```

If the Modify-Bit does get set, the test passed.

```
Y   Modify-Bit & Index .....Running -----> PASSED
```

## 6.9.22 MMU 0 - Read/Modify/Write Cycle

MMU 0

The read/modify/write cycle test command (MMU 0) is described in the following paragraphs.

**Description**

This test performs the Test-And-Set (TAS) instruction in three modes to verify that the MMU functions correctly during read/modify/write cycles.

The message Hit Page is displayed. The MMU is turned on and a write access is performed to cache the address translation for that location. The first TAS is then done to verify that a hit page can be accessed. No bus error should result from this. The test fails if either a bus error occurs or the location (in RAM) written to does not contain \$80 afterward.

The MMU is shut off and the message Missed Page displayed. The MMU is turned on, flushing the Address Translation Cache (ATC). A TAS is then attempted. Because the ATC was just flushed, the access should cause a table walk and a single bus error. An error is declared if other than one bus error occurred.

If the two previous phases completed successfully, the message Unmodified Page is displayed and the final phase begun. The Modified bit for a particular page is cleared, then a read from that page is performed to cache its address translation. Next, a TAS is attempted to that location. A bus error should occur to allow the MMU time to set the Modified bit. An error is declared if the Modified bit was not set or if other than one bus error occurred.

**Command Input**

```
141-Diag> MMU 0
```

**Response/Messages**

After entering this command, the display should read as follows:

```
0   R/M/W Cycles .....Running ----->
      Hit page .....
```

If the access to the hit page causes a bus error, or the location written to does not contain \$80, then the test fails and the table walk is displayed.

```
0   R/M/W Cycles .....Running ----->
      Hit page .....
```

Addr=XXXXXXXX	Expect=80000000	Read=00000000
---------------	-----------------	---------------

(map of table walk displayed here  
.... FAILED

If the access to the missed page does not cause a bus error, then the test fails and the table walk is displayed.

```

0   R/M/W Cycles .....Running ----->
      Hit page .....
      Missed page .....
      Addr=XXXXXXXX      Expect=80000000      Read=00000000
(map of table walk displayed here
.... FAILED

```

If the access to the modified page does not cause a bus error, or the Modified bit for the page does not get set, then the test fails and the table walk is displayed.

```

0   R/M/W Cycles .....Running ----->
      Hit page .....
      Missed page .....
      Modified page ....
      Addr=XXXXXXXX      Expect=80000000      Read=00000000
(map of table walk displayed here
.... FAILED

```

If all three phases of the test complete successfully, then the test passes and the display appears as follows:

```

0   R/M/W Cycles .....Running ----->
      Hit page .....
      Missed page .....
      Modified page .... PASSED

```



**6.10 REAL-TIME CLOCK TEST - Command RTC****RTC**

The real-time clock test command (RTC) is described in the following paragraphs.

**Description**

This command tests the MK48T02 RTC. The battery backed-up RAM is tested, the oscillator is stopped and started, and the output is checked for roll-over.

**Command Input**

```
141-Diag> RTC
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
RTC Real Time Clock Test.....Running ----->
```

If the non-destructive test of the RAM fails, the following message appears:

```
RTC Real Time Clock Test.....Running ----->..... FAILED
RAM failed at $XXXXXXXX; Wrote $XX; Read $XX
```

**CAUTION**

**WHETHER THE TESTS PASS OR FAIL, TIME DISPLAYED AFTER TEST MAY NOT BE CORRECT.**

If the oscillator does not stop on command, this message is displayed:

```
RTC Real Time Clock Test.....Running ----->..... FAILED
Can't stop RTC oscillator
```

If the oscillator does not restart on command, this message is displayed:

```
RTC Real Time Clock Test.....Running ----->..... FAILED
Can't start RTC oscillator
```

The test next checks time, day of week, and date in the roll-over. If any digit is wrong in roll-over, then the test fails and the appropriate one of the following error message appears as :

```
Time read was xx:xx:xx, should be 00:00:01
Day of week not 1
Date read was xx/xx/xx, should be 01/01/00
```

If a bus error occurs, the error message is:

Unexpected Bus Error

If all parts of the test are completed correctly, then the test passes.

RTC Real Time Clock Test.....Running -----> PASSED

6

**6.11 BUS ERROR TEST - Command BERR****BERR**

The bus error test command (BERR) is described in the following paragraphs.

**Description**

This command tests for local bus time-out and global bus time-out bus error conditions, including the following:

- no bus error by reading from ROM
- local bus time-out by reading from an undefined FC location
- local bus time-out by writing to an undefined FC location

**Command Input**

```
141-Diag> BERR
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
BERR Bus Error Test.....Running ----->
```

If a bus error occurs in the first part of the test, then the test fails and the display appears as follows.

```
BERR Bus Error Test.....Running ----->..... FAILED
Got Bus Error when reading from ROM
```

If no bus error occurs in one of the other parts of the test, then the test fails and the appropriate error message appears as one of the following:

```
No Bus Error when reading from BAD address space
No Bus Error when writing to BAD address space
```

If all three parts of the test are completed correctly, then the test passes.

```
BERR Bus Error Test.....Running -----> PASSED
```

## 6.12 FLOATING POINT COPROCESSOR (MC68882) TEST - Command FPC

FPC

The floating point coprocessor test command (FPC) is described in the following paragraphs.

## Description

This command tests the functions of the FPC, including all the types of FMOVE, FMOVEM, FSAVE, and FRESTORE instructions; and tests various arithmetic instructions that set and clear the bits of the FPC Status Register (FPSR).

## Command Input

141-Diag> FPC

## Response/Messages

After the command has been issued, the following line is printed:

FPC Floating Pnt. Coprocessor Test.....Running ----->

If there is no FPC or if it is inoperable, then the display appears as follows:

FPC Floating Pnt. Coprocessor Test.....Running ----->..... FAILED  
No FPC detected

If any part of the test fails, then the display appears as follows.

FPC Floating Pnt. Coprocessor Test.....Running ----->..... FAILED  
Test failed FPC routine at XXXXXXXX

Here XXXXXXXX is the hexadecimal address of the part of the test that failed. The user may look in detail at this location in the 141Bug EPROM to determine exactly what function failed.

If any part of the test is halted by an unplanned interrupt, then the display appears as follows.

FPC Floating Pnt. Coprocessor Test.....Running ----->..... FAILED  
Unexpected interrupt

If all parts of the test are completed correctly, then the test passes.

FPC Floating Pnt. Coprocessor Test.....Running -----> PASSED

## 6.13 MC68681 DUART (SIO) TESTS

SIO

The following paragraphs detail the diagnostics needed to test the DUART device and its associated circuitry. The table below lists the serial I/O port diagnostic tests.

MC68681 DUART (SIO) Tests	
MONITOR COMMAND	TITLE
SIO A	Data formats test
SIO B	Baud rate test
SIO C	TX/RX ready IRQ test
SIO D	TX/RX ready test
SIO E	FIFO full test
SIO F	BREAK test
SIO G	SIO timer test

## Hardware Configuration

The following hardware is required to perform these tests.

- MVME141 (board being tested)
- Video display terminal
- Male RS-232C loop back connector as shown below

## Loop Back Connector Wiring Diagram

## 6.13.1 Data Formats Test

SIO A

The data formats test command (SIO A) is described in the following paragraphs.

**Description**

This test ensures that data can be sent and received in all formats at 9600 baud. The algorithm for this test is as follows:

For no parity, even parity, odd parity:

For one stop bit, two stop bits:

For 5-, 6-, 7-, 8-bit data widths:

Send/Receive full data range (\$00-\$FF).

Next data width.

Next number of stop bits.

Next parity mode

**Command Input**

141-Diag> SIO A

**Response/Messages**

The monitor responds with:

A Data Formats Test .....Running ----->

Progress of the test is output as the different steps of the tests run.

If no errors occur, then the display appears as follows:

A Data Formats Test .....Running ----->

Parity : None

One Stop Bit ... Two Stop Bits ...

Parity : Even

One Stop Bit ... Two Stop Bits ...

Parity : Odd

One Stop Bit ... Two Stop Bits ... PASSED

**6.13.2 Baud Rate Test****SIO B**

The baud rate test command (SIO B) is described in the following paragraphs.

**Description**

This test verifies that data can be transmitted at all set 2 baud rates (refer to the MC68681 DUART specification). A set of complement characters is transmitted at each baud rate on port 2 in local loopback mode. Only channel B is tested.

**Command Input**

141-Diag> SIO B

**Response/Messages**

The monitor responds with:

B    Baud Rate Test .....Running ----->

If no errors occur, then the display appears as follows:

B    Baud Rate Test .....Running -----> PASSED

## 6.13.3 TX/RX Ready IRQ Test

SIO C

The TX/RX ready IRQ test command (SIO C) is described in the following paragraphs.

## Description

This test verifies that the DUART interrupt circuitry is working properly and that the MVME141 can properly respond to a DUART IRQ. This test is performed in the internal loopback mode and tests the TXRDY/RXRDY IRQ functions for port 2. The test is performed on port 2 only and generates interrupts for both TXRDY and RXRDY. Four characters are used during this test: \$55, \$AA, \$FF, and \$00. The DUART IRQ generates a level 5 auto-vectored IRQ to the CPU, which should receive 8 IRQs during this test (4 for TX, 4 for RX).

## Command Input

```
141-Diag> SIO C
```

## Response/Messages

The monitor responds with:

```
C TX/RX Ready IRQ Test .....Running ----->
```

If no errors occur, then the display appears as follows:

```
C TX/RX Ready IRQ Test .....Running -----> PASSED
```

## Example Failure Messages:

```
DATA COMPARE ERROR ON PORT 2 -
    (Received data was not what was expected)

RX FORMAT ERROR ON PORT 2 ! STATUS RECEIVED=xxxxxxx (binary)
    (Indicates defective port in DUART, bad status on receive)

TX IRQ TIMEOUT ON PORT 2 !
    (No TX IRQ Action detected, probably no IRQ generated)

RX IRQ TIMEOUT ON PORT 2 !
    (TXRDY generated IRQs but RXRDY did not; defective DUART)

IRQ COUNT = $x -
    (Wrong number of IRQs detected; bad DUART or noisy IRQ line)
```

SIO C

RX STATUS ERROR MASK = xxxxxxxx

RX DATA ERROR MASK = xxxxxxxx

RX PORT ERROR MASK = xxxxxxxx

(Displayed at end of test if any of these errors occurred on any port, bit 1 = port 2)

## 6.13.4 TX/RX Ready Test

SIO D

The TX/RX ready test command (SIO D) is described in the following paragraphs.

**Description**

This test checks the operation of both the TXRDY and RXRDY status bits of the DUART port status registers and ensures that they reflect the proper status and that the interrupt status register bits match. This test is performed in the internal loopback mode. For port 2 the transmitter and receiver are enabled. Two characters (\$55 and \$AA) are transmitted with the appropriate status checks both before and after transmission, before and after reception, and when reading the character to verify both states of the status lines. If errors are detected, messages indicate the failed function and the port number.

**Command Input**

```
141-Diag> SIO D
```

**Response/Messages**

The monitor responds with:

```
D TX/RX Ready Test .....Running ----->
```

If no errors occur, then the display appears as follows:

```
D TX/RX Ready Test .....Running -----> PASSED
```

**6.13.5 FIFO Full Test****SIO E**

The FIFO full test command (SIO E) is described in the following paragraphs.

**Description**

This test checks the FIFO full status bit function in the port status register and in the interrupt status register of the DUART. This test is performed in the internal loopback mode. For port 2, three characters are transmitted and the FIFO full status is checked to be active in both registers. A fourth character is transmitted to fill the holding register. After the transmitter is empty, the first character transmitted is read and verified and the status bits are verified as still active. The second character is then read and verified and status is checked to be inactive (FIFO not full). The received characters are read and verified.

**Command Input**

141-Diag> **SIO E**

**Response/Messages**

The monitor responds with:

E    FIFO Full Test .....Running ----->

If no errors occur, then the display appears as follows:

E    FIFO Full Test .....Running ----->    PASSED

## 6.13.6 BREAK Test

SIO F

The BREAK test command (SIO F) is described in the following paragraphs.

## Description

This test checks the BREAK generation and detection functions of the DUART. It is performed in the internal loopback mode for port 2. A START BREAK command is issued to the port and status is checked in the port status register for BREAK DETECTED and in the interrupt status register for CHANGE-IN-BREAK. Status is cleared with the RESET ERROR STATUS and the RESET BREAK CHANGE INTERRUPT commands. A STOP BREAK command is then issued and the ISR is checked to have detected the change in BREAK.

## Command Input

```
141-Diag> SIO F
```

## Response/Messages

The monitor responds with:

```
F   BREAK Test .....Running ----->
```

If no errors occur, then the display appears as follows:

```
F   BREAK Test .....Running -----> PASSED
```

## 6.13.7 SIO Timer Test

SIO G

The SIO timer test command (SIO G) is described in the following paragraphs.

**Description**

This test checks the timer function of the DUART. It verifies loadability and readability of the counter/timer registers by writing a test value into the register, starting and stopping the counter, and then verifying the value (masking the least significant nibble). The counter is then allowed to timeout and the counter/timer/ready bit in the interrupt status register is checked to be active. A STOP COUNTER command is issued and the C/TR bit is checked to be inactive.

**Command Input**

141-Diag> SIO G

**Response/Messages**

The monitor responds with:

G SIO Timer Test .....Running ----->

If no errors occur, then the display appears as follows:

G SIO Timer Test .....Running -----> PASSED

## 6.14 VME GATE ARRAY TEST - Command VMEGA

VMEGA

The VME gate array test command (VMEGA) is described in the following paragraphs.

**Description**

This command performs a test of the VME gate array (VMEGA) registers. First VMEbus mastership is obtained and RAM accesses from the VMEbus are disabled, then the VMEbus is released.

Executes reads and writes from the local bus to all used or predictable bits in the following registers:

- System controller configuration register
- Master configuration register
- Timer configuration register
- Slave address modifier register
- Master address modifier register
- interrupt handler mask register
- Utility interrupt mask register
- Utility interrupt vector register
- VMEbus status/ID register
- GCSR base address configuration register
- Board identification register
- General purpose control/status registers 0-4

**Command Input**

```
141-Diag> VMEGA
```

**Response/Messages**

After the command has been issued, the following line is printed:

```
VMEGA   VME Gate Array Test .....Running ----->
```

If any part of the test fails, then the display appears as follows.

```
VMEGA   VME Gate Array Test .....Running ----->..... FAILED
--(error message)
```

## VMEGA

Here, (error message) is one of the following:

ROBIN bit in SCCR is high should be low  
 ROBIN bit in SCCR is low should be high  
 MCR error; data written: \$xx read: \$xx  
 TCR error; data written: \$xx read: \$xx  
 SAMR error; data written: \$xx read: \$xx  
 MAMR error; data written: \$xx read: \$xx  
 IMHR error; data written: \$xx read: \$xx  
 UIMR error; data written: \$xx read: \$xx  
 UIVR error; data written: \$xx read: \$xx  
 SIDR error; data written: \$xx read: \$xx  
 GCSRBAR error; data written: \$xx read: \$xx  
 BIDR error; data written: \$xx read: \$xx  
 GPRO error; data written: \$xx read: \$xx  
 GPR1 error; data written: \$xx read: \$xx  
 GPR2 error; data written: \$xx read: \$xx  
 GPR3 error; data written: \$xx read: \$xx  
 GPR4 error; data written: \$xx read: \$xx

Here, \$xx are hexadecimal numbers. For further information on the VME gate array device refer to the MVME141 User's Manual.

If all parts of the test are completed correctly, then the test passes.

VMEGA VME Gate Array Test .....Running -----> PASSED

THIS PAGE INTENTIONALLY LEFT BLANK.

6

## APPENDIX A - MVME141Bug SYSTEM MODE OPERATION

## GENERAL DESCRIPTION

To provide compatability with the Motorola VME Delta Series systems, the MVME141Bug has a special mode of operation that allows the following features to be enabled:

- a. Extended confidence tests that are run automatically on power up or reset of the MVME141.
- b. A menu that allows several system start up features to be selected, such as:
  - . Continue start up.
  - . Select alternate boot device.
  - . Select MVME141Bug debugger.
  - . Initiate a service call.
  - . Display test errors found during start up confidence testing.
  - . Dump contents of system memory to tape.
- c. Return to the menu upon system start up errors instead of return to the debugger.
- d. Enabling of the Bug autoboot sequence.

The flow of system mode operation is shown in Figure 1. Upon either power up or system reset, the MVME141 first executes a limited confidence test suite. This is the same test suite that the Bug normally executes on power up when not in the system mode. Upon successful completion of the limited confidence tests, a five second period is allowed to interrupt the autoboot sequence. By typing an "h" the user can cause the module to display the service menu, permitting the selection of an alternate boot device, entry to the debugger, etc., as described above. Upon selection of "continue start up" the module conducts a more extensive confidence test, including a complete parity memory test. This memory test takes a minimum of 90 seconds for a 4Mb onboard memory. Successful completion of the extended confidence test initiates the autoboot sequence, with boot taking place either from the default device (refer to Chapter 3 for information on entering/changing the default boot device) or from the selected boot device if an alternate device has been selected.

If the limited confidence test fails to complete correctly, it may display an error message. Explanations of these error messages can be found in Appendix B. Error message explanations for the extended confidence test are given in Chapter 6 under the heading for the failed test.

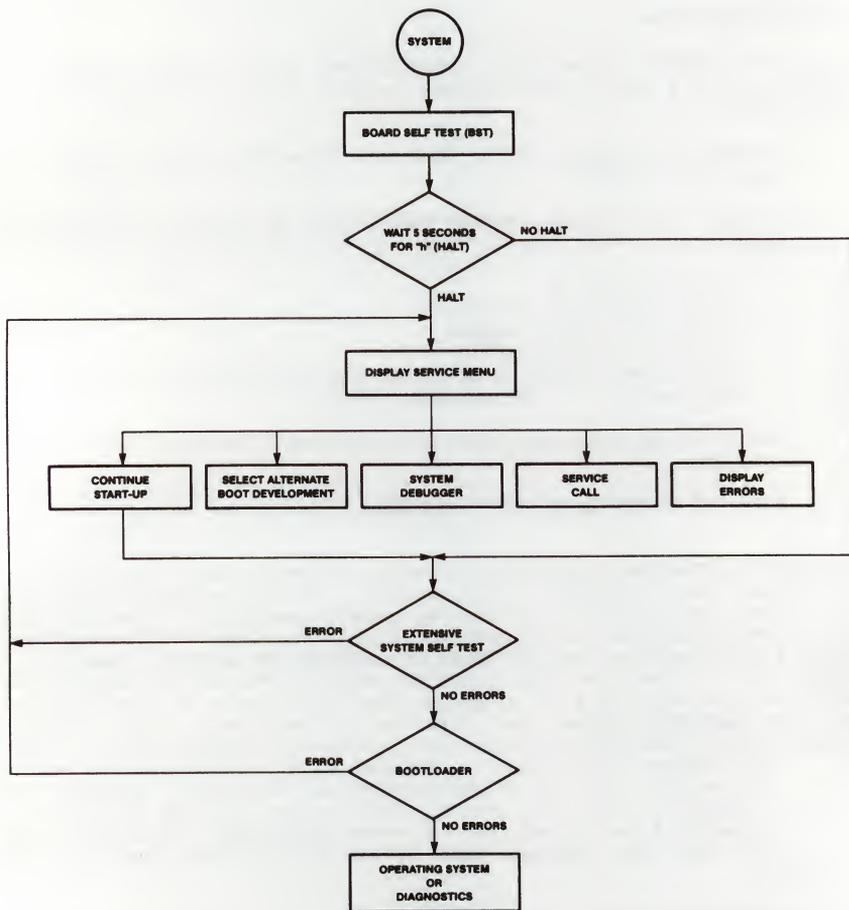


FIGURE 1. Flow Diagram of 141Bug System Operation Mode

The sequence of extended confidence tests for the MVME141 is as follows:

- 68030 register tests
- 68030 instruction tests
- 68030 address mode tests
- 68030 exception processing tests
- 68030 cache tests
- fast memory test
- fast address test
- VMEbus test
- MMU tests
- onboard tests
  - bus error test
  - FPC (floating point coprocessor) test
  - PCC (Peripheral Controller Chip) test
  - VMEbus chip test
  - LANCE chip test
  - RTC test
  - serial port test

#### MENU DETAILS

Following are more detailed descriptions of the menu selections.

##### Continue System Start Up

The only action required by the user is to enter a 1 followed by a carriage return. The system then continues the start up process by initializing extended confidence testing followed by a system boot.

##### Select Alternate Boot Device

The user is prompted with "Enter Alternate Boot Device (Controller, Drive, File):". The selection of devices that are supported by the 141Bug are listed in Appendix E. Entry of a selected device followed by a carriage return redisplayes the menu for another selection, normally "continue system start up" at this point.

##### Go to System Debugger

When selected, this entry places the user in 141Bug, diagnostic mode, indicated by the prompt 141-Diag>. If desired, return to the menu can be accomplished by typing "menu" when the Bug prompt appears. When in 141-Diag mode, operation is defined by sections of this manual dealing with the Bug and FAT diagnostics.

## Initiate service call

The initiate service call function is described in the following paragraphs.

### General Flow

Initiated by typing a 4 (CR) in response to the menu prompt, this function is normally used to complete a connection to a service organization which can then use the "dual console" mode of operation to assist a customer with a problem. Interaction with the service call function proceeds as follows:

First, the system asks

Is the modem: 0-UDS, 1-Hayes, 2-Manual, 3-Terminal: Your Selection?

Explanation:

UDS means that the modem is compatible with the UDS modem protocol as used in internal VME Delta Series modems. The model number of this modem is UDS 2122662.

Hayes means that the modem is compatible with a minimal subset of the Hayes modem protocol. This minimum subset is chosen to address the broadest spectrum of Hayes compatible modem products. Note that the modem itself is not tested when Hayes protocol is chosen, while the modem is tested with the UDS protocol choice.

Manual mode connects directly to the modem in an ASCII terminal mode, allowing any nonstandard protocol modem to be used.

Terminal mode is used to connect any ASCII terminal in place of a modem, via a null modem, or equivalent cable. It is useful in certain troubleshooting applications for providing a slave terminal without the necessity of dialing through a modem.

When a selection of one of the above options is made (option 0 in this case), the system asks:

Do you want to change the baud rate from 1200 (Y/N)?

Note that any question requiring a Y or N answer defaults to the response listed furthest to the right in the line (i.e., a question with Y/N defaults to NO if only a carriage return is entered). If the user answers Y to the baud rate question, the system prompts:

Baud rate [300, 1200, 2400, 4800, 9600] 1200?

The user should enter a selected baud rate, such as 300, and type a return. A return only leaves the baud rate as previously set. The system then asks:

Is the modem already connected to customer service (Y/N)?

When a connection has been made to customer service (or any other remote device) hang up does not automatically occur; it is an operation initiated by the user. If a system reset has occurred, for instance, a hang up does not take place, and connection to CSO is still in effect. In this case, it is not necessary or desirable to attempt to reconnect on a connection that is already in effect. When an answer is entered to the question, the system responds:

Enter System ID Number:

This number is one assigned to the user system by its affiliated Customer Service Organization. The system itself does not care what is entered here, but the Customer Service computer may do a check to assure the validity of this number for login purposes. The system responds with:

Wait for an incoming Call or Dial Out (W/D)?

The user has the option of either waiting for the other computer to dial in to complete the connection, or dialing out itself. If W is selected, then skip the next two steps. If D is selected, the system asks:

Hayes Modem:

(T) = Tone Dialing (Default), (P) = Pulse Dialing  
(,) = Pause and Search for a Dial Tone

UDS Modem:

(T) = Tone Dialing (Default), (P) = Pulse Dialing  
(=) = Pause and Search for a Dial Tone  
(,) = Wait 2 Seconds

Enter CSO phone number:

The user must enter the number, including area code if required, without any separators except for a (,) or (=) if required to allow search for a dial tone (depending on which modem protocol was selected), such as when dialing out of a location having an internal switchboard. Additionally, the number must be prefaced by one of the above dialing mode selections. The dialing selection can also be changed within the number being dialed if necessary if an internal dialing system takes a different dialing mode than the external world switched network. When connection has been made, the system reports:

Service Call in progress - Connected

The remote system can now send one of three unique commands to the local system to request specific actions via the local firmware. These commands are:

#### Dump Memory Command

The command to dump the private RAM used by the 141 ROM to log errors is *dpp1*. The command must be received as shown in lowercase with no carriage return. Also no editing is allowed and each byte must be received within 2 seconds of the last. This command is four bytes long.

The memory dumped is the first block of memory past the exception vectors in the address range \$800 to \$1FFF. The memory is formatted into S-records as defined by Motorola. The S-record is sent and an ACK character \$06 is expected after each record is sent. If any other byte is received, the record is resent. The record is resent 10 times before the command is aborted. The S2 record is used for the 24-bit address of the data sent. When the end of the private memory is reached, the S9 record is sent to terminate the dumping of memory.

The dumping command displays on the console that s-records are being dumped and that dumping is complete. After dumping memory is complete, the code waits for another command.

Refer to Appendix C for details on S-records.

#### Message Command

The command to send a message from the CSO center to the console of the calling system is <mess>, 4 bytes, followed by a string of data no more than 80 bytes in length terminated with a carriage return. The ROM code moves the string to the console followed by a carriage return and a line feed.

This command can be used to send canned messages to the operator, giving some indication of activity while various processes are taking place at CSO. For example, "Please Stand By". Many of these message commands may be sent while in the command mode.

#### Request for Concurrent Console Command

The request for concurrent console command is <rcc>, 3 bytes only. This prompts the operator about the request. If the operator enters "y", a single character "y" is sent to CSO followed by the console menu as displayed on the operators console. If the operator enters "n", then the single character "f" is sent to CSO and the call is terminated.

When concurrent mode is entered all input from either port, console, or remote, is taken simultaneously. All output is sent to both ports concurrently. Either the console or the remote console may terminate the concurrent mode at any time by typing a control-a. The phone line is hung up by the 141 ROM code and a message is displayed indicating the end of the concurrent mode.

The most likely command sequence at this point is a message command to indicate connection to the remote system, followed by a request for concurrent mode operation. When these are received, the user system asks:

Concurrent mode (Y/N)?

If the user wishes to enter concurrent mode, Y must be selected. The system then presents the information:

Control A to exit concurrent mode

The menu is redisplayed and concurrent mode is in effect. Any normal system operation can now be initiated at either the local or remote connected terminal, including system reboot.

- 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
- 7) Start Conversation Mode

Note that a seventh entry has been added to the menu. This Conversation Mode entry allows either party to initiate a direct conversation mode between the two terminals, the remote system terminal, and the local terminal. This seventh entry is only displayed when the system is in concurrent mode, although it actually can be selected and used at any time; only the prompt line is not displayed in normal operation. Conversation mode can be exited by typing a control A, in which case concurrent mode is terminated as well and the modem is hung up. To terminate conversation mode, but remain in concurrent mode, type the following command:

(CR),(CR)

The system then redisplayes the selection menu for further operator action.

When the menu is displayed, and concurrent mode is in effect, there is another path available to terminate the concurrent connection. If the user selects menu entry 4 (Initiate Service Call) while a call is underway, the system asks:

Do you wish to disconnect the remote link (Y/N)?

If the user answers N, the system gives the option of returning to (or entering) the conversation mode:

Do you wish the conversation mode (Y/N)?

A Y response results in return to conversation mode, while an N redisplay the menu.

The system responds with the following series of messages if the disconnect option is chosen:

Wait for concurrent mode to terminate  
Hanging up the phone  
Concurrent mode is terminated

The last message is followed by the display of the menu WITHOUT the seventh selection available. Normal system operation is now possible.

### Manual Mode Connection

As described briefly earlier, a manual modem connect mode is available to allow use of modems that do not adhere to either of the standard protocols supported, but have a defined ASCII command set. If the manual mode is selected, a few differences must be taken into account. A new mode, called "Transparent Mode" is entered when manual modem control is attempted. This means that the user terminal is in effect connected directly to the modem for control purposes. When in transparent mode, the user must take responsibility for modem control, and informing the system of when connection has taken place, etc. If "manual mode" selection is made from the "is the modem --" prompt, the following dialog takes place.

All prompts and expected responses through the "Enter System ID Number:" takes place as above. However, in manual mode, after the ID number has been entered, the system prompts:

Manually call CS0 and when you are connected, exit the transparent mode  
Escape character: \$01=^A

The user should type a control A when connection is made, or if for any reason a connection cannot be made. Because the system has no knowledge of the status of the system when transparent mode is exited, it asks:

Did you make the connection (Y/N)?

If the user answers Y to the question, the system then continues with a normal dialog with the remote system, which would be for the remote system to send the "banner" message followed by a request for concurrent mode operation. If N is the response, the system asks:

Terminate CSO conversation (Y/N)?

A positive response to this question causes the system to reenter transparent mode and prompt:

Manually hang up the modem and when you are done, exit the transparent mode  
Escape character: \$01 = ^A

The system is now in normal operation, and the menu is redisplayed.

Note that in manual mode of operation, transparent mode refers to the connection between the user terminal and the modem for manual modem control, and concurrent mode refers to the concurrent operation of a modem connected terminal and the system console.

### Terminal Mode Operation

Operation with the terminal mode selected from the prompt string "Is the Modem --" is in most ways identical to other connection modes, except that after the prompt to allow change of baud rate, the system automatically enters concurrent mode. Additionally, exiting concurrent mode does not give prompts and messages referring to the hang up sequence. All other system operation is the the same as other modes of connection.

### **Display System Test Errors**

This menu selection displays any errors accumulated by the extended confidence test suite when last run. This can be a useful field service tool.

### **Dump Memory to Tape**

This selection creates an image of the system area of memory on a streaming tape if the prerequisite controller is attached. The option works only with QIC-2 devices as used with the Motorola MVME350 controller. Latest versions of SYSTEM V/68 operating system "crash" utilities do not utilize the results of this tape image.

This completes the description of system mode operation of the MVME141Bug.

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX B - DEBUGGING PACKAGE MESSAGES

The following tables list the debugging package error messages.

DEBUGGER ERROR MESSAGES	MEANING
Error Status: XXXX	Disk communication error status word when IOP command, or .DSKRD or .DSKWR TRAP #15 functions, are unsuccessful. Refer to Appendix F for details.
*** Illegal argument ***	Improper argument in known command.
Invalid command	Unknown command.
Invalid LUN	Controller and device selected during IOP or IOT command do not correspond to a valid controller and device.
*** Invalid Range ***	Range entered wrong in BF, BI, BM, BS, or DU commands.
Long Bus Error	Message displayed when using an unassigned or reserved function code or mnemonic.
<i>part of S-record data</i>	Printed out if non-hex character is encountered in data field in LO or VE commands.
RAM FAIL AT \$XXXXXXXX	Parity is not correct at address \$XXXXXXXX during a BI command.
*STATUS* No error since start of program Upload of S-Records complete.	Message from VERSAdos UPLOADS utility after successful DU command.
The following record(s) did not verify .... SNXXYYYYAAAA.....ZZ.....CS	Failure during the LO or VE commands. ZZ is the non-matching byte and CS is the non-matching checksum.
<i>unassembled line</i> -----^	Message and pointer ("^") to field of suspected error when using ;DI option in MM command.
*** Unknown Field ***	
Verify passes	Successful VE command.

DEBUGGING PACKAGE MESSAGES

DIAGNOSTIC ERROR MESSAGES	MEANING
Addr=XXXXXXXX Expect=YYYYYYYY Read=ZZZZZZZ	Error message in all MMU tests except A, B, and 0. XXXXXXXX, YYYYYYYY, and ZZZZZZZ are hex numbers.
Battery low (data may be corrupted)	Power-up test error message.
N CACHE (HITS!/MISSES!) CACHED IN XXXX MODE, RERAN IN XXXX MODE ..... FAILED	MC68030 Cache Tests error message, where N is a number and XXXX is SUPY or USER.
CPU Addressing Modes test failed	Power-up test error message.
CPU Instruction test failed	Power-up test error message.
CPU Register test failed	Power-up test error message.
Date read was xx/xx/xx, should be	01/01/00 RTC Test error message.
Day of week not 1	RTC Test error message.
Exception Processing test failed	Power-up test error message.
Expect=XXXXXXXX Read=YYYYYYYY	Error message in MMU A or B tests. XXXXXXXX and YYYYYYYY are hex numbers.
FAILED	Error message in non-verbose (NV) mode.
Failed <i>name</i> addressing check	MPU Address Mode Test error message. <i>name</i> is the particular addressing mode(s) whose test(s) failed.
Failed <i>name</i> instruction check	MPU Instruction Test error message. <i>name</i> is the particular instruction(s) whose test(s) failed.
Failed <i>name</i> register check	MPU Register Test error message. <i>name</i> is the particular register(s) whose test(s) failed.
FC TEST ADDR 10987654321098765432109876543210	EXPECTED READ
N NNNNNNNN -----X-X-----	NNNNNNNN NNNNNNNN
	Error message display format for Memory Tests E - J, where the N's are numbers.

DIAGNOSTIC ERROR MESSAGES	MEANING
Got Bus Error when reading from ROM	Bus Error Test error message.
Hit page .....	Error messages in MMU 0 test.
Missed page .....	
Modified page .....	
Insufficient Memory PASSED	Memory Test I Program Test error message when the range of memory selected is less than 388 bytes and the program segment cannot be copied into RAM.
No Bus Error when (writing to/reading from) BAD address space	Bus Error Test error message.
No FPC detected	FPC Test error message when there is no FPC on the module.
MMU does not respond	MMU Test error message when the MMU fails/does not respond.
Non-volatile RAM access error PASSED	Power-up test error message. Successful test message in non-verbose (NV) mode.
RAM test failed	Power-up test error message.
ROM test failed	Power-up test error message.
Test failed FPC routine at \$XXXXXXXX	FPC Test error message. \$XXXXXXXX is address of part of test that failed.
Test Failed Vector # XXX	MPU Exception Processing Test error message. # XXX is the exception vector offset.
Time read was xx:xx:xx, should be 00:00:01	RTC Test error message.
Unexpected Bus Error	MPU Address Mode, MFP Functionality, RTC, or Z8530 Functionality Test error message.
Unexpected exception taken to Vector # XXX	MPU Exception Processing Test error message. # XXX is the exception vector offset.
Unexpected interrupt	FPC Test error message.

OTHER MESSAGES	MEANING
141-Bug>	Debugger prompt.
141-Diag>	Diagnostic prompt.
At Breakpoint	Indicates program has stopped at breakpoint.
Auto Boot from controller X, device Y, STRING	Message when Autoboot is enabled by AB command. X and Y are hex numbers; STRING is an ASCII string.
Autoboot in progress... To Abort hit (BREAK)	If Autoboot is enabled, this message is displayed at Power-Up informing user that Autoboot has begun.
!!Break!!	BREAK key on console has stopped operation.
(Clock is in Battery Save Mode)	Message output when PS command halts the RTC oscillator.
COLD Start	Vectors have been initialized.
Data = \$XX	XX is truncated data cut to fit data field size during BF or BV commands.
Effective address: XXXXXXXX	Exact location of data during BF, BI, BM, BS, BV, DU, and EEP commands; or where program was executed during GD, GN, GO, and GT commands.
Effective count : &XXX	Actual number of data patterns acted on during BF, BI, BS, BV, or EEP commands; or the number of bytes moved during DU command.
Escape character: \$HH=AA	Exit code from transparent mode, in hex (HH) and ASCII (AA) during TM command.
Initial data = \$XX, increment = \$YY	XX is starting data and YY is truncated increment cut to fit data field size during BF or BV commands.
-last match extends over range boundary-	String found in BS command ends outside specified range.

OTHER MESSAGES	MEANING
Logical unit \$XX unassigned	Message that may be output during PA or PF commands. \$XX is a hex number indicating the port involved.
No Auto Boot from controller X, device Y, STRING	Message when Autoboot is disabled by NORB command. X and Y are hex numbers; STRING is an ASCII string.
No printer attached -not found-	Message that may be output during NOPA command. String not found in BS command.
OK to proceed (y/n)?	"Interlock" prompt before configuring port in PF command.
Press "RETURN" to continue	Message output during BS or HE command when more than 24 lines of output are available.
ROM boot disabled	Message output when NORB command disables ROMboot function.
UPLOAD "S" RECORDS Version x.y Copyrighted by MOTOROLA, INC.	Message from VERSAdos UPLOADS utility during DU command.
volume=xxxx catlg=xxxx file=FILE1 ext=MX	
WARM Start	Vectors have not been initialized.
Weekday xx/xx/xx xx:xx:xx	Day, date, and 24-hour time presentation during SET and TIME commands.

B

3

THIS PAGE INTENTIONALLY LEFT BLANK.

## APPENDIX C - S-RECORD OUTPUT FORMAT

The S-record format for output modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can thus be visually monitored and the S-records can be more easily edited.

S-RECORD CONTENT

When viewed by the user, S-records are essentially character strings made of several fields which identify the record type, record length, memory address, code/data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The five fields which comprise an S-record are shown below:

type	record length	address	code/data	checksum
------	---------------	---------	-----------	----------

where the fields are composed as follows:

<u>FIELD</u>	<u>PRINTABLE CHARACTERS</u>	<u>CONTENTS</u>
type	2	S-record type -- S0, S1, etc.
record length	2	The count of the character pairs in the record, excluding the type and record length.
address	4, 6, or 8	The 2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
code/data	0-2n	From 0 to n bytes of executable code, memory-loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record).
checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing system.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

### S-RECORD TYPES

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other record transportation control programs, as well as cross assemblers, linkers, and other file-creating or debugging programs, utilize only those S-records which serve the purpose of the program. For specific information on which S-records are supported by a particular program, the user's manual for that program must be consulted. This Bug supports S0, S1, S2, S3, S7, S8, and S9 records.

An S-record-format module may contain S-records of the following types:

- S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. Under VERSAdos, the resident linker's IDENT command can be used to designate module name, version number, revision number, and description information which will make up the header record. The address field is normally zeroes.
- S1 A record containing code/data and the 2-byte address at which the code/data is to reside.
- S2 A record containing code/data and the 3-byte address at which the code/data is to reside.
- S3 A record containing code/data and the 4-byte address at which the code/data is to reside.
- S5 A record containing the number of S1, S2, and S3 records transmitted in a particular block. This count appears in the address field. There is no code/data field.
- S7 A termination record for a block of S3 records. The address field may optionally contain the 4-byte address of the instruction to which control is to be passed. There is no code/data field.

- S8 A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.
- S9 A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. Under VERSAdos, the resident linker's ENTRY command can be used to specify this address. If not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. S7 and S8 records are usually used only when control is to be passed to a 3- or 4-byte address. Normally, only one header record is used, although it is possible for multiple header records to occur.

### CREATION OF S-RECORDS

S-record-format programs may be produced by several dump utilities, debuggers, VERSAdos' resident linkage editor, or several cross assemblers or cross linkers. On VERSAdos, the Build Load Module (MBLM) utility allows an executable load module to be built from S-records, and has a counterpart utility in BUILDSD, which allows an S-record file to be created from a load module.

Several programs are available for downloading a file in S-record format from a host system to an 8-bit microprocessor-based or a 16-bit microprocessor-based system.

### EXAMPLE

Shown below is a typical S-record-format module, as printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The module consists of one S0 record, four S1 records, and an S9 record.

The S0 record is comprised of the following character pairs:

S0 S-record type S0, indicating that it is a header record.  
 06 Hexadecimal 06 (decimal 6), indicating that six character pairs (or ASCII bytes) follow.  
 00 Four-character 2-byte address field, zeroes in this example.  
 00  
 48  
 44 ASCII H, D, and R - "HDR".  
 52  
 1B The checksum.

The first S1 record is explained as follows:

S1 S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.  
 13 Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.  
 00 Four-character 2-byte address field; hexadecimal address 0000, where the data which follows is to be loaded.  
 00

The next 16 character pairs of the first S1 record are the ASCII bytes of the actual program code/data. In this assembly language example, the hexadecimal opcodes of the program are written in sequence in the code/data fields of the S1 records:

<u>OPCODE</u>	<u>INSTRUCTION</u>
285F	MOVE.L (A7)+,A4
245F	MOVE.L (A7)+,A2
2212	MOVE.L (A2),D1
226A0004	MOVE.L 4(A2),A1
24290008	MOVE.L FUNCTION(A1),D2
237C	MOVE.L #FORCEFUNC,FUNCTION(A1)

. (The balance of this code is continued in the  
 . code/data fields of the remaining S1 records,  
 . and stored in memory location 0010, etc.)

2A The checksum of the first S1 record.

The second and third S1 records each also contain \$13 (19) character pairs and are ended with checksums 13 and 52, respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The S9 record is explained as follows:

- S9 S-record type S9, indicating that it is a termination record.
- 03 Hexadecimal 03, indicating that three character pairs (3 bytes) follow.
- 00 The address field, zeroes.  
00
- FC The checksum of the S9 record.

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted. For example, the first S1 record above is sent as:

THIS PAGE INTENTIONALLY LEFT BLANK.

## APPENDIX D - INFORMATION USED BY BO AND BH COMMANDS

## VOLUME ID BLOCK #0 (VID)

LABEL	OFFSET\$(&)	LENGTH (BYTES)	CONTENTS
VIDOSS	\$14 (20)	4	Starting block number of operating system.
VIDOSL	\$18 (24)	2	Operating system length in blocks.
VIDOSA	\$1E (30)	4	Starting memory location to load operating system.
VIDCAS	\$90 (144)	4	Media configuration area starting block.
VIDCAL	\$94 (148)	1	Media configuration area length in blocks.
VIDMOT	\$F8 (248)	8	Contains the string "MOTOROLA" or "EXORMACS".

## CONFIGURATION AREA BLOCK #1 (CFGA)

LABEL	OFFSET\$(&)	LENGTH (BYTES)	CONTENTS
IOSATM	\$04 (4)	2	Attributes mask.
IOSPRM	\$06 (6)	2	Parameters mask.
IOSATW	\$08 (8)	2	Attributes word.
IOSREC	\$0A (10)	2	Record (block) size in bytes.
IOSSPT	\$18 (24)	1	Sectors/track.
IOSHDS	\$19 (25)	1	Number of heads on drive.
IOSTRK	\$1A (26)	2	Number of cylinders.
IOSILV	\$1C (28)	1	Interleave factor on media.
IOSSOF	\$1D (29)	1	Spiral offset.
IOSPSM	\$1E (30)	2	Physical sector size of media in bytes.
IOSSHD	\$20 (32)	2	Starting head number.
IOSPCOM	\$24 (36)	2	Precompensation cylinder.
IOSSR	\$27 (39)	1	Stepping rate code.

## CONFIGURATION AREA BLOCK #1 (CFGA) (cont'd)

LABEL	OFFSET\$(&)	LENGTH (BYTES)	CONTENTS
IOSRWCC	\$28 (40)	2	Reduced write current cylinder number.
IOSECC	\$2A (42)	2	ECC data burst length.
IOSEATM	\$2C (44)	2	Extended attributes mask.
IOSEPRM	\$2E (46)	2	Extended parameters mask.
IOSEATW	\$30 (48)	2	Extended attributes word.
IOGSPB1	\$32 (50)	1	Gap byte 1.
IOGSPB2	\$33 (51)	1	Gap byte 2.
IOGSPB3	\$34 (52)	1	Gap byte 3.
IOGSPB4	\$35 (53)	1	Gap byte 4.
IOSSSC	\$36 (54)	1	Spare sectors count.
IOSRUNIT	\$37 (55)	1	Reserved area units.
IOSRSVC1	\$38 (56)	2	Reserved count 1.
IOSRSVC2	\$3A (58)	2	Reserved count 2.

## IOSATM and IOSEATM

A "1" in a particular bit position indicates that the corresponding attribute from the attributes (or extended attributes) word should be used to update the configuration. A "0" in a bit position indicates that the current attribute should be retained.

## IOSATM ATTRIBUTE MASK BIT DEFINITIONS

LABEL	BIT POSITION	DESCRIPTION
IOADDEN	0	Data density.
IOATDEN	1	Track density.
IOADSIDE	2	Single/double sided.
IOAFRMT	3	Floppy disk format.
IOARDISC	4	Disk type.
IOADDEND	5	Drive data density.
IOATDEND	6	Drive track density.
IOARIBS	7	Embedded servo drive seek.
IOADPCOM	8	Post-read/pre-write precompensation.
IOASIZE	9	Floppy disk size.
IOATKZD	13	Track zero data density.

At the present, all IOSEATM bits are undefined and should be set to 0.

## IOSPRM and IOSEPRM

A "1" in a particular bit position indicates that the corresponding parameter from the configuration area (CFG) should be used to update the device configuration. A "0" in a bit position indicates that the parameter value in the current configuration will be retained.

## IOSPRM PARAMETER MASK BIT DEFINITIONS

LABEL	BIT POSITION	DESCRIPTION
IOSRECB	0	Operating system block size.
IOSSPTB	4	Sectors per track.
IOSHDSB	5	Number of heads.
IOSTRKB	6	Number of cylinders.
IOSILVB	7	Interleave factor.
IOSSOFB	8	Spiral offset.
IOSPSMB	9	Physical sector size.
IOSSHDB	10	Starting head number.
IOSPCOMB	12	Precompensation cylinder number.
IOSSRB	14	Step rate code.
IOSRWCCB	15	Reduced write current cylinder number and ECC data burst length.

## IOSEPRM PARAMETER MASK BIT DEFINITIONS

LABEL	BIT POSITION	DESCRIPTION
IOAGPB1	0	Gap byte 1.
IOAGPB2	1	Gap byte 2.
IOAGPB3	2	Gap byte 3.
IOAGPB4	3	Gap byte 4.
IOASSC	4	Spare sector count.
IOARUNIT	5	Reserved area units.
IOARVC1	6	Reserved count 1.
IOARVC2	7	Reserved count 2.

## IOSATW and IOSEATW

Contains various flags that specify characteristics of the media and drive.

## IOSATW BIT DEFINITIONS

BIT NUMBER	DESCRIPTION
Bit 0	Data density: 0 = Single density (FM encoding) 1 = Double density (MFM encoding)
Bit 1	Track density: 0 = Single density (48 TPI) 1 = Double density (96 TPI)
Bit 2	Number of sides: 0 = Single sided floppy 1 = Double sided floppy
Bit 3	Floppy disk format: 0 = Motorola format (sector numbering) 1 to N on side 0 N+1 to 2N on side 1 1 = Standard IBM format 1 to N on both sides
Bit 4	Disk type: 0 = Floppy disk 1 = Hard disk
Bit 5	Drive data density: 0 = Single density (FM encoding) 1 = Double density (MFM encoding)
Bit 6	Drive track density: 0 = Single density 1 = Double density
Bit 7	Embedded servo drive: 0 = Do not seek on head switch 1 = Seek on head switch
Bit 8	Post-read/pre-write precompensation: 0 = Pre-write 1 = Post-read
Bit 9	Floppy disk size: 0 = 5-1/4 inch floppy 1 = 8-inch floppy
Bit 13	Track zero density: 0 = Single density (FM encoding) 1 = Same as remaining tracks
Unused bits	All unused bits must be set to 0.

At the present, all IOSEATW bits are undefined and should be set to 0.

## PARAMETER FIELD DEFINITIONS

PARAMETER	DESCRIPTION
Record (Block) size	Number of bytes per record (block). Must be an integer multiple of the physical sector size.
Sectors/track	Number of sectors per track.
Number of heads	Number of recording surfaces for the specified device.
Number of cylinders	Number of cylinders on the media.
Interleave factor	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.
Physical sector size	Actual number of bytes per sector on media.
Spiral offset	Used to displace the logical start of a track from the physical start of a track. The displacement is equal to the spiral offset times the head number, assuming that the first head is 0. This displacement is used to give the controller time for a head switch when crossing tracks.
Starting head number	Defines the first head number for the device.
Precompensation cylinder	Defines the cylinder on which precompensation begins.
Stepping rate code	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	Winchester Hard Disks	5-1/4 inch Floppy	8 inch Floppy
000	0 msec	12 msec	6 msec
001	6 msec	6 msec	3 msec
010	10 msec	12 msec	6 msec
011	15 msec	20 msec	10 msec
100	20 msec	30 msec	15 msec

## PARAMETER FIELD DEFINITIONS (cont'd)

PARAMETER	DESCRIPTION
Reduced write current cylinder	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.
ECC data burst length	This field defines the number of bits to correct for an ECC error when supported by the disk controller.
Gap byte 1	This field contains the number of words of zeros that are written before the header field in each sector during format.
Gap byte 2	This field contains the number of words of zeros that are written between the header and data fields during format and write commands.
Gap byte 3	This field contains the number of words of zeros that are written after the data fields during format commands.
Gap byte 4	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	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.
Reserved area units	This field specifies the units used for the next two fields (IOSRSVC1 and IOSRSVC2). If zero, the units are in <u>tracks</u> ; if 1, the units are in <u>cylinders</u> .
Reserved count 1	This field specifies the number of tracks (IOSRUNIT = 0), or the number of cylinders (IOSRUNIT = 1) reserved for the alternate mapping area on the disk.
Reserved count 2	This field specifies the number of tracks (IOSRUNIT = 0), or the number of cylinders (IOSRUNIT = 1) reserved for use by the controller.

## APPENDIX E - DISK CONTROLLER DATA

## DISK CONTROLLER MODULES SUPPORTED

The following VMEbus disk/tape controller modules are supported by 141Bug. The default address for each type of controller is FIRST ADDR and the controller can be addressed by FIRST CLUN during commands BH, BO, or IOP, or during TRAP #15 calls .DSKRD or .DSKWR. Note that if another one of the same type of controller is used, the second one must have its address changed by its onboard jumpers and/or switches, so that it matches SECOND ADDR and can be called up by SECOND CLUN. Additionally, if an MVME319, MVME320, and/or MVME321 are used together, the 141Bug firmware automatically assigns one of them to its default conditions (FIRST CLUN and FIRST ADDR), and also automatically assigns the other(s) to the next available higher CLUN. The priority for assigning CLUN \$00 is: MVME321, MVME320, MVME319, then MVME323.

CONTROLLER TYPE	FIRST CLUN FIRST ADDR	SECOND CLUN SECOND ADDR
MVME319 - SCSI/Floppy/Tape Controller	\$00 \$FFFF0000	\$07 \$FFFF0200
MVME320 - Winchester/Floppy Controller	\$00 \$FFFFB000	\$06 \$FFFFAC00
MVME321 - Winchester/Floppy Controller	\$00 \$FFFF0500	\$01 \$FFFF0600
MVME323 - ESDI Controller	\$08 or greater \$FFFA000	\$08 or greater \$FFFA200
MVME350 - Streaming Tape Controller	\$04 \$FFFF5000	\$05 \$FFFF5100
MVME360 - SMD Controller	\$02 \$FFFF0C00	\$03 \$FFFF0E00

## DISK CONTROLLER DEFAULT CONFIGURATIONS

SYSTEM V/68 or VERSAdos

## Controller LUN 8 or greater

Controller Type : MVME319  
 Controller Address : \$FFFF0000  
 Number of Devices : 8  
 Devices : DLUN 0 = 40Mb Winchester hard drive (NOTE) WIN40  
           DLUN 1 = 40Mb Winchester hard drive (NOTE) WIN40  
           DLUN 2 = 40Mb Winchester hard drive (NOTE) WIN40  
           DLUN 3 = 40Mb Winchester hard drive (NOTE) WIN40  
           DLUN 4 = 8" DS/SD Motorola format floppy drive FLP8  
           DLUN 5 = 8" DS/SD Motorola format floppy drive FLP8  
           DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive FLP5  
           DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive FLP5

## Controller LUN 8 or greater

Controller Type : MVME319  
 Controller Address : \$FFFF0200  
 Number of Devices : 8  
 Devices : DLUN 0 = 40Mb Winchester hard drive (NOTE) WIN40  
           DLUN 1 = 40Mb Winchester hard drive (NOTE) WIN40  
           DLUN 2 = 40Mb Winchester hard drive (NOTE) WIN40  
           DLUN 3 = 40Mb Winchester hard drive (NOTE) WIN40  
           DLUN 4 = 8" DS/SD Motorola format floppy drive FLP8  
           DLUN 5 = 8" DS/SD Motorola format floppy drive FLP8  
           DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive FLP5  
           DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive FLP5

NOTE

Devices 0 through 3 are accessed via the SCSI interface on the MVME319. An ADAPTEC ACB-4000 Winchester Disk Controller module is required to interface between the SCSI and the disk drive. Refer to the MVME319 User's Manual for further information.

## Controller LUN 8 or greater

Controller Type : MVME320  
 Controller Address : \$FFFFB000  
 Number of Devices : 4  
 Devices : DLUN 0 = 40Mb Winchester hard drive WIN40  
           DLUN 1 = 40Mb Winchester hard drive WIN40  
           DLUN 2 = 5-1/4" DS/DD 96 TPI floppy drive FLP5  
           DLUN 3 = 5-1/4" DS/DD 96 TPI floppy drive FLP5

## DISK CONTROLLER DATA

## Controller LUN 8 or greater

Controller Type : MVME320  
 Controller Address : \$FFFFAC00  
 Number of Devices : 4  
 Devices

DLUN 0 = 40Mb Winchester hard drive	WIN40
DLUN 1 = 40Mb Winchester hard drive	WIN40
DLUN 2 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
DLUN 3 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5

## Controller LUN 8

Controller Type : MVME321  
 Controller Address : \$FFFF0500  
 Number of Devices : 8  
 Devices

DLUN 0 = 40Mb Winchester hard drive	WIN40
DLUN 1 = 40Mb Winchester hard drive	WIN40
DLUN 2 = 40Mb Winchester hard drive	WIN40
DLUN 3 = 40Mb Winchester hard drive	WIN40
DLUN 4 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
DLUN 5 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5

## Controller LUN 8 or greater

Controller Type : MVME321  
 Controller Address : \$FFFF0600  
 Number of Devices : 8  
 Devices

DLUN 0 = 40Mb Winchester hard drive	WIN40
DLUN 1 = 40Mb Winchester hard drive	WIN40
DLUN 2 = 40Mb Winchester hard drive	WIN40
DLUN 3 = 40Mb Winchester hard drive	WIN40
DLUN 4 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
DLUN 5 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5

## Controller LUN 8 or greater

Controller Type : MVME323  
 Controller Address : \$FFFA000  
 Number of Devices : 1  
 Devices

DLUN 0 = 182Mb CDC WREN III (512b sectors)	WREN
DLUN 1 = 182Mb CDC WREN III (512b sectors)	WREN
DLUN 2 = 182Mb CDC WREN III (512b sectors)	WREN
DLUN 3 = 182Mb CDC WREN III (512b sectors)	WREN

## Controller LUN 8 or greater

Controller Type : MVME323  
 Controller Address : \$FFFA200  
 Number of Devices : 1  
 Devices

DLUN 0 = 182Mb CDC WREN III (512b sectors)	WREN
DLUN 1 = 182Mb CDC WREN III (512b sectors)	WREN
DLUN 2 = 182Mb CDC WREN III (512b sectors)	WREN
DLUN 3 = 182Mb CDC WREN III (512b sectors)	WREN

## DISK CONTROLLER DATA

## Controller LUN 8 or greater

Controller Type : MVME350  
 Controller Address : \$FFFF5000  
 Number of Devices : 1  
 Devices : DLUN 0 = QIC-02 Streaming Tape Drive

## Controller LUN 8 or greater

Controller Type : MVME350  
 Controller Address : \$FFFF5100  
 Number of Devices : 1  
 Devices : DLUN 0 = QIC-02 Streaming Tape Drive

## Controller LUN 8 or greater

Controller Type : MVME360  
 Controller Address : \$FFFF0C00  
 Number of Devices : 4 (NOTE)  
 Devices : DLUN 0 = 2333K Fujitsu SMD drive (512b sectors) FJI20  
           DLUN 1 = null device (SMD half) SMDHALF  
           DLUN 2 = 2333K Fujitsu SMD drive (512b sectors) FJI10  
           DLUN 3 = null device (SMD half) SMDHALF

## Controller LUN 8 or greater

Controller Type : MVME360  
 Controller Address : \$FFFF0E00  
 Number of Devices : 4 (NOTE)  
 Devices : DLUN 0 = 2322K Fujitsu SMD drive (256b sectors) FJI10V  
           DLUN 1 = null device (SMD half) SMDHALF  
           DLUN 2 = 80Mb fixed CMD drive FXCMD80  
           DLUN 3 = 16Mb removable CMD drive RMCMD16

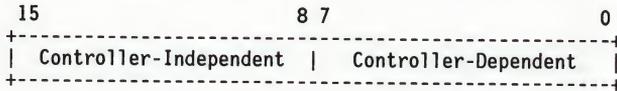
NOTE

Only two physical SMD drives may be connected to an MVME360 controller, but the drive may be given two DLUNs, as is the case for Controller LUN 3 above.

DISK COMMUNICATION STATUS CODES

APPENDIX F - DISK COMMUNICATION STATUS CODES

The status word returned by the disk TRAP #15 routines flags an error condition if it is nonzero. The most significant byte of the status word reflects controller independent errors, and they are generated by the disk trap routines. The least significant byte reflects controller dependent errors, and they are generated by the controller. The status word is shown below:



=====  
Controller-Independent Status Codes  
=====

- \$00 No error detected.
  - \$01 Invalid controller type.
  - \$02 Invalid controller LUN.
  - \$03 Invalid device LUN.
  - \$04 Controller initialization failed.
  - \$05 Command aborted via break.
  - \$06 Invalid command packet.
  - \$07 Invalid address for transfer.
  - \$50 Block conversion error.
- =====

F

DISK COMMUNICATION STATUS CODES

=====

MVME319 Controller-Dependent Status Codes

=====

\$00	Correct execution without error.
\$01	Data CRC/ECC error.
\$02	Disk write protected.
\$03	Drive not ready.
\$04	Deleted data mark read.
\$05	Invalid drive number.
\$06	Invalid disk address.
\$07	Restore error.
\$08	Record not found.
\$09	Sector ID CRC/ECC error.
\$0A	VMEbus DMA error.
\$0F	Controller error.
\$10	Drive error.
\$11	Seek error.
\$19	I/O DMA error.

=====

=====

MVME320 Controller-Dependent Status Codes

=====

\$00	Correct execution without error.
\$01	Nonrecoverable error which cannot be completed (auto retries were attempted).
\$02	Drive not ready.
\$03	Reserved.
\$04	Sector address out of range.
\$05	Throughput error (floppy data overrun).
\$06	Command rejected (illegal command).
\$07	Busy (controller busy).
\$08	Drive not available (head out of range).
\$09	DMA operation cannot be completed (VMEbus error).
\$0A	Command abort (reset busy).
\$0B-\$FF	Not used.

=====

```
=====
MVME321 Controller-Dependent Status Codes
=====
```

```
*** General Error Codes ***
```

```
$00 Correct execution without error.
$17 Time-out.
$18 Bad drive.
$1A Bad command.
$1E Fatal error.
```

```
*** Hard Disk Error Codes ***
```

```
$01 Write protected disk.
$02 Sector not found.
$03 Drive not ready.
$04 Drive fault or time-out on recalibrate.
$05 CRC or ECC error in data field.
$06 UPD7261 FIFO overrun/underrun.
$07 End of cylinder.
$08 Illegal drive specified.
$09 Illegal cylinder specified.
$0A Format operation failed.
$0B Bad disk descriptor.
$0C Alternate track error.
$0D Seek error.
$0E UPD7261 busy.
$0F Data does not verify.
$10 CRC error in ID field.
$11 Reset request (missing address mark).
$12 Correctable ECC error.
$13 Abnormal command completion.
$20 Missing data mark.
```

```
=====
MVME321 Controller-Dependent Status Codes (cont'd)
=====
```

```
*** Floppy Disk Error Codes ***
```

```
$01 End-of-transfer size mismatch
$02 Bad TPI combination specified.
$03 Drive motor not coming on.
$04 Disk door open.
$05 Command not completing.
$06 Bad restore operation.
$07 Illegal side reference on device.
$08 Illegal track reference on device.
$09 Illegal sector reference on device.
$0A Illegal step rate specified.
$0B Bad density specified.
$0C Write protected disk.
$0D Format error.
$0E Can't find side, track, or sector.
$0F CRC error in ID field(s).
$10 CRC error in data field.
$11 DMA underrun.
$20 Bad disk size in descriptor.
```

## DISK COMMUNICATION STATUS CODES

 =====  
 MVME323 Controller-Dependent Status Codes  
 =====

\$00	Correct execution without error.
\$10	Disk not ready.
\$11	Not used.
\$12	Seek error.
\$13	ECC code error-data field.
\$14	Invalid command code.
\$15	Illegal fetch and execute command.
\$16	Invalid sector in command.
\$17	Illegal memory type.
\$18	Bus time-out.
\$19	Header checksum error.
\$1A	Disk write protected.
\$1B	Unit not selected.
\$1C	Seek error time-out.
\$1D	Fault time-out.
\$1E	Drive faulted.
\$1F	Ready time-out.
\$20	End of medium.
\$21	Translation fault.
\$22	Invalid header pad.
\$23	Uncorrectable error.
\$24	Translation error, cylinder.
\$25	Translation error, head.
\$26	Translation error, sector.
\$27	Data overrun.
\$28	No index pulse on format.
\$29	Sector not found.
\$2A	ID field error -- wrong head.
\$2B	Invalid sync in data field.
\$2C	No valid header found.
\$2D	Seek time-out error.
\$2E	Busy time-out.
\$2F	Not on cylinder.
\$30	RTZ time-out.
\$31	Invalid sync in header.
\$32-3F	Not used.
\$40	Unit not initialized.
\$41	Not used.
\$42	Gap specification error.
\$43-4A	Not used.
\$4B	Seek error.
\$4C-4F	Not used.
\$50	Sectors per track error.
\$51	Bytes per sector specification error.
\$52	Interleave specification factor.
\$53	Invalid head address.
\$54	Invalid cylinder address.
\$55-5C	Not used.
\$5D	Invalid DMA transfer count.

## DISK COMMUNICATION STATUS CODES

## =====

## MVME323 Controller-Dependent Status Codes (cont'd)

=====

\$5E-5F	Not used.
\$60	IOPB failed.
\$61	DMA failed.
\$62	Illegal VME address.
\$63-69	Not used.
\$6A	Unrecognized header field.
\$6B	Mapped header error.
\$6C-6E	Not used.
\$6F	No spare sector enabled.
\$70-76	Not used.
\$77	Command aborted.
\$78	ACFAIL detected.
\$79-EF	Not used.
\$F0-FE	Unforeseen error - call your Field Service representative, and tell them the IOPB and UIB information that was available at the time that the error occurred.
\$FF	Command not implemented.

=====

## =====

## MVME350 Controller-Dependent Status Codes

=====

\$00	Correct execution without error.
\$01	Block in error not located.
\$02	Unrecoverable data error.
\$03	End of media.
\$04	Write protected.
\$05	Drive offline.
\$06	Cartridge not in place.
\$0D	No data detected.
\$0E	Illegal command.
\$12	Tape reset did not occur.
\$17	Time-out.
\$18	Bad drive.
\$1A	Bad command.
\$1E	Fatal error.

=====

## DISK COMMUNICATION STATUS CODES

-----  
MVME360 Controller-Dependent Status Codes  
-----

\$00	Correct execution without error.
\$10	Disk not ready.
\$11	Not used.
\$12	Seek error.
\$13	ECC code error-data field.
\$14	Invalid command code.
\$15	Illegal fetch and execute command.
\$16	Invalid sector in command.
\$17	Illegal memory type.
\$18	Bus time-out.
\$19	Header checksum error.
\$1A	Disk write protected.
\$1B	Unit not selected.
\$1C	Seek error time-out.
\$1D	Fault time-out.
\$1E	Drive faulted.
\$1F	Ready time-out.
\$20	End of medium.
\$21	Translation fault.
\$22	Invalid header pad.
\$23	Uncorrectable error.
\$24	Translation error, cylinder.
\$25	Translation error, head.
\$26	Translation error, sector.
\$27	Data overrun.
\$28	No index pulse on format.
\$29	Sector not found.
\$2A	ID field error -- wrong head.
\$2B	Invalid sync in data field.
\$2C	No valid header found.
\$2D	Seek time-out error.
\$2E	Busy time-out.
\$2F	Not on cylinder.
\$30	RTZ time-out.
\$31	Invalid sync in header.
\$32-3F	Not used.
\$40	Unit not initialized.
\$41	Not used.
\$42	Gap specification error.
\$43-4A	Not used.
\$4B	Seek error.
\$4C-4F	Not used.
\$50	Sectors per track error.
\$51	Bytes per sector specification error.
\$52	Interleave specification factor.
\$53	Invalid head address.
\$54	Invalid cylinder address.
\$55-5C	Not used.
\$5D	Invalid DMA transfer count.

DISK COMMUNICATION STATUS CODES

=====

MVME360 Controller-Dependent Status Codes (cont'd)

=====

\$5E-5F Not used.  
\$60 IOPB failed.  
\$61 DMA failed.  
\$62 Illegal VME address.  
\$63-69 Not used.  
\$6A Unrecognized header field.  
\$6B Mapped header error.  
\$6C-6E Not used.  
\$6F No spare sector enabled.  
\$70-76 Not used.  
\$77 Command aborted.  
\$78 ACFAIL detected.  
\$79-EF Not used.  
\$F0-FE Unforseen error - call your Field  
Service representative, and tell them  
the IOPB and UIB information that was  
available at the time that the error  
occurred.  
\$FF Command not implemented.

=====

DISK COMMUNICATION STATUS CODES

THIS PAGE INTENTIONALLY LEFT BLANK.

## INDEX

147-Bug, 1-1  
147-Diag, 1-1  
147Bug Assembler Addressing Modes, 4-7  
147Bug Generalized Exception Handler, 2-12  
147Bug system console, 1-5, 6-2  
147Bug Vector Table and Wordspace, 2-8

## A

AB command, 3-3  
ABORT switch, 1-5, 6-2  
Address, 2-3  
Address as a Parameter, 2-4  
address formats, 2-4  
Addressing Modes, 4-6, 4-8  
Allowed operators, 4-7  
arithmetic operators, 2-3  
ASCII string, 2-3  
Assembler Addressing Modes, 4-7  
Assembler Output/Program Listings, 4-12  
assembler/disassembler, 4-10  
assembler/disassembler function, 4-1  
AUTOBOOT, 1-6

## B

backspace, 2-2  
Battery Backed Up RAM, 1-11 (see also BBRAM)  
Battery low (data may be corrupted), 1-6, 6-3  
baud rate, 1-5, 6-2, A-5  
baud rate test command (SIO B), 6-77  
BBRAM, 1-11 (see also Battery Backed Up RAM)  
BC command, 3-4  
BF command, 3-6  
BH Bootstrap and Halt, 1-15  
BH command, 3-8  
BI command, 3-9  
binary, 2-3  
BINDEC, 5-40  
BM command, 3-10  
BO AND BH COMMANDS, D-1  
BO Bootstrap Operating System, 1-15  
BO command, 3-12  
Boldface brackets, 2-2  
boldface string, 2-2  
BR command, 3-14

BRA, 4-4 (see also branch always)  
 branch always, 4-4 (see also BRA)  
 branch conditionally true, 4-4  
 BRD ID, 5-46  
 BREAK test command (SIO F), 6-82  
 brief parity test command MT K, 6-40  
 BS command, 3-15  
 BT, 4-4  
 bus error test command (BERR), 6-73  
 BV command, 3-18

## C

CA30 A basic data caching test, 6-14  
 CA30 B data cache tag RAM test, 6-15  
 CA30 C data cache data RAM test, 6-17  
 CA30 D data cache valid flags test, 6-18  
 CA30 E data cache burst fill test, 6-19  
 CA30 F basic instruction caching test, 6-20  
 CA30 G unlike instruction function codes test, 6-21  
 CA30 H disable test, 6-22  
 CA30 I clear test, 6-23  
 Calculate BCD equivalent, 5-40  
 CALLING SYSTEM UTILITIES FROM USER PROGRAMS, 2-8  
 cancel line, 2-2  
 carriage return, 2-3  
 CHANGEV, 5-41  
 Character Set, 4-6  
 Check for break, 5-9  
 CHKBRK, 5-9  
 CHK\_SUM, 5-45  
 Clear (Zero) Error Counters - Command ZE, 6-6  
 COLD and WARM reset modes, 1-10  
 Command Entry and Directories, 6-4  
 command identifier, 2-1  
 command line editing, 2-1  
 command parsing, 6-3  
 Compare two strings (pointer/count), 5-42  
 Comparison with MC68030 Resident Structured Assembler, 4-2  
 CONFIGURATION AREA BLOCK 1 (CFGA), D-1  
 Continue System Start Up, A-3  
 control codes, 2-1  
 Controller-Independent Status Codes, F-1  
 Count, 2-3  
 CPU Addressing Modes test failed, 1-6, 6-3  
 CPU Instruction test failed, 1-6, 6-3  
 CPU Register test failed, 1-6, 6-3  
 Creating a New Vector Table, 2-10  
 CS command, 3-20

## D

Data Formats Test, 6-76  
data formats test command (SIO A), 6-76  
Date initialization for RTC, 5-34  
DBF, 4-4 (see also decrement and branch never)  
DBRA, 4-4 (see also decrement and branch always)  
DC command, 3-23  
DC.W Define Constant Directive, 4-9  
DC.W directive, 4-9  
debugger command, 2-1  
Debugger Commands, 3-1  
debugger directory, 1-1, 1-19  
decimal, 2-3  
decrement and branch always, 4-4 (see also DBRA)  
decrement and branch never, 4-4 (see also DBF)  
Default 147Bug Controller and Device Parameters, 1-16  
default input port, 5-1  
default output port, 5-1  
DELAY, 5-32  
delete, 2-2  
Delimiter, 2-3  
DESCRIPTION OF MVME147Bug, 1-1  
diagnostic directory, 1-1, 1-19  
DIAGNOSTIC FACILITIES, 1-19  
diagnostic guide, 6-1  
DIAGNOSTIC MONITOR, 6-3  
Directives, 4-1  
Disassembled Source Line, 4-4  
DISK COMMUNICATION STATUS CODES, F-1  
Disk configure, 1-15  
Disk configure function, 5-13  
Disk control, 1-15  
Disk control function, 5-19  
DISK CONTROLLER DEFAULT CONFIGURATIONS, E-2  
DISK CONTROLLER MODULES SUPPORTED, E-1  
Disk format, 1-15  
Disk format function, 5-17  
Disk I/O Error Codes, 1-16  
DISK I/O SUPPORT, 1-14  
Disk I/O via 147Bug Commands, 1-14  
Disk I/O via 147Bug System Calls, 1-15  
Disk read, 1-15  
Disk read function, 5-10  
Disk write, 1-15  
Disk write function, 5-10  
Display Error Counters - Command DE, 6-6  
Display Pass Count - Command DP, 6-6  
Display System Test Errors, A-9  
Display time from RTC, 5-35

DIVU32, 5-44  
 DSKCFIG, 5-13  
 DSKCTRL, 5-19  
 DSKFMT, 5-17  
 DSKRD, 5-10  
 DSKWR, 5-10  
 DU command, 3-24  
 dual console mode of operation, A-4  
 Dump Memory to Tape, A-9

## E

EEP command, 3-27  
 ellipsis (three dots), 2-2  
 Entering a Source Line, 4-11  
 ENTERING AND DEBUGGING PROGRAMS, 2-7  
 ENTERING AND MODIFYING SOURCE PROGRAMS, 4-10  
 Entering Branch and Jump Addresses, 4-12  
 ENTERING DEBUGGER COMMAND LINES, 2-1  
 ENV command, 3-28  
 Erase Line, 5-25  
 ERASLN, 5-25  
 error messages, B-1  
 error reporting, 6-3  
 Exception Processing test failed, 1-6, 6-3  
 Exception Vectors Used by 147Bug, 2-9  
 Expression, 2-3  
 Expression as a Parameter, 2-3  
 extended confidence tests, A-3  
 extended parity test command MT L, 6-42

## F

FIFO full test command (SIO E), 6-81  
 Floating Point Coprocessor Registers, 4-6  
 floating point coprocessor test command (FPC), 6-74  
 Flow Diagram of 147Bug Normal Operational Mode, 1-2  
 Flow Diagram of 147Bug System Operation Mode, A-2  
 Flow Diagram of 147Bug System Operational Mode, 1-3  
 format/vector, 1-11  
 Formats for Debugger Address Parameters, 2-5  
 fully filled address translation cache (ATC) test command MMU G, 6-52  
 FUNCTION CODE SUPPORT, 2-14

## G

GD command, 3-30  
 generalized exception handler, 2-10, 2-12  
 Generate checksum for address range, 5-45  
 GN command, 3-32  
 GO command, 3-34  
 Go to System Debugger, A-3  
 GT command, 3-36

## H

hardware handshaking, 1-5, 6-2  
 HDS-300, 1-8  
 HDS-400, 1-8  
 HE command, 3-38  
 Help Command HE, 6-4  
 hexadecimal, 2-3  
 Hit Page, 6-68  
 HOW TO USE THIS MANUAL, 1-4

## I

INCHR, 5-4  
 indirect page test command MMU J, 6-55  
 initialize the system  
   Abort, 1-11  
   Break, 1-12  
   Reset, 1-10  
 Initiate service call, A-4  
 INLN, 5-6  
 Input character routine, 5-4  
 Input line routine, 5-6  
 Input serial port status, 5-5  
 input/output facilities, 6-3  
 INSTALLATION AND STARTUP, 1-4  
 INSTAT, 5-5  
 interactive assembler/editor, 4-1  
 interrupt handling, 6-3  
 invalid page test command MMU P, 6-59  
 invalid segment test command MMU Q, 6-60  
 Invoking System Calls Through TRAP 15, 5-1  
 Invoking the Assembler/Disassembler, 4-10  
 IOC command, 3-39  
 IOC I/O Control, 1-15  
 IOP command, 3-40

IOP Physical I/O to Disk, 1-14  
 IOSATM ATTRIBUTE MASK BIT DEFINITIONS, D-3  
 IOSATW BIT DEFINITIONS, D-6  
 IOSEPRM PARAMETER MASK BIT DEFINITIONS, D-5  
 IOSPRM PARAMETER MASK BIT DEFINITIONS, D-4  
 IOT command, 3-43  
 IOT I/O Teach, 1-15  
 italic string, 2-2

## L

label field, 4-3  
 level seven interrupt, 1-11  
 limited confidence test suite, A-1  
 LO command, 3-50  
 logical block, 1-14  
 Loop Back Connector Wiring Diagram, 6-75  
 Loop-Continue Mode - Prefix LC, 6-5  
 Loop-On-Error Mode - Prefix LE, 6-5  
 lower-limit violation test command MMU W, 6-64  
 LSAD command, 3-53

## M

MA command, 3-54  
 Machine-Instruction Operation Codes, 4-1  
 MAE command, 3-57  
 Main Processor Registers, 4-5  
 MAL command, 3-59  
 Manual mode, A-4  
 manual modem connect mode, A-8  
 MAR command, 3-60  
 march address test command MT E, 6-31  
 MAW command, 3-60  
 MC68030 Assembly Language, 4-1  
 MC68030 Cache Diagnostic Tests, 6-13  
 MC68030 MPU Diagnostic Tests, 6-8  
 MC68030 ONCHIP CACHE TESTS - Command CA30, 6-13  
 MC68681 DUART (SIO) TESTS, 6-75  
 MD[S] command, 3-62  
 Memory Diagnostic Tests, 6-24  
 Memory Error Display Format, 6-43  
 memory management unit, 2-13 (see also MMU)  
 Memory Management Unit Diagnostic Tests, 6-44  
 Memory Management Unit Registers, 4-5  
 MEMORY MANAGEMENT UNIT SUPPORT, 2-13  
 MEMORY MANAGEMENT UNIT TESTS - Command MMU, 6-44

memory map, 1-12  
 MEMORY REQUIREMENTS, 1-12  
 MEMORY TESTS - Command MT, 6-24  
 menu, A-1  
 MENU command, 3-64  
 MENU DETAILS, A-3  
 Missed Page, 6-68  
 MM command, 3-65  
 MMU, 2-13 (see also memory management unit)  
 MMU failed test, 2-13  
 MMU passed test, 2-13  
 Mnemonics and Delimiters, 4-4  
 modify-bit and index test command MMU Y, 6-67  
 Monitor Start-Up, 6-3  
 MPAR, 1-17 (see also Multi-processor Address Register)  
 MPCR, 1-17 (see also Multi-processor Control Register)  
 MPU A register test, 6-9  
 MPU B instruction test, 6-10  
 MPU C address mode test, 6-11  
 MPU D exception processing test, 6-12  
 MPU TESTS FOR THE MC68030 - Command MPU, 6-8  
 MS command, 3-68  
 multi-level directory, 6-3  
 Multi-processor Address Register, 1-17 (see also MPAR)  
 Multi-processor Control Register, 1-17 (see also MPCR)  
 Multi-processor Control Register (MPCR) Method, 1-17  
 MULTIPROCESSOR SUPPORT, 1-16  
 MULU32, 5-43  
 MVME319 Controller-Dependent Status Codes, F-2  
 MVME320 Controller-Dependent Status Codes, F-2  
 MVME321 Controller-Dependent Status Codes, F-3  
 MVME323 Controller-Dependent Status Codes, F-4  
 MVME350, 1-7  
 MVME350 Controller-Dependent Status Codes, F-5  
 MVME360 Controller-Dependent Status Codes, F-6

## N

NOAB command, 3-3  
 NOMA command, 3-54  
 NOMAL command, 3-59  
 Non-Verbose Mode - Prefix NV, 6-6  
 Non-volatile RAM access error, 1-6, 6-3  
 NOPA command, 3-72  
 NOPF command, 3-76  
 NORB command, 3-78  
 Numeric values, 2-3

## 0

OBA command, 3-69  
 octal, 2-3  
 offset registers, 2-5  
 Operand Field, 4-4  
 operands types, 4-7  
 Operation Field, 4-3, 4-4  
 option field, 2-1  
 OUTCHR, 5-21  
 OUTLN, 5-22  
 Output character routine, 5-21  
 Output string along with (CR)(LF), 5-22  
 Output string to default output port, 5-22  
 Output string with CR and LF, 5-23  
 Output string with data, 5-26  
 Output string with data and (CR)(LF), 5-26  
 Output string with no CR or LF, 5-23  
 OUTSTR, 5-22  
 OVERVIEW OF DIAGNOSTIC FIRMWARE, 6-1

## P

PA command, 3-72  
 page-descriptor modify-bit test command MMU L, 6-57  
 page-descriptor used-bit test command MMU K, 6-56  
 PARAMETER FIELD DEFINITIONS, D-7  
 Parse value, assign to variable, 5-41  
 PCRLF, 5-24  
 PF command, 3-73  
 Pointer/Count Format, 5-2  
 Pointer/Pointer Format, 5-2  
 Port Numbers, 2-7  
 power-up confidence test, 1-6, 6-3  
 prefetch on invalid-page boundary test command MMU X, 6-65  
 PRESERVING THE DEBUGGER OPERATING ENVIRONMENT, 2-8  
 Print (CR)(LF) sequence, 5-24  
 program counter, 1-11  
 program test command MT I, 6-37  
 PS command, 3-77  
 Pseudo Registers, 4-5  
 pseudo-registers, 2-5

## Q

quote mark, 2-4

## R

RAM test failed, 1-6, 6-3  
random byte test command MT H, 6-35  
range, 2-3  
RB command, 3-78  
RD command, 3-79  
Read line to fixed-length buffer, 5-8  
Read Loop - Command RL.size, 6-7  
read mapped ROM test command MMU F, 6-50  
Read string into variable-length buffer, 5-7  
Read the RTC registers, 5-36  
read/modify/write cycle test command MMU 0, 6-68  
READLN, 5-8  
READSTR, 5-7  
real-time clock test command (RTC), 6-71  
REDIR, 5-37  
Redirect I O function, 5-37  
Redirect input, 5-38  
Redirect output, 5-38  
REDIR I, 5-38  
REDIR O, 5-38  
redisplay, 2-2  
REFERENCE MANUALS, 1-19  
refresh test command MT G, 6-33  
REMOTE command, 3-84  
RESET command, 3-85  
resident assembler, 4-2  
resident structured assembler, 4-2  
RESTARTING THE SYSTEM, 1-10  
resume, 2-2  
RETURN, 5-39  
Return pointer to board ID packet, 5-46  
Return to 147Bug, 5-39  
RM command, 3-86  
ROM test failed, 1-6, 6-3  
ROMBOOT, 1-7  
root pointer (RP) register command MMU A, 6-45  
RS command, 3-89  
RTC\_DSP, 5-35  
RTC\_DT, 5-34  
RTC\_RD, 5-36  
RTC\_TM, 5-33  
rubout, 2-2

## S

S-record format, C-1  
 Sample Table Walk Display, 6-70  
 SD command, 3-90  
 sector, 1-14  
 segment-descriptor used-bit test command MMU M, 6-58  
 Select Alternate Boot Device, A-3  
 Self Test Prefix/Command ST, 6-4  
 Send break, 5-28  
 set bus data width command MT D, 6-30  
 SET command, 3-91  
 set function code command MT A, 6-25  
 set start address command MT B, 6-26  
 set stop address command MT C, 6-28  
 SIO timer test command (SIO G), 6-83  
 SNDBRK, 5-28  
 Source Line Format, 4-2  
 SOURCE PROGRAM CODING, 4-2  
 Square brackets, 2-2  
 status register, 1-11  
 Stop-On-Error Mode - Prefix SE, 6-5  
 STRCMP, 5-42  
 String Formats for I O, 5-2  
 super\_data space test command MMU D, 6-48  
 super\_prog space test command MMU C, 6-47  
 Switch Directories - Command SD, 6-5  
 syntactic variables, 2-3  
 SYSCALL System Call Directive, 4-10  
 SYSTEM CALL ROUTINES, 5-3  
 system console, 1-5, 6-2  
 system mode, 6-3, 6-5  
 SYSTEM MODE OPERATION, A-1  
 SYSTEM START-UP, 6-1

## T

T command, 3-92  
 TA command, 3-94  
 Table Walk Display Format, 6-70  
 target code, 1-8  
 target vector table, 2-10  
 TAS test command MT J, 6-39  
 TC command, 3-95  
 Terminal mode, A-4, A-9  
 TIME command, 3-96  
 Time initialization for RTC, 5-33  
 TM command, 3-97  
 TM\_INI, 5-29  
 TM\_RD, 5-31

TM\_STRO, 5-30  
translation control (TC) register test command MMU B, 6-46  
Transparent Mode, A-8  
TRAP 15, 1-15, 2-1, 2-8, 4-10, 5-1  
TRAP 15 handler, 5-1  
TT command, 3-98  
TX/RX ready IRQ test command (SIO C), 6-78  
TX/RX ready test command (SIO D), 6-80

## U

Unmodified Page, 6-68  
Unsigned 32-bit x 32-bit divide, 5-44  
Unsigned 32-bit x 32-bit multiply, 5-43  
upper-limit violation test command MMU V, 6-63  
user\_data space test command MMU H, 6-53  
user\_program space test command MMU I, 6-54  
Using 147Bug Target Vector Table, 2-10  
UTILITIES, 6-6

## V

Valid expression examples, 2-4  
VE command, 3-100  
Vector Table and Wordspace, 2-8  
vector table area, 2-8  
vertical bar, 2-2  
VME gate array test command (VMEGA), 6-84  
VMEchip Method, 1-18  
VOLUME ID BLOCK 0 (VID), D-1

## W

wait, 2-2  
walk a bit test command MT F, 6-32  
WRITD, 5-26  
WRITDLN, 5-26  
WRITE, 5-23  
Write Loop - Command WL.size, 6-7  
write-protect page test command MMU R, 6-61  
write-protect segment test command MMU S, 6-62  
write/mapped-read pages test command MMU E, 6-49  
Write/Read Loop - Command WR.size, 6-7  
WRITELN, 5-23

X

XON XOFF, 1-5, 6-2  
XON/XOFF, 2-2

Z

Zero Pass Count - Command ZP, 6-6



