MVME133BUG/D1

MVME133BUG 133Bug Debugging Package User's Manual

MOTOROLA INC.

SYSTEMS

MOTOROLA

SAFETY SUMMARY
SAFETY DEPENDS ON YOU

The following general safety precautions must be observed during all phases of operation, service, and repair of this equipment. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the equipment. Motorola Inc. assumes no liability for the customer's failure to comply with these requirements. The safety precautions listed below represent warnings of certain dangers of which we are aware. You, as the user of the product, should follow these warnings and all other safety precautions necessary for the safe operation of the equipment in your operating environment.

GROUND THE INSTRUMENT.

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

DO NOT OPERATE IN AN EXPLOSIVE ATMOSPHERE.

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

KEEP AWAY FROM LIVE CIRCUITS.

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

DO NOT SERVICE OR ADJUST ALONE.

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

USE CAUTION WHEN EXPOSING OR HANDLING THE CRT.

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

DO NOT SUBSTITUTE PARTS OR MODIFY EQUIPMENT.

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

DANGEROUS PROCEDURE WARNINGS.

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

WARNING

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

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

CHAPTER 1

GENERAL INFORMATION

1.1 DESCRIPTION OF 133Bug

The 133Bug package, MVME133BUG, is a powerful evaluation and debugging tool for systems built around the MVME133 or MVME133-1 monoboard microcomputers. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 133Bug 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 133Bug routines that handle I/O, data conversion, and string functions are available to user programs through the TRAP #15 handler.

133Bug consists of three parts: (1) a command-driven user-interactive software debugger, described in Chapter 2 and hereafter referred to as the "debugger", (2) a command-driven diagnostic package for the MVME133 hardware, described in Appendix G and hereafter referred to as "the diagnostics", and (3) a user interface which accepts commands from the system console terminal.

When using 133Bug, 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 "133Bug>" is displayed and the user has all of the debugger commands at his disposal. If in the diagnostic directory, then the diagnostic prompt "133Diag>" 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 (refer to paragraph 3.32), or may examine the commands in the particular directory that the user is currently in by using the Help (HE) command (refer to paragraph 3.18).

Because 133Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. Figure 1-1 illustrates the flow of control in 133Bug. When a command is entered, 133Bug 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 133Bug, depending on the outcome of the user program.

Those users who have used one or more of Motorola's other debugging packages will find 133Bug very similar. There are two noticable differences. Many of the commands are more flexible and powerful. Also, the debugger in general is more "user-friendly", with more detailed error messages (refer to Appendix B) and an expanded online help facility.

MOTOROLA

TABLE OF CONTENTS (cont'd)

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 133Bug. This will give an idea of 133Bug structure and capabilities.

Paragraph 1.3, "Installation and Startup", describes a step-by-step procedure to power up the module and obtain the 133Bug prompt on the terminal screen.

For a question about syntax or operation of a particular 133Bug 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.

1.3 INSTALLATION AND STARTUP

For 133Bug to operate properly with the MVME133, 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 MVME133 User's Manual and configure the header jumpers on the module as required for the user's particular application. The only jumper configurations specifically dictated by 133Bug are those on J6. J6 must have jumpers between pins 1 and 3, and between pins 4 and 6. This sets EPROM sockets XU31 and XU46 for 64K x 8 chips in bank 1. (This may NOT be the factory configuration of the MVME133 as shipped.)

2. Refer to the MVME133 User's Manual and configure the Module Status Register (MSR) as required for the user's particular application. J15 sets or resets bits 0 through 4 of the MSR, and J1 enables or disables the system controller functions of the MVME133 thus allowing the SYSCON bit of the MSR to reflect system controller status (details in Appendix A).

3. Be sure that the two 64K x 8 133Bug EPROMs are installed in sockets XU31 (odd bytes, odd BXX label) and XU46 (even bytes, even BXX label) on the MVME133 module.

4. Refer to the set-up procedure for the user's particular chassis or system for details concerning the installation of the MVME133.

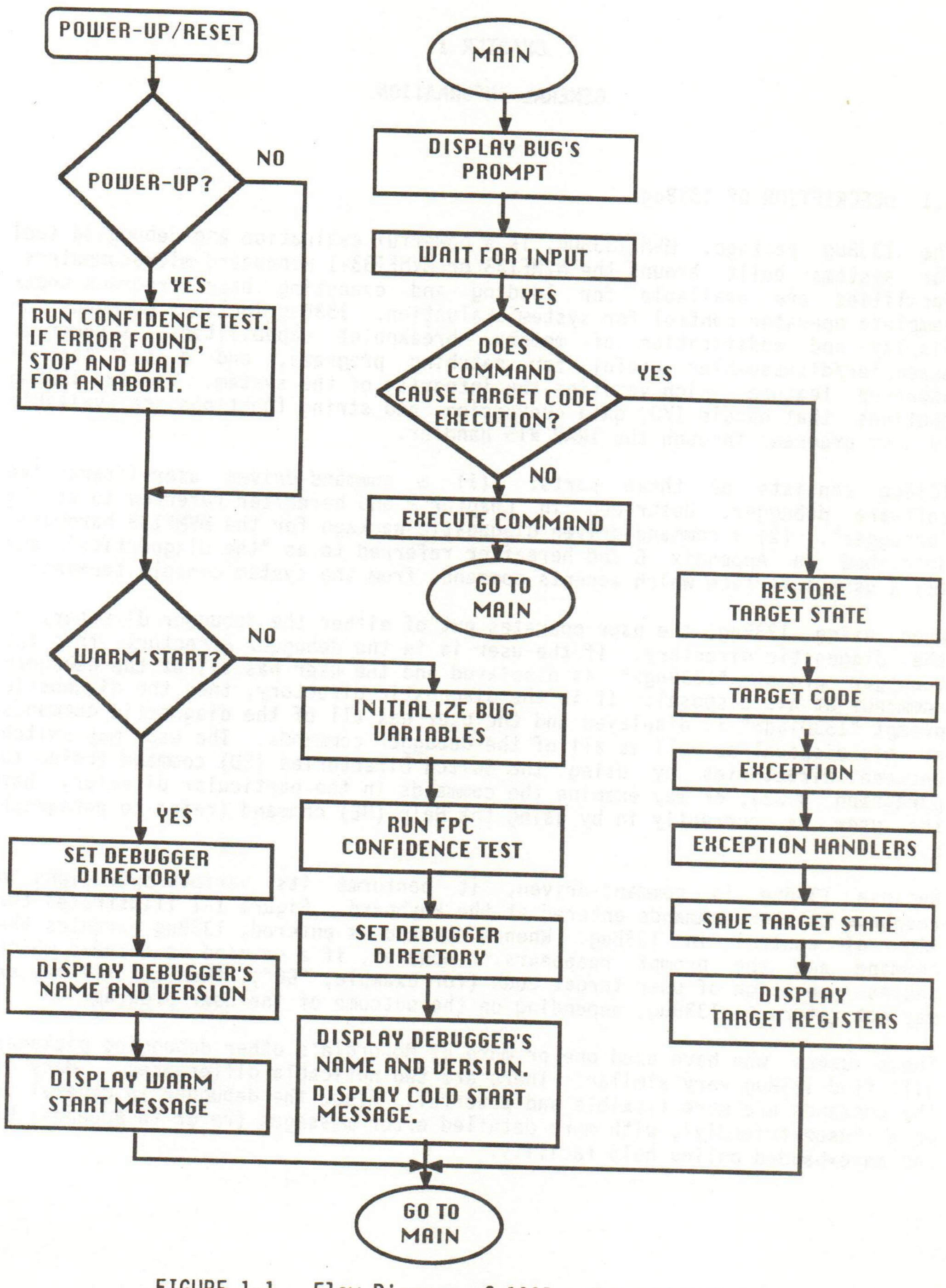

FIGURE 1-1. Flow Diagram of 133Bug Operational Mode

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'
... '68901 Register test failed'
```

Control remains with the confidence test and the monitor does not come up. The user may force the monitor to come up by pressing the ABORT switch on the MVME133 front panel.

1.4 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.

1.4.1 Reset

Pressing and releasing the MVME133 front panel RESET switch initiates a system reset. COLD and WARM reset modes are available. By default, 133Bug is in COLD mode (refer to the RESET command description). During COLD reset, a total system initialization takes place, as if the MVME133 had just been powered up. All static variables (including disk device and controller parameters) are restored to their default states. Ports 1 and 2 are reconfigured to their default state. 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.

During WARM reset, the 133Bug variables and tables are preserved, as well as the target state registers and breakpoints. If the particular MVME133 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 MVME133 illuminated HALT LED), for example after a double bus fault; or if the 133Bug environment is ever lost (vector table is destroyed, etc.).

5. Connect the terminal which is to be used as the 133Bug system console to the debug port connector J14 (DB25 connector) on the MVME133 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 MVME133 debug port at power-up.

   After power-up, the baud rate of the J14 debug port can be reconfigured by programming the MC68901 Multi-Function Peripheral (MFP) chip on the MVME133 module, or by using the Port Format (PF) command of the 133Bug debugger. Refer to the MVME133 User's Manual for details.

   NOTE

   In order for high-baud rate serial communication between 133Bug 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 (EXORterms do hardware handshaking), 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 up some device(s) (such as a host computer system or a serial printer) to port A (RS-485/RS-422) and/or port B (RS-232C) on the MVME133 rear connector J2, connect the appropriate cables and configure the port(s) as detailed in the MVME133 User's Manual. After power-up, these ports can be reconfigured by programming the Z8530 chip on the MVME133, or by using the PF command of the 133Bug debugger.

7. Power up the system. 133Bug executes some self-checks and displays the debugger prompt ("133Bug>").

   When power is applied to the MVME133, bit 13 at location $F80000 (Module Status Register = MSR) is set to zero indicating that power was just applied. (Refer to Appendix A for a complete description of the MSR.) This bit is tested within the 'Reset' logic path to see if the power-up confidence test needs to be executed. Location $FB0000 (Real-Time Clock = RTC) is read, thereby setting the power-up indicator to a one thus preventing any future power-up confidence test execution.

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

TABLE 1-1. Memory Map for Onboard RAM

J2 CONNECTIONS	ADDRESSES
1-2 and 3-4	\$000000 - \$0FFFFF
1-2	\$100000 - \$1FFFFF
3-4	\$200000 - \$2FFFFF
none	\$300000 - \$3FFFFF

NOTE

J2 controls the base address of the RAM, only as seen by the VMEbus. But, to the onboard logic (for example, a monitor), the RAM address is fixed at \$00000-\$FFFFF.

Regardless of where the onboard RAM is located, the first 16Kb is used for 133Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME133 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.

1.6 TERMINAL INPUT/OUTPUT CONTROL

When entering a command at the prompt, the following control codes may be entered for limited command line editing.

NOTE

The presence of the upward caret, "^", before a character indicates that he 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.

1.4.2 Abort

Abort is invoked by pressing and releasing the ABORT switch on the MVME133 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.

1.4.3 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 Port 1. 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.

1.5 MEMORY REQUIREMENTS

The program portion of 133Bug is approximately 128Kb of code. The EPROM sockets on board the MVME133 are mapped at locations $XXF00000 through $XXF1FFFF. However, the 133Bug code is position-independent and executes anywhere in memory.

133Bug requires a minimum of 16Kb of read/write memory to operate. This memory is usually the MVME133 onboard read/write memory, requiring stand-alone operation of the MVME133. The user selects the address at which onboard RAM appears from the VMEbus, by using jumpers on J2 and/or reprogramming U22, PALDP. When U22 contains the default factory program, J2 selects the base addresses as given in Table 1-1.

1.7.2 Disk I/O via 133Bug Commands

These following 133Bug 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 133Bug so that the next disk command defaults to use the same controller and device.

1.7.2.1 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.

1.7.2.2 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.

1.7.2.3 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.

1.7.2.4 BO (Bootstrap Operating System). BO reads an operating system or control program from the specified device into memory, and then transfers control to it.

1.7.2.5 BH (Bootstrap and Halt). BH reads an operating system or control program from a specified device into memory, and then returns control to 133Bug. It is used as a debugging tool.

1.7.3 Disk I/O via 133Bug System Calls

All operations that actually access the disk are done directly or indirectly by 133Bug TRAP #15 system calls. (The command-level disk operations provide a convenient way of using these system calls without writing and executing a program.) The following system calls are provided to allow user programs to do disk I/O:

.DSKRD - System call to read blocks from a disk into memory.
.DSKWR - System call to write blocks from memory onto a disk.

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

When observing output from any 133Bug 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 133Bug 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.

1.7 DISK I/O SUPPORT

133Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 133Bug 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 133Bug. Default values for these parameters are assigned at power-up and cold-start reset, but may be altered as described in paragraph 1.7.4.

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

1.7.1 Blocks Versus Sectors

The logical block defines the unit of information for disk devices. A disk is viewed by 133Bug 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. 133Bug 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 DIAGNOSTIC FACILITIES

Included in the 133Bug package is a complete set of hardware diagnostics intended for testing and troubleshooting of the MVME133 (refer to Appendix G). 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 ("133Diag>") should appear. Refer to Appendix G 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.

1.9 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.

DOCUMENT TITLE	MOTOROLA PUBLICATION NUMBER
MVME050 System Controller Module and MVME701/MVME701A I/O Transition Module User's Manual	MVME050
MVME133 VMEmodule 32-Bit Monoboard Microcomputer User's Manual	MVME133
Customer Letter, MVME133bug Source Code - Version 1.0	MVME133BSC/L1
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
MVME360 Storage Module Drive Disk Controller User's Manual	MVME360
VERSAdos to VME Hardware and Software Configuration User's Manual	MVMEVDOS
MC68020 32-Bit Microprocessor User's Manual	MC68020UM
MC68881 Floating-Point Coprocessor User's Manual	MC68881UM
MC68901 Multi-Function Peripheral Data Sheet	ADI-984
M68000 Family VERSAdos System Facilities Reference Manual	M68KVSF

To perform a disk operation, 133Bug 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 (refer to paragraph 1.9) on the particular controller module for the format of its packets, and for using the IOC command.

1.7.4 Default 133Bug Controller and Device Parameters

133Bug 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 three 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, IOT 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.

Finally, the user may change the configuration files and re-create 133Bug so that it has different defaults. This last option is described in detail in customer letter MVME133BSC/L1 (refer to paragraph 1.9). Refer to Appendix E for disk controller data.

1.7.5 Disk I/O Error Codes

133Bug returns an error code if an attempted disk operation is unsuccessful. Refer to Appendix F for an explanation of disk I/O error codes.

CHAPTER 2

USING THE 133Bug DEBUGGER

2.1 ENTERING DEBUGGER COMMAND LINES

133Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt ("133Bug>") 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, if necessary, using the control characters described in paragraph 1.6.

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 paragraph 5.2.19). For more about this, refer to the description in paragraphs 3.14 and 3.16 for the GD and GO commands.

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

a. The command identifier (i.e., "MD" or "md" for the Memory Display command). Note that either upper- or 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.

The commands are shown using a modified Backus-Naur form syntax. The metasymbols used are:

< >	The angular brackets enclose an item, known as a syntactic variable, that is replaced in a command line by one of a class of items it represents.
[ ]	Square brackets enclose an item that is optional.
|	This symbol indicates that a choice is to be made. One of several items, separated by this symbol, should be selected.
/	The slash indicates that one or more of the items separated by this symbol can be selected.
{ }	Braces enclose an optional item that may occur zero or more times.

THIS PAGE INTENTIONALLY LEFT BLANK.

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

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

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

Valid expression examples:

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

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

2.1.1.2 Address as a Parameter. Many commands use <ADDR> as a parameter. The syntax accepted by 133Bug is similar to the one accepted by the MC68020 one-line assembler. All control addressing modes are allowed. An address + offset register mode is also provided.

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

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.

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

2.1.1.1 <u>Expression as a Parameter</u>. 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	$	$FFFFFFFF
Decimal	&	&1974, &10-&4
Octal	@	@456
Binary	%	%1000110

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

MOTOROLA

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 MC68020 VERSAdos Resident Assembler is shown below:

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

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

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

```
133Bug>MD 1327C;DI <CR>
0001327C 48E78080                 MOVEM.L  D0/A0,-(A7)
00013280 4280                     CLR.L    D0
00013282 1018                     MOVE.B   (A0)+,D0
00013284 5340                     SUBQ.W   #1,D0
00013286 12D8                     MOVE.B   (A0)+,(A1)+
00013288 51C8FFFC                 DBF      D0,$13286
0001328C 4CDF0101                 MOVEM.L  (A7)+,D0/A0
00013290 4E75                     RTS
133Bug>
```

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

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

Another way to enter a program is to download an object file from a host system, for example, an EXORmacs. 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 133Bug 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 MVME133 port B. (Refer to hardware configuration details in paragraph 1.3.) The file is downloaded from the host into MVME133 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 133Bug routines via the MC68020 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. 133Bug uses certain of the MVME133 onboard resources and may also use offboard system memory to contain temporary variables, exception vectors, etc. If the user disturbs resources upon which 133Bug depends, then the debugger may function unreliably or not at all.

2.4.1 133Bug Vector Table and Wordspace

As described in paragraph 1.5, "Memory Requirements", 133Bug needs 12Kb of read/write memory to operate and also allocates another 4Kb as user space for a total of 16Kb allocated. 133Bug 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, 133Bug reserves space for static variables and initializes these static variables to predefined default values. After the static variables, 133Bug 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 1.5 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.

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

```
133Bug>OF R0 <CR>
R0 =00000000 00000000? 1327C:16. <CR>
133Bug>MD 0+R0;DI <CR>
00000+R0 48E78080                    MOVEM.L   D0/A0,-(A7)
00004+R0 4280                        CLR.L     D0
00006+R0 1018                        MOVE.B    (A0)+,D0
00008+R0 5340                        SUBQ.W    #1,D0
0000A+R0 12D8                        MOVE.B    (A0)+,(A1)+
0000C+R0 51C8FFFC                    DBF       D0,$A+R0
00010+R0 4CDF0101                    MOVEM.L   (A7)+,D0/A0
00014+R0 4E75                        RTS
133Bug>
```

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

2.1.2 Port Numbers

Some 133Bug commands give the user the option of choosing the port which will be used to input or output. The valid port numbers which may be used for these commands are:

0 - MVME133 RS-232C Debug (Terminal Port at J14)
1 - MVME133 RS-232C (at P2)
2 - MVME133 RS-485 (at P2)

NOTE

These logical port numbers (0, 1, and 2) are referred to as "Debug Port", "Serial Port B" or "RS-232C Port", and "Serial Port A" or "RS-485 Port", respectively, by the MVME133 hardware documentation. For example, the command DU0 (Dump S-records to Port 0) would actually output data to the device connected to the debug port.

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 133Bug Assembler/Disassembler.

2.4.2.2 Tick Timer Example. For interrupts every 10 milliseconds, set up timer A.

Clear bit #5 of MFP_IERA	($F80007)	Disable interrupts from timer A.
Move #$10 into MFP_TACR	($F80019)	Reset and stop timer A.
Move #$7B into MFP_TADR	($F8001F)	Load countdown value into timer A data register.
Move #$06 into MFP_TACR	($F80019)	Set timer A for delay mode with 100 as prescaler.
Move #$68 into MFP_VR	($F80017)	Set starting vector for block of 16 interrupt vectors at $60. Also set software interrupt mode bit.

NOTE

The vector passed to the MPU for the timer A interrupt will be $6D, therefore, vector address = 4 x $6D = $1B4. User must place the beginning address of the timer A interrupt handling routine at this $1B4 location.

Move #$DF into MFP_IPRA	($F8000B)	Clear timer A interrupt-pending bit (bit #5 of IPRA).
Move #$DF into MFP_ISRA	($F8000F)	Clear timer A interrupt-in-service bit (bit #5 of ISRA).
Set bit #5 of MFP_IMRA	($F80013)	Unmask interrupts from timer A.
Set bit #5 of MFP_IERA	($F80007)	Enable interrupts from timer A.

2.4.2.3 Timer A Interrupt Handler.

Read MFP_ISRA	($F8000F)	Read interrupt-in-service register A.
Investigate MFP_ISRA	($F8000F)	If bit #5 is set, then the interrupt was in fact from timer A.
Take necessary action.		
Move #$DF into MFP_ISRA	($F8000F)	Clear timer A interrupt-in-service bit.
Return		

2.4.2 MC68901 Timer Registers

The MVME133 uses the Multi-Function Peripheral (MFP) MC68901 for its front panel debug port, its tick timers, its watchdog timer, and the status and control information. There are four timers in the MC68901, assigned as follows:

C - baud rate generator for the serial port.

A - software tick timer, capable of generating a periodic interrupt.

B - tick timer overflow/watchdog time-out, capable of generating a local/ system reset.

D - delay mode only, unassigned by hardware.

2.4.2.1 Calculation of the Countdown Value for Timer Data Register. The countdown value may be calculated as follows:

$$CD = \frac{TI * TO}{PS}$$

where:

CD = countdown value to be loaded into timer data register.
TI = timer interrupt frequency in Hz = 1,230,000 Hz.
TO = tick timer interrupts interval in seconds.
PS = prescaler value (4, 10, 16, 50, 64, 100, or 200).

Table 2-2 contains the values for PS and CD for some selected TO interrupts intervals.

Table 2-2. MC68901 Timer Prescaler and Countdown Values

INTERRUPTS INTERVAL DESIRED TO	PRESCALER VALUE PS	COUNTDOWN VALUE CD
1 ms = 0.001 sec	10	$7B (&123)
5 ms = 0.005 sec	50	$7B (&123)
10 ms = 0.010 sec	100	$7B (&123)
20 ms = 0.020 sec	100	$F6 (&246)
40 ms = 0.040 sec	200	$F6 (&246)
41.6 ms = 0.0416 sec	200	$0 (&256)

table area is the base address of the MVME133, determined as described in paragraph 1.5. 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.

133Bug 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 will be lost.

133bug 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.

2.4.3.2 Creating a New Vector Table.

A user program may create a separate vector table in memory to contain its exception vectors. 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 133Bug vector table into the corresponding vector locations in the user vector table.

The vector for the 133Bug 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 will provide 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
```

2.4.3 Exception Vectors Used by 133Bug

The exception vectors used by the debugger are listed in Table 2-3. These vectors must reside at the specified offsets in the target program vector table for the associated debugger facilities (breakpoints, trace mode, etc.) to operate.

Table 2-3. Exception Vectors Used by 133Bug

VECTOR OFFSET	EXCEPTION	133Bug 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)
$7C	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.

```
133Bug>RD <CR>
PC   =00003000 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   =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   =00004000
00003000 203900100000          MOVE.L   ($100000).L,D0
133Bug>T <CR>
PC   =00003006 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   =12345678 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
00003006 D280                  ADD.L    D0,D1
133Bug>
```

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 133Bug or it may create a separate exception vector table of its own. The two following paragraphs detail these two methods.

2.4.3.1 Using 133Bug Target Vector Table. 133Bug initializes and maintains a vector table area for target programs. A target program is any user program started by the bug, either manually with GO or Trace type commands or automatically with the BOot command. The start address of this target vector

```
133Bug>RD <CR>
PC   =00003000 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   =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   =00004000
00003000 203900F00000       MOVE.L  ($F00000).L,D0
133Bug>T <CR>
Exception: Long Bus Error
Format/Vector=B008
SSW=0145 Fault Addr.=00F00000 Data In=00000000 Data Out=00002006
PC   =00003000 SR   =A700=TR:ALL_S._7_.....
USP  =00003830 MSP  =00003C18 ISP* =00003FA4 VBR  =00000000
SFC  =0=XX     DFC  =0=XX     CACR =0=..     CAAR =00000000
D0   =12345678 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00003FA4
00003000 203900F00000       MVE.L   ($F00000).L,D0
133Bug>
```

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:

```
133Bug>MD (A7):&44 <CR>
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
133Bug>
```

2.5 FLOATING POINT COPROCESSOR SUPPORT

The floating point coprocessor (MC68881) is supported in this version of 133Bug. The commands RD, RM, MD, and MM have been extended to allow display and modification of floating point data in registers and in memory. Floating point instructions can be assembled/disassembled with the DI option of the MD/MM commands. Finally, the MC68881 target state is saved and restored along with the processor state as required when switching between the target program and 133Bug.

At power-up/reset an FPC confidence check is executed. Initially, a read of one of the floating point registers is attempted. If a bus error time-out is received (no FPC detected) or if an error is detected, then the test is aborted and an internal flag is set accordingly. If the test runs without errors, then an internal flag is set accordingly. This flag is later checked

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 133Bug target program vector table (which the program saved), yielding the address of the 133Bug exception vector. The user program then jumps to the address stored at this vector location, which is the address of the 133Bug 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. Below 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.
```

2.4.3.3 <u>133Bug Generalized Exception Handler</u>. 133Bug has a generalized exception handler which it uses to handle all of the exceptions not listed in Table 2-3. For all these exceptions, the target stack pointer is left pointing to the top of the exception stack frame created. In this way, if an unexpected exception occurs during execution of 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.

Single Precision Real

This format would appear in memory as:

```
 1-bit sign field               (1 binary digit)
 8-bit biased exponent field    (2 hex digits.  Bias = $7F)
23-bit fraction field           (6 hex digits)
```

A single precision number takes 4 bytes in memory.

Double Precision Real

This format would appear in memory as:

```
 1-bit sign field               (1 binary digit)
11-bit biased exponent field    (3 hex digits.  Bias = $3FF)
52-bit fraction field           (13 hex digits)
```

A double precision number takes 8 bytes in memory.

Extended Precision Real

This format would appear in memory as:

```
 1-bit sign field               (1 binary digit)
15-bit biased exponent field    (4 hex digits.  Bias = $3FFF)
64-bit mantissa field           (16 hex digits)
```

An extended precision number takes 12 bytes in memory. This is because there is a 16-bit undefined field following the exponent field. This field is never displayed nor required to be entered when modifying extended precision data.

NOTE

Single and double precision formats have an implied integer bit (always 1).

Packed Decimal Real

This format would appear in memory as:

```
 4-bit sign field               (4 binary digits)
16-bit exponent field           (4 hex digits)
68-bit mantissa field           (17 hex digits)
```

A packed decimal number takes 12 bytes in memory.

by the bug when doing a task switch. The FPC state will be saved and restored only if this flag is set. This allows proper bug operation in systems that do not have an FPC.

Valid data types that can be used when modifying a floating point data register or a floating point memory location are:

INTEGER DATA TYPES	
12	Byte
1234	Word
12345678	Long

Floating Point Data Types	
1_FF_7FFFFF	Single Precision Real Format
1_7FF_FFFFFFFFFFFFF	Double Precision Real Format
1_7FFF_FFFFFFFFFFFFFFFF	Extended Precision Real Format
1111_2103_123456789ABCDEF01	Packed Decimal Real Format
-3.12345678901234501_E+123	Scientific Notation Format (Decimal)

When entering data in single, double, extended, or packed decimal, the following rules must be observed:

- The sign field is the first field and is a binary field.
- The exponent field is the second field and is a hexadecimal field.
- The mantissa field is the last field and is a hexadecimal field.
- The sign field, the exponent field, and at least the first digit of the mantissa field must be present (any unspecified digits in the mantissa field are set to zero).
- Each field must be separated from adjacent fields by an underscore.
- All the digit positions in the sign and exponent fields must be present.

CHAPTER 3

THE 133Bug DEBUGGER COMMAND SET

3.1 INTRODUCTION

This chapter contains descriptions of each of the debugger commands, with one or more examples of each. Table 3-1 summarizes the 133Bug debugger commands.

TABLE 3-1. Debugger Commands

COMMAND MNEMONIC	TITLE	PARAGRAPH
BF	Block of Memory Fill	3.2
BH	Bootstrap Operating System and Halt	3.3
BI	Block of Memory Initialize	3.4
BM	Block of Memory Move	3.5
BO	Bootstrap Operating System	3.6
BR/NOBR	Breakpoint Insert/Delete	3.7
BS	Block of Memory Search	3.8
BV	Block of Memory Verify	3.9
CS	Checksum	3.10
DC	Data Conversion	3.11
DU	Dump S-records	3.12
EEP	EEPROM Programming	3.13
GD	Go Direct (Ignore Breakpoints)	3.14
GN	Go to Next Instruction	3.15
GO	Go Execute User Program	3.16
GT	Go to Temporary Breakpoint	3.17
HE	Help	3.18
IOC	I/O Control for Disk	3.19
IOP	I/O Physical (Direct Disk Access)	3.20
IOT	I/O "TEACH" for Configuring Disk Controller	3.21
LO	Load S-records from Host	3.22
MD	Memory Display	3.23
MM	Memory Modify	3.24
MS	Memory Set	3.25
OF	Offset Registers Display/Modify	3.26
PA/NOPA	Printer Attach/Detach	3.27
PF/NOPF	Port Format/Detach	3.28
RD	Register Display	3.29
RESET	Cold/Warm Reset	3.30
RM	Register Modify	3.31
SD	Switch Directories	3.32
SET	Set Time and Date	3.33
T	Trace	3.34
TA	Terminal Attach	3.35
TC	Trace on Change of Control Flow	3.36
TIME	Display Time and Date	3.37
TM	Transparent Mode	3.38
TT	Trace to Temporary Breakpoint	3.39
VE	Verify S-records Against Memory	3.40

Scientific Notation

This format provides a convenient way to enter and display a floating point decimal number. Internally, the number is assembled into a packed decimal number and then converted into a number of the specified data type.

Entering data in this format requires the following fields:

An optional sign bit (+ or -).
One decimal digit followed by a decimal point.
Up to 17 decimal digits. At least one digit must be entered.
An optional Exponent field that consists of:

An optional underscore.
The Exponent field identifier, letter "E".
An optional Exponent sign (+, -).
From 1 to 3 decimal digits.

The MC68881 registers are:

3 system registers:

FPCR - Floating-point Control Register
FPSR - Floating-point Status Register
FPIAR - Floating-point Instruction Address Register

8 data registers:

FP0-FP7 - Floating-point Data Registers

For more information about the MC68881 coprocessor, refer to the MC68881 Floating Point Coprocessor User's Manual, as listed in Chapter 1.

3.2 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.)

```
133Bug>BF 20000,2001F 4E71 <CR>
Effective address: 00020000
Effective address: 0002001F
133Bug>MD 20000:30;B <CR>
00020000 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71   NqNqNqNqNqNqNqNq
00020010 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71   NqNqNqNqNqNqNqNq
00020020 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
```

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

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 underlined. This is done for clarity in understanding the examples (to distinguish between characters input by the user and characters output by 133Bug). No underline is typed in actual input. The symbol <CR> represents the carriage return key on the user's terminal keyboard. Whenever this symbol appears, it means that a carriage return was entered by the user.

3.3 BOOTSTRAP OPERATING SYSTEM AND HALT

```
BH [<Device LUN>][<DEL><Controller LUN>][<DEL><String>]
```

where:

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

BH is used to load an operating system or control program from disk into memory. This command works in exactly the same way as the BO command, except that control is not given to the loaded program. After the registers are initialized, control is returned to the 133Bug debugger and the prompt reappears on the terminal screen. Because control is retained by 133Bug, all the 133Bug facilities are available for debugging the loaded program if necessary.

Examples:

```
133Bug>bh 1,0 <CR>             Boot and halt from device LUN 1, controller 0.
133Bug>


133Bug>bh a,3,test2;d <CR>     Boot and halt from device LUN $A, controller 3,
133Bug>                        and pass the string "test2;d" to the loaded
                               program.
```

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

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

```
133Bug>BF 20000:10 4E71 ;B <CR>
Effective address: 00020000
Effective count  : &16
Data = $71
133Bug>MD 20000:30;B <CR>
00020000 71 71 71 71 71 71 71 71   71 71 71 71 71 71 71 71   qqqqqqqqqqqqqqqq
00020010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00020020 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
```

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

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

```
133Bug>BF 20000,20006 12345678 ; L <CR>
Effective address: 00020000
Effective address: 00020003
133Bug>MD 20000:30;B <CR>
00020000 12 34 56 78 00 00 00 00   00 00 00 00 00 00 00 00   .4Vx............
00020010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00020020 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
```

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

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

```
133Bug>BF 20000:18 0 1 <CR>             (default size is word)
Effective address: 00020000
Effective count  : &24
133Bug>MD 20000:18 <CR>
00020000 00 00 00 01 00 02 00 03   00 04 00 05 00 06 00 07   ................
00020010 00 08 00 09 00 0A 00 0B   00 0C 00 0D 00 0E 00 0F   ................
00020020 00 10 00 11 00 12 00 13   00 14 00 15 00 16 00 17   ................
```

3.5 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.)

```
133Bug>MD 21000:20;B <CR>
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    ................

133Bug>BM 21000 2100F 20000 <CR>
Effective address: 00021000
Effective address: 0002100F
Effective address: 00020000

133Bug>MD 20000:20;B <CR>
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    ................
133Bug>
```

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...

```
133Bug>MD 20000 2000A;DI <CR>
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.

3.4 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:

```
133Bug>BI 0 : 10000 ;B <CR>
Effective address: 00000000
Effective count  : &65536
133Bug>
```

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

```
133Bug>BI 0,1FFFFF <CR>
Effective address: 00000000
Effective address: 001FFFFF
RAM FAIL AT $00100000
133Bug>
```

3.6 BOOTSTRAP OPERATING SYSTEM BO

```
BO [<Device LUN>][<DEL><Controller LUN>][<DEL><String>]
```

where:

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

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

The following sequence of events occurs when BO is invoked:

1. Block 0 of the device LUN and controller 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.
   ```

```
133Bug>BM 20002 2000B 20004 <CR>
Effective address: 00020002
Effective address: 0002000B
Effective address: 00020004

133Bug>MD 20000 2000C;DI <CR>
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.

```
133Bug>MM 20002;DI <CR>
00020002 E2A2                      ASR.L     D1,D2 ? NOP <CR>
00020002 4E71                      NOP
00020004 E2A2                      ASR.L     D1,D2 ? . <CR>
133Bug>

133Bug>MD 20000 2000C;DI <CR>
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
133Bug>
```

3.7 BREAKPOINT INSERT/DELETE **BR NOBR**

```
BR   [<ADDR>[:<COUNT>]]
NOBR [<ADDR>]
```

The BR command allows the user to set a target code instruction address as a "breakpoint address" for debugging purposes. If, during target code execution, a breakpoint with 0 count is found, the target code state is saved in the target registers and control is returned back to 133Bug. 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 133Bug.

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:

```
133Bug>BR 14000,14200  14700:&12 <CR>     Set some breakpoints.
BREAKPOINTS
00014000          00014200
00014700:C
133Bug>NOBR 14200 <CR>                     Delete one breakpoint.
BREAKPOINTS
00014000          00014700:C
133Bug>NOBR <CR>                           Delete all breakpoints.
BREAKPOINTS
133Bug>
```

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 = 'IPLx', with x = $0C ($49504C0C)
   The ASCII string 'IPL' indicates that this is the Initial Program Load sequence; the code $0C indicates TRAP #15 support with stack parameter passing and TRAP #15 disk support.
   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:

133Bug><u>BO <CR></u>	Boot from device LUN 0, controller LUN 0.
133Bug><u>BO 3 <CR></u>	Boot from device LUN 3, controller LUN 0.
133Bug><u>bo , 3 <CR></u>	Boot from device LUN 0, controller LUN 3.
133Bug><u>bo 8 0,test <CR></u>	Boot from device LUN 8, controller LUN 0, and pass the string "test" to the booted program.

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.

NOTE

Due to a minor bug in the firmware, the over range message may not appear on all releases of 133Bug.

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

```
00030000 00 00 00 45 72 72 6F 72   20 53 74 61 74 75 73 3D   ...Error Status=
00030010 34 46 2F 2F 43 6F 6E 66   69 67 54 61 62 6C 65 53   4F//ConfigTableS
00030020 74 61 72 74 3A 00 00 00   00 00 00 00 00 00 00 00   tart:...........
```

```
133Bug>BS 30000 3002F 'Task Status' <CR>
Effective address: 00030000
Effective address: 0003002F
-not found-
```

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

```
133Bug>BS 30000 3002F 'Error Status' <CR>
Effective address: 00030000
Effective address: 0003002F
00030003
```

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

```
133Bug>BS 30000 3001F 'ConfigTableStart' <CR>
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.

```
133Bug>BS 30000:30 't' ; B <CR>
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.

```
133Bug>BS 30000:18,2F2F <CR>
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.

3.8 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 block search 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:

1. 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.

2. A compare is made with successive bytes, words, or longwords (depending on the size in effect) within the range for a match with the 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.

3. 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.

4. 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 <u>NOT</u> match the user-specified pattern. Otherwise this mode is identical to Mode 2.

3.9 BLOCK OF MEMORY VERIFY

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.)

```
133Bug>MD 20000:30;B <CR>
00020000 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71    NqNqNqNqNqNqNqNq
00020010 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71    NqNqNqNqNqNqNqNq
00020020 4E 71 4E 71 4E 71 4E 71   4E 71 4E 71 4E 71 4E 71    NqNqNqNqNqNqNqNq
133Bug>BV 20000 2001F 4E71 <CR>          (default size is word)
Effective address: 00020000
Effective address: 0002001F
133Bug>                                  (verify successful, nothing printed)
```

```
133Bug>bs 30000,3002F 3d34 <CR>                 Mode 2:  the default size is
Effective address: 00030000                     word and the data pattern is
Effective address: 0003002F                     not found, so a message is
-not found-                                     output.

133Bug>bs 30000,3002F 3d34 ;n <CR>              Mode 2: the default size is
Effective address: 00030000                     word and non-aligned option
Effective address: 0003002F                     is used, so the data pattern
0003000F:3D34                                   is found and displayed.

133Bug>BS 30000:30 60,F0 ;B <CR>                     Mode 2, using <RANGE>
Effective address: 00030000                          with count, mask option,
Effective count  : &48                               and size option:  count
00030006|6F   0003000B|61   00030015|6F   00030016|6E is displayed in decimal,
                                                     and the actual unmasked
00030017|66   00030018|69   00030019|67   0003001B|61 data patterns found are
                                                     displayed.
0003001C|62   0003001D|6C   0003001E|65   00030021|61

133Bug>BS 3000 1FFFF 0000 000F;V <CR>           Mode 3, on a different block
Effective address: 00003000                     of memory, mask option, scan
Effective address: 0001FFFE                     for words with low nybble
0000C000|E501    0001E224|A30E                  nonzero:  two locations
                                                failed to verify.
133Bug>
```

3.10 CHECKSUM

CS <address1> <address2>

The Checksum command provides access to the same checksum routine used by the power-up self-test firmware. This routine is used as follows within the firmware monitor.

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.

The <address> parameters can be provided in two forms:

a. An absolute address (24-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 the first address, 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 the ending address minus two is reached. Note that the last word addressed is NOT included in the checksum. This technique allows use of even ending addresses ($D40000 as opposed to $D3FFFE).

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

```
133Bug>MD 20000:30;B <CR>
00020000 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00     ................
00020010 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00     ................
00020020 00 00 00 00 00 00 00 00   00 00 4A FB 4A FB 4A FB     ..........J{J{J{
133Bug>BV 20000:30 0;B <CR>
Effective address: 00020000
Effective count  : &48
0002002A|4A   0002002B|FB   0002002C|4A   0002002D|FB    (mismatches are
0002002E|4A   0002002F|FB                                 printed out)
133Bug>
```

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

```
133Bug>MD 20000:18 <CR>
00020000 00 00 00 01 00 02 00 03   00 04 00 05 00 06 00 07     ................
00020010 00 08 FF FF 00 0A 00 0B   00 0C 00 0D 00 0E 00 0F     ................
00020020 00 10 00 11 00 12 00 13   00 14 00 15 00 16 00 17     ................
133Bug>BV 20000:18 0 1 <CR>                     (default size is word)
Effective address: 00020000
Effective count  : &24
00020012|FFFF                                   (mismatches are printed out)
133Bug>
```

```
EXAMPLE                                      COMMENT

133Bug> CS 20000 2002A <CR>                  Request checksum of area using
                                             absolute addresses.

Physical Address=00020000 0002002A
      (Even Odd)=4B34                        Checksum of even bytes is $4B.
                                             Checksum of odd bytes is $34.

133Bug> M 20026;W <CR>                       Place these bytes in zeroed area
                                             used while calculating checksum.
020026 0000 ?4B34. <CR>

133Bug> CS 20000 2002A <CR>                  Verify checksum.

Physical Address=00020000 0002002A           Result is 0000, good checksum.
      (Even Odd)=0000

133Bug> OF R3 <CR>                           Define value of relative offset
R3  =00000000 00000000? 20000 . <CR>         register 3.

133Bug> CS 0+R3 2A+R3 <CR>                   Request checksum of area using
                                             relative offset.
Physical Address=00020000 0002002A
      (Even Odd)=4B34                        Checksum of even bytes is $4B.
                                             Checksum of odd bytes is $34.

133Bug> M 26+R3;W <CR>                       Place these bytes in zeroed area used
                                             while checksum was calculated.
000026+R3 0000 ?4B34. <CR>

133Bug> CS 0+R3 2A+R3 <CR>                   Verify checksum.

Physical Address=00020000 0002002A
      (Even Odd)=0000

133Bug>
```

```
EXAMPLE                                       COMMENT

MVME133> MD 20000:3F;B <CR>                   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.

020000  42 4F 4F 54 00 00 00 14  00 00 00 A6 54 65 73 74  BOOT.......&Test
020010  41 F9 00 01 F0 00 20 3C  00 00 EF FF 11 00 51 C8  Ay..p. <..o...QH
020020  FF FC 4E 75 01 01 00 00  FF FF FF FF FF FF FF FF  .|Nu............
020030  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................

133Bug> M 20010;DI                            Display executable code plus revision
                                              number, checksum, socket ID, and a few
                                              unused bytes following the routine.

020010   41F90001F000          LEA.L          $0001F000,A0 ?<CR>
020016   203C0000EFFF          MOVE.L         #61439,D0 ?<CR>
02001C   1100                  MOVE.B         D0,-(A0) ?<CR>
02001E   51C8FFFC              DBF.L          D0,$02001C ?<CR>
020022   4E75                  RTS            ?<CR>
020024   0101                  BTST           D0,D1 ?<CR>
                                              0101 is revision.

020026   0000                  DC.W           $0000 ?<CR>
                                              0000 is where checksum is to be placed.

020028   FFFF                  DC.W           $FFFF ?<CR>
                                              FFFF is unused memory.

02002A   FFFF                  DC.W           $FFFF ?<CR>
                                              FFFF is unused memory.

02002C   FFFF                  DC.W           $FFFF ?<CR>
                                              FFFF is unused memory.

02002E   FFFF                  DC.W           $FFFF ?<CR>
                                              FFFF is unused memory.

020030   FFFF                  DC.W           $FFFF ?.<CR>
                                              FFFF is unused memory.
```

3.12 DUMP S-RECORDS

```
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 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.

<u>CAUTION</u>

IF AN OFFSET IS TO BE SPECIFIED BUT NO ENTRY ADDRESS IS TO BE SPECIFIED, THEN TWO COMMAS (INCIDATING 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.

```
133Bug> DU 20000 2002F <CR>
Effective address: 00020000
Effective address: 0002002F
133Bug>
```

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

```
133Bug> DU0 30000:&10 <CR>
Effective address: 00030000
Effective count  : &10
S0030000FC
S20E03000026025445535466084E4F7B
S9030000FC
133Bug>
```

3.11 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 MC68020 addressing mode.

Examples:

```
133Bug>DC 10 <CR>
        00000010 = $10 = &16

133Bug>DC &10-&20 <CR>
SIGNED  : FFFFFFF6 = -$A = -&10
UNSIGNED: FFFFFFF6 = $FFFFFFF6 = &4294967286

133Bug>DC 123+&345+@67+%1100001 <CR>
        00000314 = $314 = &788

133Bug>DC (2*3*8) /4 <CR>
        0000000C = $C = &12

133Bug>DC 55&F <CR>
        00000005 = $5 = &5

133Bug>DC 55>>1 <CR>
        0000002A = $2A = &42
```

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

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

133Bug>DC (A0) <CR>
        00030000 = $30000 = &196608

133Bug>DC ([A0]) <CR>
        11111111 = $11111111 = &286331153

133Bug>DC (4,A0) <CR>
        00030004 = $30004 = &196612

133Bug>DC ([4,A0]) <CR>
        22222222 = $22222222 = &572662306
```

Now enter the command for 133Bug to dump the S-records to the port.

```
133Bug> DU 20000 2000F 'FILE1' <CR>
Effective address: 00020000
Effective address: 0002000F
133Bug>

133Bug>TM <CR>                      ( Go into transparent mode again.        )
Escape character: $01=^A

QUIT <CR>                           ( 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 <CR>                          ( The VERSAdos prompt should return.     )
                                    ( Log off of the EXORmacs.               )

<^A>                                ( Enter the escape character to return to )
133Bug>                             ( 133Bug.                                )
```

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.

```
133Bug> DU 20000 2002F 'TEST' 2000A <CR>
Effective address: 00020000
Effective address: 0002002F
133Bug>
```

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

```
133Bug>TM <CR>                    ( Go into transparent mode to establish   )
Escape character: $01=^A          ( communication with the EXORmacs.        )

<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 <CR>               ( 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 )
133Bug>                           ( the escape character to return to 133Bug.)
```

3.14 GO DIRECT (IGNORE BREAKPOINTS) GD

GD [<ADDR>]

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

Once execution of the target code has begun, control may be returned to 133Bug by various conditions:

a. The user pressed the ABORT or RESET switches on the MVME133 front panel.
b. An unexpected exception occurred.
c. By execution of the .RETURN TRAP #15 function.

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

```
133Bug>MD 10000;DI <CR>
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     D2
0001000E 60FE                          BRA.B   $1000E
133Bug>RM D0 <CR>
```

Initialize D0 and start target program:

```
D0  =00000000 ? 52A9C. <CR>
133Bug>GD 10000 <CR>
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*=0000FFF8 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  =0000FFF8
0001000E 60FE                         BRA.B   $1000E
133Bug>
```

Set PC to start of program and restart target code:

```
133Bug>RM PC <CR>
PC  =0001000E ? 10000. <CR>
133Bug>GD <CR>
Effective address: 00010000
```

3.13 EEPROM PROGRAMMING

```
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.

NOTE

This command makes use of the MC68901 'D' timer for the 15 millisecond delay.

Example 1: (Assumes EEPROMs installed in XU24 and XU36 (bank 2), and J7 configured for the right size EEPROMs. Refer to the MVME133 User's Manual for jumper details. XU24 and XU36 are at addresses starting at $XXF20000 and ending at or below $XXF3FFFF in the main memory map, with the odd-byte chip in XU24 and the even-byte chip in XU36. Note that 133Bug is in the EPROMs in XU31 and XU46 (bank 1), at $XXF00000 through $XXF1FFFF, with odd bytes in U31 and even bytes in U46.)

```
133Bug>MD 21000:20;B <CR>
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   ................
133Bug>EEP 21000 2101F F20000 <CR>
Effective address: 00021000
Effective address: 0002101F
Effective address: 00F20000

133Bug>MD F20000:10;W <CR>
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   ................
133Bug>
```

Example 2:

```
133Bug>EEP 21000:8 F20000;W <CR>
Effective address: 00021000
Effective count  : &8
Effective address: 00F20000

133Bug>MD F20000:10;W <CR>
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   ................
133Bug>
```

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

```
133Bug>GN <CR>
Effective address: 00006008
Effective address: 00006004
At Breakpoint
PC  =00006008  SR  =2700=TR:OFF_S._7_.....
USP =00003830  MSP =00003C18  ISP*=00004000 VBR =00000000
SFC =0=XX      DFC =0=XX      CACR=0=..     CAAR=00000000
D0  =00000004  D1  =00000001  D2  =00000000 D3  =00000000
D4  =00000000  D5  =00000000  D6  =00000000 D7  =00000000
A0  =00000000  A1  =00000000  A2  =00000000 A3  =00000000
A4  =00000000  A5  =00000000  A6  =00000000 A7  =00004000
00006008 2600              MOVE.L       D0,D3
133Bug>
```

3.15 GO TO NEXT INSTRUCTION

GN

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

GN is especially helpful when debugging modular code because it allows 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.

```
133Bug>MD 6000:4;DI <CR>
00006000 7003                          MOVEQ.L #$3,D0
00006002 7201                          MOVEQ.L #$1,D1
00006004 61000FFA                      BSR.W   $7000
00006008 2600                          MOVE.L  D0,D3
133Bug>
```

The following simple routine resides at address $7000.

```
133Bug>MD 7000:2;DI <CR>
00007000 D081                          ADD.L   D1,D0
00007002 4E75                          RTS
133Bug>
```

Execute up to the BSR instruction.

```
133Bug>RM PC <CR>
PC   =00000000 ? 6000. <CR>
```

```
133Bug>GT 6004 <CR>
Effective address: 00006004
Effective address: 00006000
At Breakpoint
PC   =00006004  SR   =2700=TR:OFF_S._7_.....
USP  =00003830  MSP  =00003C18  ISP*=00004000 VBR =00000000
SFC  =0=XX      DFC  =0=XX      CACR=0=..     CAAR=00000000
D0   =00000003  D1   =00000001  D2   =00000000 D3   =00000000
D4   =00000000  D5   =00000000  D6   =00000000 D7   =00000000
A0   =00000000  A1   =00000000  A2   =00000000 A3   =00000000
A4   =00000000  A5   =00000000  A6   =00000000 A7   =00004000
00006004 61000FFA        BSR.W     $7000
133Bug>
```

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

Continue target program execution.

```
133Bug>G <CR>
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.

```
133Bug>NOBR <CR>
BREAKPOINTS
133Bug>GO 10000 <CR>
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*=0000FFF8 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  =0000FFF8
0001000E 60FE                BRA.B      $1000E
```

3.16 GO EXECUTE USER PROGRAM

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.

The sequence of events is as follows:

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

At this point control may be returned to 133Bug by various conditions:

1. A breakpoint with 0 count was found.
2. The user pressed the ABORT or RESET switches on the MVME133 front panel.
3. An unexpected exception occurred.
4. By execution of the .RETURN TRAP #15 function.

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

```
133Bug>MD 10000;DI <CR>
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     D2
0001000E 60FE                     BRA.B   $1000E
133Bug>RM D0 <CR>
```

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

```
D0  =00000000 ? 52A9C. <CR>

133Bug>BR 10000,1000E <CR>
BREAKPOINTS
00010000          0001000E
133Bug>GO 10000 <CR>
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
```

```
133Bug>GT 10006 <CR>
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
133Bug>
```

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

```
133Bug>GT 10002 <CR>
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
133Bug>
```

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

3.17 GO TO TEMPORARY BREAKPOINT

GT <ADDR>

GT 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.

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 133Bug by various conditions:

1. A breakpoint with count 0 was found.
2. The user pressed the ABORT or RESET switches on the MVME133 front panel.
3. An unexpected exception occurred.
4. By execution of the .RETURN TRAP #15 function.

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

```
133Bug>MD 10000;DI <CR>
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     D2
0001000E 60FE                         BRA.B   $1000E
133Bug>RM D0 <CR>
```

Initialize D0 and set a breakpoint:

```
D0   =00000000 ? 52A9C. <CR>

133Bug>BR 1000E <CR>
BREAKPOINTS
0001000E
133BUG>
```

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

```
133Bug>RM PC <CR>
PC   =0001000E ? 10000. <CR>
133Bug>
```

```
Press "RETURN" to continue <CR>
VE      Verify S-Records
133Bug>HE TC <CR>
TC      Trace on Change of Flow
133Bug>
```

3.18 HELP

HE [<COMMAND>]

HE is the 133Bug 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. Examples:

```
133Bug>HE <CR>
BF      Block Fill
BI      Block Initialize
BM      Block Move
BS      Block Search
BO      Boot Operating System
BH      Boot Operating System and Halt
BR      Breakpoint Insert
NOBR    Breakpoint Delete
BV      Block Verify
CS      Checksum
DC      Data conversion and expression evaluation
DU      Dump S-Records
EEP     EEPROM Programming
GO      GO to target code
G       "Alias" for previous command
GD      Go Direct (no breakpoints)
GN      Go and stop after next instruction
GT      Go and insert temporary breakpoint
HE      Help facility
IOC     I/O Control
IOP     I/O to Disk
Press "RETURN" to continue <CR>
IOP     I/O to Disk
IOT     I/O "Teach"
LO      Load S-Records
MD      Memory Display
MM      Memory Modify
M       "Alias" for previous command
MS      Memory Set
OF      Offset Registers
PA      Printer Attach
NOPA    Printer Detach
PF      Port Format
NOPF    Port Detach
RESET   Warm/Cold Reset
RD      Register display
RM      Register Modify
SD      Switch directory
SET     Set Time and Date
TA      Terminal Attach
T       Trace instruction
TC      Trace on Change of Flow
TT      Trace to temporary breakpoint
TM      Transparent Mode
TIME    Display Time and Date
```

3.20 I/O PHYSICAL (DIRECT DISK ACCESS)

IOP

The IOP command allows the user to do a single physical read or write to disk. When invoked, this command goes into a subcommand mode, prompting the user as indicated in the following example. The user may change the displayed value by typing a new value and a carriage return. The user may choose to keep the displayed value and display the next line by typing a carriage return.

As the user enters values, a disk communication command packet is prepared by the command with the entered values. After IOP has prompted the user for the last parameter, a .DSKRD or .DSKWR trap routine is called to perform the actual data transfer (refer to paragraphs on .DSKRD and .DSKWR in Chapter 5).

After reset, all parameters are initialized to certain default values. However, any new values entered are remembered by the IOP command and are displayed the next time that the IOP command is invoked.

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

a. Read/Write/Format

NOTE

The IOP command does not support Format for the MVME320 Disk Controller.

Does the user wish to ...

1. Read from the disk into memory? (user enters "R")
2. Write data from memory to the disk? (user enters "W")
3. Format a track or the entire disk? (user enters "F")

b. Controller LUN =00?

What is the Logical Unit Number (LUN) of the Controller to be used?

c. Device LUN =00?

What is the LUN of the Device of the particular controller that the user wishes to access? If the device LUN specified does not appear for the previously-specified controller, then an error results.

d. Memory Address =00003000?

If either Read or Write was selected, what is the starting address of the block of memory to be used? This memory is either written with data from the disk or else the data in the memory is copied out to the disk. The default address is $3000 past the start of the 133Bug vector table. Refer to paragraph 1.5 for an explanation of how the start of the user program area is determined.

3.19 I/O CONTROL FOR DISK

IOC

The IOC command allows a user to send command packets directly to a disk controller. The packet to be sent must already reside in memory and must follow the packet protocol of the particular disk controller. This packet protocol is outlined in the user's manual for the disk controller module. (Refer to paragraph 1.9.)

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

When invoked, this command prompts for the controller and drive required. The default controller LUN (CLUN) and device LUN (DLUN) when IOC is invoked are those most recently specified for IOP, IOT, or a previous invocation of IOC. An address where the controller command is located is also prompted for. The 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 #1. Specify an operation to the hard disk which is at DLUN #0.

```
133Bug>IOC <CR>
Controller LUN   =00? 1 <CR>
Device LUN       =00? <CR>
Packet address   =000010F0? 10000 <CR>
00010000 02 19 15 00 10 01 00 02   01 00 3D 00 30 00 00 00     ..........=.0...
00010010 00 00 00 00 03 00 00 00   00 00 02 00 03              ................
Send Packet (Y/N)? Y <CR>
133Bug>
```

3.21 I/O "TEACH" FOR CONFIGURING DISK CONTROLLER IOT

IOT [;H]

The IOT command allows the user to "teach" a new disk configuration to 133Bug 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 133Bug commands that access the disk use the TRAP #15 disk functions, changes in the descriptor tables affect all those commands. These commands include IOP, BO, BH, and also any user program that uses the TRAP #15 disk functions.

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

Note that during a boot, the configuration sector is normally read from the disk and the device descriptor table for the LUN used 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:

```
133Bug> IOT;H <CR>
   Disk Controllers Available
Lun   Type    Address     # dev
 0   VME320  $FFFFB000     4
 1   VME319  $FFFF0000     8
133Bug>
```

When invoked without the "H" option, 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 user may return to a previous parameter by typing a "^" followed by carriage return. All numerical values are interpreted as hexadecimal numbers. Decimal values may be entered by preceding the number with an "&".

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

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

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

e. Starting Block =00000000?

What is the starting block number on the disk to use? The default is block zero.

f. Number of Blocks=0002?

If either Read or Write was selected, how many blocks are to be transferred? The default is two blocks.

After the carriage return is entered for the "number of blocks" field, then the data transfer or format is initiated. If the data transfer is unsuccessful, then an error status word is displayed. Refer to Appendix F for an explanation of returned error status codes.

Example 1: The user desires to read blocks zero and one of floppy drive 2 into memory beginning at address $50000.

```
133Bug>IOP <CR>
Read/Write/Format      =R? <CR>
Controller LUN         =00? <CR>
Device LUN             =00? 2 <CR>
Memory Address         =00003000? 50000 <CR>
Starting Block         =0000000? <CR>
Number of Blocks       =0002? <CR>
133Bug>
```

g. Precomp. Cylinder

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

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

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

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

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

l. Step 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

m. Single/Double DATA Density

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

The parameters and attributes that are associated with a particular device are determined by a parameter and 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

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

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

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

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

c. Sectors/Track

   The number of sectors per track may be specified to be any number in the range of 0 to $FFFF.

5 1/4-inch floppy	- 16 sectors per track
Winchester	- 32 sectors per track

d. Starting Head

   The starting head number may be specified as any number in the range 0 to $FF.

e. Number of Heads

   The physical number of heads on the drive is entered here. This number must be in the range of 0 to $FF.

f. Number of Cylinders

   The number of cylinders per disk may be specified to be any number in the range 0 to $FFFF. 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

Example 2: Changing from a 40 Mb Winchester to a 70 Mb Winchester. (Remember, 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.)

```
133Bug>IOT <CR>
Controller LUN        =00? <CR>
Device LUN            =00? <CR>
Sector Size:
0-128 1-256
2-512 3-1024          =01? <CR>
Block Size:
0-128 1-256
2-512 3-1024          =01? <CR>
Sectors/Track         =0020? <CR>
Starting Head         =00? <CR>
Number of Heads       =06? 8 <CR>
Number of Cylinders   =033E? 400 <CR>
Precomp. Cylinder     =0000? 401 <CR>
Reduced Write Current Cylinder=0000? <CR>
Interleave Factor     =01? 0B <CR>
Spiral Offset         =00? <CR>
ECC Data Burst Length=0000? 000B <CR>
133Bug>
```

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.

```
133Bug>IOT <CR>                                         (Fixed Disk)
Controller LUN        =00? 2 <CR>
Device LUN            =00? <CR>
Sector Size:
0-128 1-256
2-512 3-1024          =02? 1 <CR>
Block Size:
0-128 1-256
2-512 3-1024          =01? <CR>
Sectors/Track         =0040? <CR>
Starting Head         =00? 10 <CR>
Number of Heads       =0A? 5 <CR>
Number of Cylinders   =0337? <CR>
Interleave Factor     =01? <CR>
Spiral Offset         =00? <CR>
Gap 1                 =10? 7 <CR>
Gap 2                 =20? 8 <CR>
Spare Sectors Count   =00? <CR>
133Bug>
```

n. Single/Double TRACK Density

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. Gap 1

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

p. Gap 2

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

q. Gap 3

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

r. Gap 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.

s. 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.

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

```
133Bug>IOT <CR>
Controller LUN        =00? <CR>
Device LUN            =00? 2 <CR>
Sector Size:
0-128 1-256
2-512 3-1024          =01? <CR>
Block Size:
0-128 1-256
2-512 3-1024          =01? <CR>
Sectors/Track         =0010? <CR>
Number of Heads       =02? <CR>
Number of Cylinders   =0050? <CR>
Precomp. Cylinder     =0028? <CR>
Step Rate Code        =00? <CR>
Single/Double DATA density =D (S/D)? <CR>
Single/Double TRACK density=D (S/D)? <CR>
133Bug>
```

3.22 LOAD S-RECORDS FROM HOST

```
LO [n] [<ADDR>] [;<X/-C/T>] [=<text>]
```

This command is used when data in the form of a file of Motorola S-records is to be downloaded from a host system to the MVME133. 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 now 9600 baud, rather than 19200 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 133Bug 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.

(Removable Disk)

```
133Bug>IOT <CR>
Controller LUN          =02 <CR>
Device LUN              =00? 1 <CR>
Sector Size:
0-128 1-256
2-512 3-1024            =01? <CR>
Block Size:
0-128 1-256
2-512 3-1024            =01? <CR>
Sectors/Track           =0040? <CR>
Starting Head           =00? <CR>
Number of Heads         =00? 1 <CR>
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>
133Bug>
```

```
7      65040004 4A00                   TST.B    D0
8      65040006 4E75                   RTS
9                                      END

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

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

```
S00F00005445535453335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
```

Load this file into MVME133 memory for execution at address $40000 as follows:

```
133Bug>TM <CR>                    ( Go into transparent mode to establish   )
Escape character: $01= ^A         ( communication with the VME/10.          )

<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     )
                                  ( 133Bug prompt.                          )

133Bug>LO -65000000 ;X=COPY TEST.MX,# <CR>
COPY TEST.MX,#
S00F00005445535453335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
133Bug>
```

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 VME/10 port which is connected with the MVME133 host port.

```
133Bug>MD 40000:4;DI <CR>
00040000 7001                 MOVEQ.L #1,D0
00040002 D088                 ADD.L   A0,D0
00040004 4A00                 TST.B   D0
00040006 4E75                 RTS
133Bug>
```

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

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 133Bug, and they set a different code in D4.

The S-record format (refer to Appendix C) allows for an entry point to be specified in the address field of the termination record of an S-record block. The contents of the address field of the termination record (plus the offset address, if any) are put into the target PC. Thus, after a download, 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 133Bug 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 133Bug 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 (a VME/10 with VERSAdos in this case) was used to create a program that looks like this:

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

Example 3:

```
133Bug>md 50008;di <CR>
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
133Bug>
```

Example 4:

```
133Bug>md 5000;d <CR>
00005000 0_3F6_44C1D0F047FC2= 2.4777000000000002_E-0003
00005008 0_423_DAEFF04800000= 1.2749000000000000_E+0011
00005010 0_000_0000000000000= 0.0000000000000000_E+0000
00005018 0_403_0000000000000= 1.6000000000000000_E+0001
00005020 1_3FF_0000000000000=-1.0000000000000000_E+0000
00005028 0_000_00000FFFFFFFF= 2.1219957904712067_E+0314
00005030 0_44D_FDE9F10A8D361= 6.0200000000000000_E+0023
00005038 0_3C0_79CA10C924223= 1.5999999999999999_E+0019
133Bug>
```

3.23 MEMORY DISPLAY

MD[S] <ADDR>[:<COUNT> | <ADDR>][; [B|W|L|S|D|X|P|DI]]   MD

This 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 MC68020 disassembler. No other option is allowed if DI is selected.

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.

<u>CAUTION</u>

IF MD REFERENCES NON-EXISTING MEMORY, THE SYSTEM HANGS UP AND DISPLAYS THE MESSAGE "VMEbus Bus Time-out". PRESSING THE ABORT SWITCH DOES NOT RECOVER SYSTEM OPERATION, BUT PRESSING THE RESET SWITCH DOES.

Example 1:

```
133Bug>md 12000 <CR>
00012000 2800  1942  2900  1942  2800  1842  2900  2846      (..B)..B(..B).(F
133Bug> <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

```
133Bug>md (a2,d5):&19;b <CR>
00013627 4F 82 00 C5 9B 10 33 7A  DF 01 6C 3D 4B 50 0F 0F   O..E..3z_.l=KP..
00013637 31 AB 80                                            1+.
133Bug>
```

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:

1. Enter <CR>. This closes the present location and continues with disassembly of next instruction.

2. Enter a new source instruction followed by <CR>. This invokes the assembler, which assembles the instruction and generates a "listing file" of one instruction.

3. Enter .<CR>. This closes the present location and exits the MM command.

If a new source line is entered (choice 2 above), the present line is erased and replaced by the new source line entered. If a hardcopy terminal is being used, port 0 should be reconfigured for hardcopy operation with the Port Format (PF) command. 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.

```
133Bug>mm 10000;di <CR>
00010000 46FC2400                 MOVE.W   #9216,SR ? divs.w -(a2),d2 <CR>
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)) <CR>
00010008                          BCHG     #12,9(A5,D6))
-----------------------------------------------------^
*** Unknown Field ***
00010008 4E7AD801                 MOVEC.L  VBR,A5 ?
```

Example 5: Step to next location and exit MM.

```
133Bug>m 1000c;di <CR>
0001000C 000000FF                 OR.B     #255,D0 ? <CR>
00010010 20C9                     MOVE.L   A1,(A0)+ ? .<CR>
133Bug>
```

3.24 MEMORY MODIFY

```
MM <ADDR>        [;[ [B|W|L|S|D|X|P][A][N] ]|[DI] ]                  MM
```

This 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.

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 133Bug.

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

Example 1:

```
133Bug>mm 10000 <CR>                     Access location 10000.
00010000 1234? <CR>
00010002 5678? 4321 <CR>                 Modify memory.
00010004 9ABC? 8765^ <CR>                Modify memory and backup.
00010002 4321? <CR>
00010000 1234? abcd. <CR>                Modify memory and exit.
```

Example 2:

```
133Bug>mm 10001;la <CR>                  Longword access to location 10001
00010001 CD432187? <CR>                  (Alternate location accesses).
00010009 00068010? 68010+10= <CR>        Modify and reopen location.
00010009 00068020? <CR>
00010009 00068020? . <CR>                Exit MM.
```

3.25 MEMORY SET

MS <ADDR> {Hexadecimal number} / {'string'}

Memory Set 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 nybble of the last byte accessed will be unchanged.

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:

```
133Bug>ms 25000 0123456789abcDEF 'This is ''133Bug''' 23456 <CR>
133Bug>md 25000:20;b <CR>
00025000 01 23 45 67 89 AB CD EF  54 68 69 73 20 69 73 20    .#Eg.+MoThis is
00025010 27 31 33 33 42 75 67 27  23 45 60 00 00 00 00 00    '133Bug'#E`.....
133Bug>
```

Example 6:

```
133Bug>m 7000;x <CR>
00007000 0_0000_FFFFFFFF00000000? 1_3C10_84782 <CR>
0000700C 1_7FFF_00000000FFFFFFFF? 0_001A_F <CR>
00070018 0_0000_FFFFFFFF00000000? 6.02E23= <CR>
00070018 0_404D_FEF4F885469B0880? ^ <CR>
0007000C 0_001A_F000000000000000? <CR>
00070000 1_3C10_8478200000000000? . <CR>
133Bug>
```

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

5. The automatic register is always added to every absolute address argument of every 133Bug command where there is not an offset register explicitly called out.

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

Examples:

Display offset registers.

```
133Bug>OF <CR>
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.

```
133Bug>OF R0 <CR>
R0 =00000000 00000000? 20000 200FF <CR>
R1 =00000000 00000000? 25000:200^ <CR>
R0 =00020000 000200FF? . <CR>
```

Look at location $20000.

```
133Bug>M 20000;DI <CR>
00000+R0 41F95445 5354               LEA.L    ($54455354).L,A0 . <CR>
133Bug>M R0;DI <CR>
00000+R0 41F95445 5354               LEA.L    ($54455354).L,A0 . <CR>
133Bug>
```

Set R0 as the automatic register.

```
133Bug>OF R0;A <CR>
R0*=00020000 000200FF? . <CR>
```

To look at location $20000.

```
133Bug>M 0;DI <CR>
00000+R0 41F95445 5354               LEA.L    ($54455354).L,A0 . <CR>
133Bug>
```

To look at location 0, override the automatic offset.

```
133Bug>M 0+R7;DI <CR>
00000000 FFF8                        DC.W     $FFF8 . <CR>
133Bug>
```

3.26 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 paragraph 2.1.1.2.2).

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:

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

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

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

3.28 PORT FORMAT/PORT DETACH

PF
NOPF

```
PF[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.

3.28.1 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:

```
133Bug>PF <CR>
Current port assignments:  (Port #: Board name, Port name)
[00: VME133- "DEBUG"] [01: VME133- "RS232"] [02: VME133- "RS485"]
133Bug>
```

3.28.2 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 paragraph 3.28.4, Assigning a New Port.

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.

3.27 PRINTER ATTACH/DETACH

PA
NOPA

```
PA   [n]
NOPA [n]
```

These two commands "attach" or "detach" a printer to the user-specified serial 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 "PAn".

```
133Bug>PF3 <CR>
Logical unit $03 unassigned
Name of board? VME050 <CR>
Name of port? PTR <CR>
Port base address = $FFFF1080? <CR>
Auto Line Feed protocol [Y,N] = N? Y. <CR>
OK to proceed (y/n)? Y <CR>
133Bug>
```

For further details, refer to the PF command.

Examples:

```
CONSOLE DISPLAY:                   PRINTER OUTPUT:
133Bug>PA7 <CR>
(attaching port 7)                 (printer now attached)
133Bug>HE NOPA <CR>                130Bug>HE NOPA
NOPA      Printer detach           NOPA      Printer detach
    133Bug>NOPA <CR>                    133Bug>NOPA
(detach all attached printers)     (printer now detached)
133Bug>NOPA <CR>
No printer attached
133Bug>
```

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.

Master or Slave:

For RS-485 type ports only, the data direction may be selected as Master (M) or Slave (S). The default is S.

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. None of the current drivers utilize FIFO buffering, therefore, none initiate an XOFF condition.

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.

3.28.4 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 VME133, 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.

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.

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

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

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

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

Example:

```
133Bug>PF 1 <CR>
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 <CR>              (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 ^ <CR>
Stop Bits [1,2] = 2? . <CR>        (value acceptable, exit interactive mode)
OK to proceed (y/n)? Y             (carriage return not required)
133Bug>
```

3.28.3 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).

Character width:

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

3.29 REGISTER DISPLAY RD

RD [+|-|=][<REG1>[-<REG2>]{[/[+|-|=][<REG1>[-<REG2>]]]}]

The RD command is used to display the target state, that is, the processor state associated with the target program (refer to GO command). The instruction pointed to by the target PC is disassembled and displayed also. 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 register range is to be added.
- is a qualifier indicating that a register range is to be removed.
= is a qualifier indicating that a register range is to be set.

The qualifier is applied to the next register range only. If no qualifier is specified, a + qualifier is assumed.

/ is a required delimiter between register ranges.
<REG1> is the first register in a range of registers.
-<REG2> is the last register in a range of registers.

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.

The processor registers are:

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

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

Example 1:

```
133Bug>rd <CR>
PC  =00003000 SR  =2700=TR:OFF_S._7_.....
USP =0000F830 MSP =00003C18 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
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  =00004000
00003000 424F             DC.W        $424F
133Bug>
```

If the configuration of the new port is not fixed, then the interactive configuration mode is entered. Refer to paragraph 3.28.2 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 3 to the MVME050 printer port.

```
133Bug>PF 3 <CR>
Logical unit $03 unassigned
Name of board? <CR> (cause PF to list supported modules (boards), ports)
Boards and ports supported:
VME133:  DEBUG,RS232,RS485
VME050:  1,2,PTR
Name of board? VME050 <CR>   (uppercase or lowercase accepted)
Name of port? PTR <CR>
Port base address = $FFFF1080? <CR>
Auto Line Feed protocol [Y,N] = N? . <CR>
(interactive mode not entered because hardware has fixed configuration)
OK to proceed (y/n)? Y
133Bug>
```

3.28.5 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.

The RD command is also used to display the MC68881 floating point coprocessor (FPC) registers. An internal mask indicates which registers are displayed when RD <CR> is executed. At reset time, this mask is configured to display only the processor registers. To change the mask and enable the display of FPC registers, type: rd +fpc<CR>. This changes the mask and at the same time displays all the registers. Afterwards, every time that RD <CR> is typed, all the MPU and all the FPC registers are displayed. The floating point data registers are always displayed in extended precision and in scientific notation format. To change the mask and disable the display of FPC registers, type: RD -fpc<CR>. Note that this mask is also used by all the exception handler routines, including the trace and breakpoint exception handlers.

Example 2:

```
133Bug>rd +fpc <CR>
PC  =00003000 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  =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   =00004000
FPCR=00000000 FPSR=00000000-(CC=....     ) FPIAR=00000000
FP0 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP1 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP2 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP3 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP4 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP5 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP6 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
FP7 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
00003000 4AFC                ILLEGAL
133Bug>
```

The floating point status register display includes a mnemonic portion for the condition codes. The bit name appears (N, Z, I, NAN) if the respective bit is set, otherwise a "." indicates that it is cleared.

Example 3: To add FP0 and FPSR to the default display.

```
133Bug>RD FP0/FPSR <CR>
PC  =00003000 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  =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   =00004000
FPSR=00000000-(CC=....     )
FP0 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
00003000 4AFC                ILLEGAL
133Bug>
```

NOTES

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

TRACE BITS

T1	T0	MNEMONIC	DESCRIPTION
0	0	TR:OFF	Trace off
0	1	TR:CHG	Trace on change of flow
1	0	TR:ALL	Trace all states
1	1	TR:INV	Invalid mode

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

Interrupt Mask: A number from 0 to 7 indicates the current processor priority level.

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

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

FUNCTION CODE	MNEMONIC	DESCRIPTION
0	XX	Undefined
1	UD	User Data
2	UP	User Program
3	XX	Undefined
4	XX	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.

3.30 COLD/WARM RESET

RESET

The RESET command 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 MVME133 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 MVME133 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:

```
133Bug>RESET <CR>                           Arm to be set to warm start, the
Cold/Warm Start [C,W] = C? W <CR>           next time a reset is performed.
Execute [Y,N] ? Y <CR>                      Do a software reset now, actually
                                            forcing a warm start.

Copyright Motorola Inc. 1986, All Rights Reserved

VME133 Monitor/Debugger Version 1.0 - 5/1/86

WARM Start
133Bug>
```

Example 4: To remove D3-D5 and A2 from the display.

```
133Bug>RD -D3-D5/-A2 <CR>
PC  =00003000 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  =00000000 D1  =00000000 D2  =00000000 D6  =00000000
D7  =00000000 A0  =00000000 A1  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
FPSR=00000000-(CC=....   )
FP0 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-0FFF
00003000 4AFC              ILLEGAL
133Bug>
```

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

```
133Bug>RD =D6/A3 <CR>
D6   =00000000  A3   =00000000
00003000 4AFC              ILLEGAL
133Bug>
```

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.

```
133Bug>rd +mpu <CR>
PC  =00003000 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  =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  =00004000
00003000 4AFC              ILLEGAL
133Bug>
```

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

```
133Bug>rd +fpc <CR>
PC   =00002000 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   =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   =00004000
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
00002000 00000000          OR.B      #0,D0
133Bug>
```

3.31 REGISTER MODIFY

RM <REG>

RM 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:

```
133Bug>RM D5 <CR>
D5  =12345678? ABCDEF^ <CR>        Modify register and back up.
D4  =00000000? 3000. <CR>          Modify register and exit.
133Bug>
```

Example 2:

```
133Bug>rm sfc <CR>
SFC =7=CS    ? 1= <CR>             Modify register and reopen.
SFC =1=UD    ? . <CR>              Exit.
133Bug>
```

The RM command is also used to modify the MC68881 floating point coprocessor registers.

Example 3 (continues on next page):

```
133Bug>rm fpsr <CR>
FPSR =00000000-(CC=....     ) ? F000000 <CR>
FPIAR=00000000 ? <CR>
FP0 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFF ? 0_1234_5 <CR>
FP1 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFF ? 1.25E3 <CR>
FP2 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFF ? 1_7F_3FF<CR>
FP3 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFF ? 1100_9261_3 <CR>
FP4 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFF ? &564 <CR>
FP5 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFF ? 0_5FF_F0AB <CR>
FP6 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFF ? 3.1415 <CR>
FP7 =0_7FFF_FFFFFFFFFFFFFFFF= 0.FFFFFFFFFFFFFFFF_E-OFF ? -2.74638369E-36. <CR>
133Bug>
```

3.33 SET TIME AND DATE

SET

The SET command is interactive and begins with the user entering the ASCII string 'SET' followed by a carriage return. At this time, a prompt asking for HH:MM:SS is displayed. 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. When the correct time 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 MM/DD/YY. 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.

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

Example: To SET a date and time of May 11, 1985 2:05:32.7 PM the command is as follows:

```
133Bug>SET <CR>
 xx/xx/xx    xx:xx:xx.x
Enter time as HH:MM:SS (24 hour clock)
14:05:32 <CR>
Enter date as MM/DD/YY
05/11/85 <CR>
133Bug>
```

3.32 SWITCH DIRECTORIES SD

SD

This 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:

```
133Bug> SD <CR>
133Diag>                          ( The user has changed from the debugger   )
                                  ( directory to the diagnostic directory,   )
                                  ( as can be seen by the "133Diag>"         )
                                  ( prompt.                                  )
```

Example 2:

```
133Diag> SD <CR>
133Bug>                           ( The user is now back in the debugger     )
                                  ( directory.                               )
```

```
133Bug>T <CR>
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
133Bug>
```

Trace next instruction:

```
133Bug> <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
133Bug>
```

Trace the next two instructions:

```
133Bug>T 2 <CR>
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
133Bug>
```

3.34 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 133Bug.

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 133Bug.

The trace functions are implemented with the trace bits (T0, T1) in the MC68020 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.)

```
133Bug>MD 10000;DI <CR>
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     D2
0001000E 60FE                           BRA.B   $1000E
133Bug>
```

Initialize PC and D0:

```
133Bug>RM PC <CR>
PC  =00008000 ? 10000. <CR>
133Bug>RM D0 <CR>
D0  =00000000 ? 8F41C. <CR>
```

Display target registers and trace one instruction:

```
133Bug>RD <CR>
PC  =00010000 SR  =2700=TR:OFF_S._7_.....
USP =0000382C MSP =00003C14 ISP*=00004000 VBR =00000000
SFC =0=XX     DFC =0=XX     CACR=0=..     CAAR=00000000
D0  =0008F41C D1  =00000000 D2  =00000000 D3  =00000000
D4  =00000000 D5  =00000000 D6  =00000000 D7  =00000000
A0  =00000000 A1  =00000000 A2  =00000000 A3  =00000000
A4  =00000000 A5  =00000000 A6  =00000000 A7  =00004000
00010000 2200               MOVE.L  D0,D1
```

3.36 TRACE ON CHANGE OF CONTROL FLOW TC

TC [<COUNT>]

TC 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 133Bug.

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 133Bug.

The trace functions are implemented with the trace bits (T0, T1) in the MC68020 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.)

```
133Bug>MD 10000;DI <CR>
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     D2
0001000E 60FE                         BRA.B   $1000E
133Bug>
```

Initialize PC and D0:

```
133Bug>RM PC <CR>
PC  =00008000 ? 10000. <CR>
133Bug>RM D0 <CR>
D0  =00000000 ? 8F41C. <CR>
```

Trace on change of flow:

```
133Bug>TC <CR>
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
133Bug>
```

3.35 TERMINAL ATTACH TA

TA [<port>]

Terminal Attach 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.

```
133Bug> TA 2 <CR>            (No prompt appears unless port 2 was already the
                             console.)
```

Example 2: Restoring console to port selected at power-up.

```
133Bug> TA <CR>              (Prompt now appears at terminal connected to port
                             0.)
```

3.38 TRANSPARENT MODE TM

TM [n] [<ESCAPE>]

TM 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:

```
133Bug>TM <CR>                          Enter TM.
Escape character:  $01=^A               Exit code is always displayed.

<^A>                                    Exit transparent mode.
```

Example 2:

```
133Bug>TM  ^g <CR>                      Enter TM and set escape character
Escape character:  $07=^G               to ^G.

<^G>                                    Exit transparent mode.
133Bug>
```

3.37 DISPLAY TIME AND DATE TIME

TIME

This 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 data and time of May 11, 1985 2:05:32.7 would be displayed as:

```
133Bug>TIME <CR>
 05/11/85    14:05:32.7
133Bug>
```

```
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
133Bug>
```

3.39 TRACE TO TEMPORARY BREAKPOINT TT

TT <ADDR>

TT sets a temporary breakpoint at the specified address and traces until a breakpoint with 0 count is encountered. The temporary breakpoint is then removed (TT is analogous to the GT command) and control is returned to 133Bug. 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 133Bug.

The trace functions are implemented with the trace bits (T0, T1) in the MC68020 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.)

```
133Bug>MD 10000;DI <CR>
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     D2
0001000E 60FE                         BRA.B   $1000E
133Bug>
```

Initialize PC and D0:

```
133Bug>RM PC <CR>
PC   =00008000 ? 10000. <CR>
133Bug>RM D0 <CR>
D0   =00000000 ? 8F41C. <CR>
```

Trace to temporary breakpoint:

```
133Bug>TT 10006 <CR>
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
```

X option - 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 133Bug 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 133Bug AND if the checksum comparison has not been disabled via the "-C" option, then an error condition exists. A message is output stating the address of the record (as obtained from the address field of the record), the calculated checksum, and the checksum read with the record. A copy of the record is also output. This is a fatal error and causes the command to abort.

Examples:

This short program was developed on a host system.

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

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

Then the program was converted into an S-record file named TEST.MX that looks like this:

```
S00F000054455354533335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
```

3.40 VERIFY S-RECORDS AGAINST MEMORY VE

VE [n] [<ADDR>] [;<X/-C>] [=<text>]

This command is identical to the LO command with the exception that data is not stored to memory but merely compared to the contents of memory.

The VE command accepts serial data from a host system in the form of a file of Motorola S-records and compares it to data already in the MVME133 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 equals sign (=), is sent to the host before 133Bug 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 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.

CHAPTER 4

USING THE ONE-LINE ASSEMBLER/DISASSEMBLER

4.1 INTRODUCTION

Included as part of the 133Bug 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 MC68020/MC68881 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 MC68020 instructions are translated.

The 133Bug assembler is effectively a subset of the MC68020 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 MC68020 code.

4.1.1 M68020 Assembly Language

The symbolic language used to code source programs for processing by the assembler is MC68020 assembly language. This language is a collection of mnemonics representing:

. Operations

- MC68020 machine-instruction operation codes
- Directives (pseudo-ops)

. Operators

. Special symbols

4.1.1.1 <u>Machine-Instruction Operation Codes</u>.

That part of the assembly language that provides the mnemonic machine-instruction operation codes for the MC68020/MC68881 machine instructions is described in the MC68020 and MC68881 User's Manuals, MC68020UM and MC68881UM. Refer to these manuals for any question concerning operation codes.

4.1.1.2 <u>Directives</u>.

Normally, assembly language can contain mnemonic directives which specify auxiliary actions to be performed by the assembler.

The 133Bug 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 133Bug utilities. Refer to paragraphs 4.2.3 and 4.2.4, respectively.

This file was downloaded into memory at address $40000. The program may be examined in memory using the Memory Display (MD) command.

```
133Bug>MD 40000:4;DI <CR>
00040000 7001         MOVEQ.L #1,D0
00040002 D088         ADD.L   A0,D0
00040004 4A00         TST.B   D0
00040006 4E75         RTS
133Bug>
```

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.

```
133Bug>VE -65000000;X=COPY TEST.MX,# <CR>
S00F000054455354533335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091
Verify passes.
133Bug>
```

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.

```
133Bug>M 40002 <CR>
00040002 D088? D089. <CR>

133Bug>VE -65000000;X=COPY TEST.MX,# <CR>
S00F000054455354533335337202001015E
S30D650400007001D0884A004E75B3
S7056504000091

The following record(s) did not verify .....
S30D65040000------88--------B3

133Bug>
```

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

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 MC68020/MC68881 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 133Bug 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	2(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.

4.1.2 Comparison with MC68020 Resident Structured Assembler

There are several major differences between the 133Bug assembler and the MC68020 Resident Structured Assembler. The resident assembler is a two-pass assembler that processes an entire program as a unit, while the 133Bug assembler processes each line of a program as an individual unit. Due mainly to this basic functional difference, the capabilities of the 133Bug 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 133Bug 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 <u>or</u> current PC.
Slash (/)	-- Divide <u>or</u> delimiter in a register list.
Ampersand (&)	-- AND <u>or</u> 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 133Bug 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 <u>not</u> used.

b. Hexadecimal - is a string of hexadecimal digits (0-9, A-F) preceded by an optional dollar sign ($). An example is:

```
$AFE5
```

One or more ASCII characters enclosed by apostrophes (') constitute an ASCII string. ASCII strings are right-justified and zero-filled (if necessary), whether stored or used as immediate operands.

```
005000    0053                DC.W      'S'
005002    223C41424344        MOVE.L    #'ABCD',D1
005008    3536                DC.W      '56'
```

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

Pseudo Registers

R0-R7	User Offset Registers

Main Processor Registers

PC	Program Counter. Used only in forcing program counter-relative addressing.
SR	Status Register.
CCR	Condition Codes Register (Lower eight bits of SR).
USP	User Stack Pointer.
MSP	Master Stack Pointer.
ISP	Interrupt Stack Pointer.
VBR	Vector Base Register.
SFC	Source Function Code Register.
DFC	Destination Function Code Register.
CACR	Cache Control Register.
CAAR	Cache Address Register.
D0-D7	Data registers.
A0-A7	Address Registers. Address register A7 represents the active system stack pointer, that is, one of USP, MSP, or ISP, as specified by the M and S bits of the status register (SR).

Floating Point Coprocessor Registers

FPCR	Control Register
FPSR	Status Register
FPIAR	Instruction Address Register
FP0-7	Floating Point Data Registers

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 <EA> 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 MC68020 instruction mnemonics. Numbers are recognized as binary, octal, decimal, and hexadecimal, with hexadecimal the default case.

a. Decimal - is a string of decimal digits (0-9) preceded by an ampersand (&). Examples are:

```
&12334
-&987654321
```

TABLE 4-1. 133Bug Assembler Addressing Modes (cont'd)

FORMAT	DESCRIPTION
(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.

Whenever a number goes in the above addressing modes, the user may specify an expression and let the assembler figure out what the number is. At present the allowed operands are:

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	(*)

and the 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.

4.2.1.5 __Character Set__. The character set recognized by the 133Bug 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 address modes, combined with operation codes, define the particular function to be performed by a given instruction. Effective addressing and data organization are described in detail in Section 2, "Data Organization and Addressing Capabilities", of the MC68020 User's Manual.

Table 4-1 summarizes the addressing modes of the MC68020 which are accepted by the 133Bug one-line assembler.

TABLE 4-1. 133Bug 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.

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
```

Possible points of confusion:

a. The user should keep in mind that where a number is intended and it could be confused with a register, it must be differentiated in some way. For example:

```
CLR       D0        means CLR.W register D0.  On the other hand,

CLR       $D0
CLR       0D0
CLR       +D0
CLR       D0+0      all mean CLR.W memory location $D0.
```

b. With the use of '*' to represent both multiply and program counter, how does the assembler know when to use which definition?

For parsing algebraic expressions, the order of parsing is

```
<OPERAND><OPERATOR><OPERAND><OPERATOR>...
```

with a possible left or right parenthesis.

Given the above order, the assembler can distinguish by placement which definition to use. For example:

```
1.  ***        means PC    x    PC
2.  *+*        means PC    +    PC
3.  2**        means  2    *    PC
4.  *&&16      means PC    AND &16
```

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

a. Address register indirect with index, base displacement.
b. Address register memory indirect postindexed.
c. Address register memory indirect pre-indexed.
d. Program counter indirect with index, base displacement.
e. Program counter memory indirect postindexed.
f. Program counter memory indirect pre-indexed.

For modes Address register/Program counter indirect with index, base displacement, the rules for omission/skipping are as follows:

a. The user may terminate the operand at any time by specifying ')'.Example:

```
CLR       ()        or
CLR       (,,)      is equivalent to

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

The disassembled line can be an MC68020 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:

```
133Bug>MM 10000;DI <CR>
00010000  2600                    MOVE.L  D0,D3 ?  ADDQ.L #1,A3 <CR>
```

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:

```
133Bug>MM 10000;DI <CR>
00010000  528B                    ADDQ.L  #1,A3
00010002  4282                    CLR.L   D2  ?
```

If a hardcopy terminal is being used, port 0 should be reconfigured for hardcopy mode for proper operation (refer to Port Format (PF) command). In this case, the above example will look as follows:

```
133Bug>MM 10000;DI <CR>
00010000  2600                    MOVE.L  D0,D3 ? ADDQ.L #1,A3 <CR>
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 will display the line unassembled with an "^" under the field suspected of causing the error and an error message will be displayed. The location being accessed is redisplayed:

```
133Bug>m 10000;di <CR>
00010000 528B                     ADDQ.L    #1,A3 ? lea.l 5(a0,d8),a4 <CR>
00010000                          LEA.L     5(A0,D8),A4
-----------------------------------------------------^
*** Unknown Field ***
00010000 528B                     ADDQ.L    #1,A3 ?
```

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 #$F
        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.

CHAPTER 5

SYSTEM CALLS

5.1 INTRODUCTION

This chapter describes the 133Bug TRAP #15 handler, which allows system calls from user programs. The system calls can be used to access selected functional routines contained within 133Bug, including input and output routines. TRAP #15 may also be used to transfer control to 133Bug at the end of a user program (refer to the .RETURN function, paragraph 5.2.19).

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 MVME133 debug port). The defaults may be changed, however, using the .REDIR_I and .REDIR_O functions, as described in paragraph 5.2.18.

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 133Bug
        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 simply does 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 133Bug 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.

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,D0)
```

In the case of forward branches or jumps, the absolute address of the destination may not be known as the program is being entered. The user may temporarily enter an "*" for branch to self in order to reserve space. After the actual address is discovered, the line containing the branch instruction can be re-entered using the correct value.

4.3.4 Assembler Output/Program Listings

A listing of the program is obtained using the 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 PA command to activate the Port 1 printer. 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.

MOTOROLA

TABLE 5-1. 133Bug System Call Routines (cont'd)

CODE	FUNCTION	DESCRIPTION
$0040	.TM_INI	Time initialization
$0041	.DT_INI	Date initialization
$0042	.TM_DISP	Display time from RTC
$0043	.TM_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 133Bug
$0064	.BINDEC	Convert binary to Binary Coded Decimal (BCD)
$006B	.CHK_SUM	Generate checksum

Example:

```
133Bug>M 3000;DI <CR>
0000 3000 00000000        ORI.B #$0,D0? SYSCALL .OUTLN <CR>
0000 3000 4E4F0022        SYSCALL .OUTLN
0000 3004 00000000        ORI.B #$0,D0? . <CR>
133Bug>
```

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 <CR><LF>.

5.2 SYSTEM CALL ROUTINES

Table 5-1 summarizes the TRAP #15 functions. Refer to the writeups on the utilities for specific use information.

TABLE 5-1. 133Bug 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)
$0010	.DSKRD	Disk read	
$0011	.DSKWR	Disk write	
$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)

5.2.2 .INSTAT FUNCTION .INSTAT

TRAP FUNCTION: .INSTAT - Input serial port status

CODE: $0001

DESCRIPTION: INSTAT is 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
```

5.2.1 .INCHR FUNCTION

TRAP FUNCTION: .INCHR - Input character routine

CODE: $0000

DESCRIPTION: Reads a character from the default input port. The character is returned in the stack.

ENTRY CONDITIONS:

```
        SP ==> Space for character <byte>
               Word fill           <byte>
```

EXIT CONDITIONS DIFFERENT FROM ENTRY:

```
        SP ==> Character <byte>
               Word fill <byte>
```

EXAMPLE:

```
        SUBQ.L    #2,SP          Allocate space for result.
        SYSCALL   .INCHR         Call INCHR.
        MOVE.B    (SP)+,D0       Load character in D0.
```

5.2.4 .READSTR FUNCTION

TRAP FUNCTION: .READSTR - Read string into variable-length buffer

CODE: $0003

DESCRIPTION: READSTR is 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 <long>

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      (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 1.6, Terminal Input/Output Control, is in effect.

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 ==> Address of string buffer <long>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Address of last character in the string+1 <long>

EXAMPLE: If A0 contains the address where the string is to go;

```
SUBQ.L   #4,A7      Allocate space for result.
PEA      (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, returned pointer points to it. Control character processing as described in paragraph 1.6, Terminal Input/Output Control, is in effect.

5.2.6 .DSKRD, .DSKWR FUNCTIONS

TRAP FUNCTIONS: .DSKRD - Disk read function
.DSKWR - Disk write function

CODES: $0010
$0011

DESCRIPTION: These functions are used to read and write one or more sectors from/to a disk drive. 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, arranged as follows:

```
  F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
+===============================================================+
|        Controller LUN         |          Device LUN           |
|===============================================================|
|                          Status Word                          |
|===============================================================|
|            Memory Address Most-Significant Word               |
|====                                                       ====|
|            Memory Address Least-Significant Word              |
|===============================================================|
|             Block Number Most-Significant Word                |
|====                                                       ====|
|             Block Number Least-Significant Word               |
|===============================================================|
|                 Number of Blocks to Transfer                  |
|===============================================================|
| 0   0   0   0   0   0   0   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	- 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	- Starting block number to transfer. On a disk read, data is read starting at this block. On a disk write, data is written starting at this block.

5.2.5 .READLN FUNCTION

TRAP FUNCTION: .READLN - Read line to fixed-length buffer

CODE: $0004

DESCRIPTION: READLN is used to read a string of characters from the default input port. Characters are echoed to the default output port. A string consists of a count byte followed by the characters read from the input. The count byte indicates the number of characters in the input string, excluding <CR><LF>. A string may be up to 254 characters.

ENTRY CONDITIONS:

SP ==> Address of input buffer <long>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
The first byte in the buffer indicates the string length.

EXAMPLE: If A0 points to a 256-byte buffer;

```
        PEA       (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 1.6, Terminal Input/Output Control, is in effect.

5.2.7 .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.
```

Number of Blocks to Transfer	- This field indicates the number of blocks to be read from the disk (.DSKRD) or written to the disk (.DSKWR).
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 <long>           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       (A0)
                  SYSCALL   .DSKRD        Read from disk
                  BNE       ERROR         Branch if error
                  PEA       (A1)
                  SYSCALL   .DSKWR        Write to disk
                  BNE       ERROR         Branch if error
                  "
                  "
                  "
        ERROR     xxxxx     xxx           Handle error
                  xxxxx     xxx
```

5.2.9 .WRITE, .WRITELN FUNCTIONS

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 <long>

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      MESSAGE1(PC)  Push address of string.
            SYSCALL  .WRITE        Use TRAP #15 macro.
            PEA      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.

5.2.8 .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   <long>
        +4     Address of last character+1  <long>
```

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.
```

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.

5.2.10 .WRITD, .WRITDLN FUNCTIONS

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 which has embedded variable fields in it. The user passes the starting address of the string and the address of a data stack from whence the data which is inserted into the string is read. The output goes to the default output port.

ENTRY CONDITIONS:

Eight bytes of parameter positioned in stack as follows:

```
SP ==> Address of string <long>
       Data list pointer <long>
```

A separate data stack or data list arranged as follows:

```
Data list pointer => Data for 1st variable in string <long>
                     Data for next variable          <long>
                     Data for next variable          <long>
                     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       (A5)              Push data stack location.
            PEA       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 2. following).

5.2.12 .ERASLN FUNCTION .ERASLN

TRAP FUNCTION: .ERASLN - Erase Line

CODE: $0027

DESCRIPTION: Erase line is 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
```

5.2.11 .PCRLF FUNCTION

TRAP FUNCTION: .PCRLF - Print <CR><LF> sequence

CODE: $0026

DESCRIPTION: PCRLF 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
```

5.2.14 .DT_INI FUNCTION .DT_INI

TRAP FUNCTION: .DT_INI - Date initialization

CODE: $0041

DESCRIPTION: .DT_INI initializes the MM58274 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:

```
                 +-----------+
                 | | | | | | |
                 |Y|Y|M|M|D|D|
                 | | | | | | |
                 +-----------+
                  |         |
begin buffer ---+         +--- buffer + five bytes
```

ENTRY CONDITIONS:

SP ==> 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 Nov. 18, 1985

```
Data in BUFFER is 3835 3131 3138 or
                  x8x5 x1x1 x1x8.   (x = don't care)

PEA       BUFFER      Put buffer address on stack
SYSCALL   .DT_INI     Initialize date and start clock
```

5.2.13 .TM_INI FUNCTION

TRAP FUNCTION: .TM_INI - Time initialization

CODE: $0040

DESCRIPTION: .TM_INI initializes the MM58274 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:

```
                +-----------+
                | | | | | | |
                |H|H|M|M|S|S|
                | | | | | | |
                +-----------+
                 |         |
 begin buffer ---+         +--- buffer + five bytes
```

ENTRY CONDITIONS:

SP ==> 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

Data in BUFFER is 3134 3035 3332 or
x1x4 x0x5 x3x2. (x = don't care)

```
PEA       BUFFER     Put buffer address on stack
SYSCALL   .TM_INI    Initialize time and start clock
```

5.2.16 .TM_RD FUNCTION .TM_RD

TRAP FUNCTION: .TM_RD - Read the RTC registers

CODE: $0043

DESCRIPTION: .TM_RD is used to read the Real-Time Clock registers. The data returned is in unpacked BCD. The last byte of the returned data is the Status Register; bit #3 indicates whether the RTC has been accessed since the last change of time.

The order of the data in the buffer is:

```
                 +----------------------------+
                 | | | | | | | | | | | | | |  |
                 |Y|Y|M|M|D|D|H|H|M|M|S|S|s|SR|
                 | | | | | | | | | | | | | |  |
                 +----------------------------+
                  |                           |
     begin buffer ---+                           +--- buffer + 13 bytes
```

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 unpacked BCD format.

EXAMPLE: A date and time of May 11, 1985 2:05:32.7 would be returned in the buffer as:

```
0805 0005 0101 0104 0005 0302 070x
```

(where x = don't care)

```
LEA       BUFFER(PC),A0        Fix buffer pointer for result
PEA       (A0)                 Put buffer address on stack
SYSCALL   .TM_RD               Read timer
```

5.2.15 .TM_DISP FUNCTION **.TM_DISP**

TRAP FUNCTION: .TM_DISP - Display time from RTC

CODE: $0042

DESCRIPTION: .TM_DISP displays the date and time from the current cursor position. The format is as follows:

MM/DD/YY hh:mm:ss.s

ENTRY CONDITIONS:

No arguments required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

The cursor is left at the end of the string.

EXAMPLE:

```
SYSCALL  .TM_DISP     Displays the date and time on the console.
```

5.2.18 .REDIR_I, .REDIR_O FUNCTIONS

TRAP FUNCTIONS: .REDIR_I - Redirect input
.REDIR_O - Redirect output

CODES: $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 ==> Port Number <word>
```

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 called
```

EXAMPLE:

```
            MOVE.W   #1,-(SP)      Load port number
            SYSCALL  .REDIR_I      Set it as the new default
```

5.2.17 .REDIR FUNCTION

TRAP FUNCTION: .REDIR - Redirect I/O function

CODE: $0060

DESCRIPTION: .REDIR is 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: ---

NOTES: To use .REDIR, the caller should first allocate space and push the parameters required by the desired I/O function in the stack:

```
        SUBQ.L     #2,A7          Allocate  space  (no  parameters
                                  required by .INCHR)
```

Then the parameters required by .REDIR should be pushed and a call is 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 are popped from the stack:

```
        MOVE.B     (SP)+,D0       Read character
```

The above example reads a character from port 1 using .REDIR.

5.2.20 .BINDEC FUNCTION .BINDEC

TRAP FUNCTION: .BINDEC FUNCTION (Used to calculate the Binary Coded Decimal (BCD) equivalent of the binary number specified)

CODE: $0064

DESCRIPTION: .BINDEC takes a 32-bit unsigned binary number and changes it to an equivalent BCD number.

ENTRY CONDITIONS:

```
SP ==> Argument:Hex number<long>
       Space for result  <2 long>
```

EXIT CONDITIONS DIFFERENT FROM ENTRY:

```
SP ==> Decimal number (2 most significant DIGITS) <long>
                      (8 least significant DIGITS) <long>
```

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
```

5.2.19 .RETURN FUNCTION

TRAP FUNCTION: .RETURN - Return to 133Bug

CODE: $0063

DESCRIPTION: .RETURN is used to return control to 133Bug 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 133Bug.

ENTRY CONDITIONS:

No arguments required.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Control is returned to 133Bug.

EXAMPLE:

```
        SYSCALL   .RETURN        Return to 133Bug.
```

NOTES: .RETURN must be used only by code that was started using 133Bug.

APPENDIX A

MVME133 MODULE STATUS REGISTER (MSR)

In addition to the status and control bits that are implemented in the MC68901, the MVME133 has eight status bits that are read only, have no latching mechanism, and cause no interrupts (with one exception). These bits are called the Module Status Register (MSR). Because of hardware savings, the MSR and the MC68901 are grouped together and appear as a 16-bit word port to the MPU. Therefore, it is important to note that even though the MSR ignores all write accesses, a write to the MSR will affect the MC68901.

The MC68901 appears on the lower byte of the word, and the MSR appears on the upper byte. The bit assignments for the MSR are:

```
 BIT 15   BIT 14   BIT 13   BIT 12   BIT 11   BIT 10   BIT 9    BIT 8
+----------------------------------------------------------------------+
| ACFAIL | SYSCON | PWRUP* | SRBIT4 | SRBIT3 | SRBIT2 | SRBIT1 | SRBIT0 |
+----------------------------------------------------------------------+
```

ACFAIL -	When ACFAIL* is low, this bit is 1. When ACFAIL* is high, it is 0. [ACFAIL] is also an input to the interrupt handler.
SYSCON -	If this module is the system controller, J1 pins 1-2 are connected, and this bit is 1. When it is not the system controller, the pins are open, and this bit is 0.
PWRUP* -	This bit is set to 0 only by Power-Up reset when power is first applied to the MVME133. A read cycle to the real-time clock resets this bit to a 1.
SRBIT4 -	This bit is 0 when J15 pins 9-10 are connected, and is 1 when they are open.
SRBIT3 -	This bit is 0 when J15 pins 7-8 are connected, and is 1 when they are open.
SRBIT2 -	This bit is 0 when J15 pins 5-6 are connected, and is 1 when they are open.
SRBIT1 -	This bit is 0 when J15 pins 3-4 are connected, and is 1 when they are open.
SRBIT0 -	This bit is 0 when J15 pins 1-2 are connected, and is 1 when they are open.

The MSR appears in the main memory map of the MVME133 at locations $XXF80000 through $XXF8002E, even bytes only, and is repeated throughout locations $XXF80030 through $XXF9FFFE. The MC68901 Multi-Function Peripheral (MFP) occupies the odd bytes. Even though the MSR is read-only, it should not be written to, because cycles that access the MSR also access the MFP.

5.2.21 .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          <long>
       Ending address + 1         <long>
       Space for checksum         <word>
```

EXIT CONDITIONS DIFFERENT FROM ENTRY:

```
SP ==> Checksum                   <word>
```

EXAMPLE:

```
CLR.W    -(SP)         Make room for the checksum.
PEA      A1            Push pointer to ending address + 1.
PEA      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 bug 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.

APPENDIX B

DEBUGGING PACKAGE 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 IOT command do not correspond to a valid controller and device.
*** Invalid Range ***	Range entered wrong in BF, BI, BM, BS, or DU commands.
<part of S-record data>	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.
<unassembled line> ----------------^ *** Unknown Field ***	Message and pointer ("^") to field of suspected error when using ;DI option in MM command.
Verify passes	Successful VE command.
VMEbus Bus Time-out	Referenced nonexisting memory in MD command.

THIS PAGE INTENTIONALLY LEFT BLANK.

DIAGNOSTIC ERROR MESSAGES (cont'd)	MEANING
RTC did not interrupt	RTC Test error message.
SIO Receive Error Code $XX	MFP Funct. Test error message. $XX is contents of MC68901 Receiver Status Register.
Test failed FPC routine at $XXXXXXXX	FPC Test error message. $XXXXXXXX is address of part of test that failed.
Test failed routine at $XXXXXXXX	MPU Register, Instruction, or Address Mode Test error message. $XXXXXXXX is the address of the part of the test that failed.
Test Failed Vector $XX	MPU Exception Processing Test error message. $XX is the exception vector offset.
Tx/Rx Problem; did not get expected interrupt	MFP Functionality Test error message.
Unexpected Bus Error	MPU Address Mode, MFP Functionality, RTC, or Z8530 Functionality Test error message.
Unexpected Exception Taken	MPU Exception Processing Test error message.
Unexpected interrupt	FPC Test error message.
Unexpected MFP Interrupt(interrupt was disabled)	MFP Functionality Test error message.
Unit Sec. greater than 1	RTC Test error message.
(Unit/Tens) <time> <> (0,1)	RTC Test error message. <time> goes from Sec. to Years.

OTHER MESSAGES	MEANING
133Bug>	Debugger prompt.
133Diag>	Diagnostic prompt.
At Breakpoint	Indicates program has stopped at breakpoint.
!!Break!!	BREAK key on console has stopped operation.
COLD Start	Vectors have been initialized.

DIAGNOSTIC ERROR MESSAGES	**MEANING**
68901 Register test failed	Power-up test error message.
Bad SIO Xfer: expected $YY,received $ZZ	MFP Functionality Test error message.
N CACHE (HITS!/MISSES!) CACHED IN XXXX MODE, RERAN IN XXXX MODE ..... FAILED	MC68020 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.
Day not 1	RTC Test error message.
Did not receive interrupt from Timer (A/D)	MFP Functionality Test error message.
Error Code = $XXXX	Z8530 Functionality Test error message. $XXXX is explained in the test writeup.
Exception Processing test failed	Power-up test error message.
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.
Got Bus Error when reading from ROM	Bus Error Test error message.
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.
Interrupt flag not set in Status Reg.	RTC Test error message.
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 MVME133 module.
RAM test failed	Power-up test error message.
ROM test failed	Power-up test error message.

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:

FIELD	PRINTABLE CHARACTERS	CONTENTS
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.

OTHER MESSAGES (cont'd)	MEANING
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.
Logical unit $XX unassigned	Message that may be output during PA or PF commands. $XX is a hex number indicating the port involved.
No printer attached	Message that may be output during NOPA command.
-not found-	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 command when more than 24 lines of matches are found.

UPLOAD "S" RECORDS
Version x.y
Copyrighted by MOTOROLA, INC.
 	
volume=xxxx
catlg=xxxx
file=FILE1
ext=MX
	Message from VERSAdos UPLOADS utility during DU command.

WARM Start	Vectors have not been initialized.

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 BUILDS, 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. Programs are also available for uploading an S-record file to or from an EXORmacs 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.

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. 133Bug 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.

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 00	The address field, zeroes.
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:

TYPE				LENGTH				ADDRESS								CODE/DATA									CHECKSUM			
S		1		1		3		0		0		0		0		2		8		5		F		...	2		A	
5	3	3	1	3	1	3	3	3	0	3	0	3	0	3	0	3	2	3	8	3	5	4	6	...	3	2	4	1
0101	0011	0011	0001	0011	0001	0011	0011	0011	0000	0011	0000	0011	0000	0011	0000	0011	0010	0011	1000	0011	0101	0100	0110	...	0011	0010	0100	0001

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 00	Four-character 2-byte address field, zeroes in this example.
48 44 52	ASCII H, D, and R - "HDR".
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 00	Four-character 2-byte address field; hexadecimal address 0000, where the data which follows is to be loaded.

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:

```
OPCODE          INSTRUCTION

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.

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.
IOSPSM	$1E (30)	2	Physical sector size of media in bytes.
IOSSOF	$1D (29)	1	Spiral offset.
IOSSHD	$20 (32)	2	Starting head number.
IOSPCOM	$24 (36)	2	Precompensation cylinder.
IOSSR	$27 (39)	1	Stepping rate code.
IOSRWCC	$28 (40)	2	Reduced write current cylinder number.
IOSECC	$2A (42)	2	ECC data burst length.

THIS PAGE INTENTIONALLY LEFT BLANK.

IOSPRM and IOSEPRM

A "1" in a particular bit position indicates that the corresponding parameter from the configuration area (CFGA) 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 3.
IOASSC	4	Spare sector count.

CONFIGURATION AREA BLOCK (CFGA) (cont'd)

CONFIGURATION AREA BLOCK #1 (CFGA)

LABEL	OFFSET$(&)	LENGTH (BYTES)	CONTENTS
IOSEATM	$2C (44)	2	Extended attributes mask.
IOSEPRM	$2E (46)	2	Extended parameters mask.
IOSEATH	$30 (48)	2	Extended attributes word.
IOSGPB1	$32 (50)	1	Gap byte 1.
IOSGPB2	$33 (51)	1	Gap byte 2.
IOSGPB3	$34 (52)	1	Gap byte 3.
IOSGPB4	$35 (53)	1	Gap byte 4.
IOSSSC	$36 (54)	1	Spare sectors count.

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.

PARAMETER FIELD DEFINITIONS

PARAMETER	DESCRIPTION
Record Size (Block)	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

IOSATW attributes word. 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	Imbedded 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
Bits 10-15	Reserved (must be zero).

APPENDIX E

DISK CONTROLLER DATA

DISK CONTROLLER MODULES SUPPORTED

The following VMEbus disk controller modules are supported by 133Bug:

1. MVME320 - VMEbus Winchester/Floppy Controller
2. MVME319 - VMEbus SCSI/Floppy/Tape Controller
3. MVME360 - VMEbus SMD Controller

DISK CONTROLLER DEFAULT CONFIGURATIONS

```
Controller LUN 0
  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


Controller LUN 1
  Controller Type    : MVME319
  Controller Address : $FFFF0000
  Number of Devices  : 8
  Devices            : DLUN 0 = 40Mb Winchester (NOTE)                WIN40
                       DLUN 1 = 40Mb Winchester (NOTE)                WIN40
                       DLUN 2 = 40Mb Winchester (NOTE)                WIN40
                       DLUN 3 = 40Mb Winchester (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.

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.

D

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:

```
 15                          8 7                      0
+-------------------------------------------------------+
|  Controller-Independent   |   Controller-Dependent    |
+-------------------------------------------------------+
```

Controller-Independent Status Codes

$00	No error detected.
$01	Invalid controller type.
$02	Controller descriptor not found.
$03	Device descriptor not found.
$04	Controller already attached.
$05	Descriptor table not available.
$06	Invalid packet.
$07	Invalid address for transfer.

MVME320 Controller-Dependent Status Codes

$0	Correct execution without error.
$1	Nonrecoverable error which cannot be completed (auto retries were attempted).
$2	Drive not ready.
$3	Reserved.
$4	Sector address out of range.
$5	Throughput error (floppy data overrun).
$6	Command rejected (illegal command).
$7	Busy (controller busy).
$8	Drive not available (head out of range).
$9	DMA operation cannot be completed (VMEbus error).
$A	Command abort (reset busy).
$B-$FF	Not used.

```
Controller LUN 2
  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 = 2322K Fujitsu SMD drive (512b sectors) FJI10
                       DLUN 3 = null device (SMD half)                 SMDHALF

Controller LUN 3
  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
```

E

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.

MVME360 Controller-Dependent Status Codes (cont'd)

$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	Mapped header error.
$4D-4F	Not used.
$50	Sector per track error.
$51	Bytes per sector specification error.
$52	Interleave specification error.
$53	Invalid head address.
$54	Invalid cylinder address.
$55-5C	Not used.
$5D	Invalid DMA transfer count.
$5E-5F	Not used.
$60	IOPB failed.
$61	DMA failed.
$62	Illegal VME address.
$63-69	Not used.
$6A	Unrecognizable 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.

MVME319 Controller-Dependent Status Codes

$0	Correct execution without error.
$1	Data CRC/ECC error.
$2	Disk write protected.
$3	Drive not ready.
$4	Deleted data mark read.
$5	Invalid drive number.
$6	Invalid disk address.
$7	Restore error.
$8	Record not found.
$9	Sector ID CRC/ECC error.
$A	VMEbus DMA error.
$F	Controller error.
$10	Drive error.
$11	Seek error.
$19	I/O DMA error.

MVME360 Controller-Dependent Status Codes

$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.

F

APPENDIX G

133Bug DIAGNOSTIC FIRMWARE GUIDE

TABLE OF CONTENTS

F

THIS PAGE INTENTIONALLY LEFT BLANK.

G.1 SCOPE

This diagnostic guide contains information about the operation and use of the MVME133 Diagnostic Firmware Package, hereafter referred to as "the diagnostics". Paragraphs G.3 and G.4 give the user guidance in setting up the system and invoking the various utilities and tests. G.5 describes utilities available to the user. G.6 through G.13 are guides to using each test.

G.2 OVERVIEW OF DIAGNOSTIC FIRMWARE

The MVME133 diagnostic firmware package consists of two 64K x 8 EPROMs which are plugged into the MVME133. These two EPROMs (which also contain 133Bug) contain a complete diagnostic monitor along with a battery of utilities and tests for exercise, test, and debug of hardware in the MVME133 environment.

The diagnostics are menu-driven for ease of use. The Help (HE) command (explained fully in paragraph G.4.3) 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.

G.3 SYSTEM START-UP

For 133Bug diagnostics to operate properly, follow this setup procedure.

CAUTION

INSERTING/REMOVING MODULES WITH POWER APPLIED COULD DAMAGE COMPONENTS.

1. Turn all equipment power OFF. Refer to the MVME133 User's Manual and configure the header jumpers on the module as required for the user's particular application. The only jumper configurations specifically dictated by 133Bug are those on J6. J6 must have jumpers between pins 1 and 3, and between pins 4 and 6. This sets EPROM sockets XU31 and XU46 for 64K x 8 chips in bank 1. (This may **NOT** be the factory configuration of the MVME133 as shipped.)

2. Refer to the MVME133 User's Manual and configure the Module Status Register (MSR) as required for the user's particular application. J15 sets or resets bits 0 through 4 of the MSR, and J1 enables or disables the system controller functions of the MVME133 thus allowing the SYSCON bit of the MSR to reflect system controller status (details in Appendix A).

3. Be sure that the two 64K x 8 133Bug diagnostic EPROMs are installed in sockets XU31 (odd bytes, odd BXX label) and XU46 (even bytes, even BXX label) on the MVME133 module. BE SURE CHIP ORIENTATION IS CORRECT, WITH PIN 1 LINED UP WITH THE MARKS ON THE SILKSCREEN ON THE PC BOARD.

TABLE OF CONTENTS (cont'd)

G

```
... 'CPU Register test failed'
... 'CPU Instruction test failed'
... 'ROM test failed'
... 'RAM test failed'
... 'CPU Addressing Modes test failed'
... 'Exception Processing test failed'
... '68901 Register test failed'
```

Control remains with the confidence test and the monitor does not come up. The user may force the monitor to come up by pressing the ABORT switch on the MVME133 front panel.

G.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.

G.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 (debug port terminal):

```
Copyright Motorola Inc. 1986, All Rights Reserved

VME133 Monitor/Debugger Version 1.0 - 5/1/86

COLD Start
133Bug>
```

At the prompt, enter "SD" to switch to the diagnostics directory. Switch Directories (SD) is explained in detail in paragraph G.4.5. The prompt should now read "133Diag>".

G.4.2 Command Entry and Directories

Entry of commands is made when the prompt "133Diag>" 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.

4. Refer to the set-up procedure for the user's particular chassis or system for details concerning the installation of the MVME133.

5. Connect the terminal which is to be used as the 133Bug system console to the debug port connector J14 (DB25 connector) on the MVME133 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 MVME133 debug port at power-up.

   In order for high-baud rate serial communication between 133Bug 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 (EXORterms do hardware handshaking), 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 up some device(s) (such as a host computer system or a serial printer) to port A (RS-485/RS-422) and/or port B (RS-232C) on the MVME133 rear connector J2, connect the appropriate cables and configure the port(s) as detailed in the MVME133 User's Manual. After power-up, these ports can be reconfigured by programming the Z8530 chip on the MVME133, or by using the Port Format (PF) command of the 133Bug debugger.

7. Power up the system. 133Bug executes some self-checks and displays the debugger prompt ("133Bug>"). (Refer to paragraph G.4.1.)

   When power is applied to the MVME133, bit 13 location $F80000 (Module Status Register = MSR) is set to zero indicating that power was just applied. (Refer to Appendix A for a complete description of the MSR.) This bit is tested within the "Reset" logic path to see if the power-up confidence test needs to be executed. Location $FB0000 (Real-Time Clock = RTC) is read, thereby setting the power-up indicator to a one thus preventing any future power-up confidence test execution.

   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:

G.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.

G.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.

G.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 MVME133 front panel may become necessary.

G.4.9 NON-VERBOSE MODE - Prefix "NV"

Upon Detecting an error, the tests included for the MVME133 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).

G.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.

G.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.

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 "CA20". These commands are used to refer to a set of lower level commands.

To call up a particular 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 MC68020
		A	Register Test
	CA20		MC68020 Onchip Cache Tests
		G	Unlike Function Codes

G.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.

G.4.4 SELF-TEST - Prefix/Command "ST"

The monitor provides an automated test mechanism called self-test. Entering "ST" before a command causes the monitor to run the tests included in an internal self-test directory. "ST" without any parameters runs most of the MVME133 diagnostics.

Each test for that particular command is listed in the paragraph pertaining to the command (i.e., refer to paragraph G.8 for the MT commands whether or not they are included in the self-test chain).

G.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 133Bug function. When in the 133Bug directory, the prompt reads "133Bug>". To return to the diagnostic directory, the command "SD" is entered again. When in the diagnostic directory, the prompt reads "133Diag>". The purpose of this feature is to allow the user to access 133Bug without the diagnostics being visible.

G.6 MPU TESTS FOR THE MC68020 - Command "MPU"

G.6.1 General Description

This paragraph details the diagnostics provided to test the MC68020 MPU.

TABLE G-1. MC68020 MPU Diagnostic Tests

MONITOR COMMAND	TITLE	PARAGRAPH
MPU A	Register Test	G.6.3
MPU B	Instruction Test	G.6.4
MPU C	Address Mode Test	G.6.5
MPU D	Exception Processing Test	G.6.6

The normal procedure for fixing an MC68020 MPU error is to replace the MPU.

G.6.2 Hardware Configuration

The following hardware is required for using these tests:

MVME133 - Module being tested
VME chassis
Video display terminal

G.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 (refer to paragraph G.4.8). To display this information without using "LC", enter "DP".

G.4.13 ZERO PASS COUNT - Command "ZP"

Invoking this command resets the pass counter "DP" (described in paragraph G.4.12) 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.

G.5 UTILITIES

The monitor is supplemented by several utilities that are separate and distinct from the monitor itself and the diagnostics.

G.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 0 if the first digit is other than 0-9, i.e., $FF would be entered as "0FF". To write $00 out to address $FFFB0030, enter "WL.B 0FFFB0030 00". Omission of either or both parameters causes prompting for the missing values.

G.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 $FFFB0030, enter "RL.B FFFB0030". Omission of the parameter causes prompting for the missing value.

G.6.4 MPU B - Instruction Test

G.6.4.1 Description. This command tests various data movement, integer arithmetic, logical, shift and rotate, and bit manipulation instructions of the MC68020 chip.

G.6.4.2 Command Input.

```
133Diag>MPU B
```

G.6.4.3 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
Test failed 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 EPROM to determine exactly what instruction failed.

If all parts of the test are completed correctly, then the test passes.

```
B     MPU Instruction Test.....................Running ----------> PASSED
```

G.6.3 MPU A - Register Test

G.6.3.1 Description. This command does a thorough test of all the registers in the MC68020 chip, including checking for bits stuck high or low.

G.6.3.2 Command Input.

```
133Diag>MPU A
```

G.6.3.3 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
Test failed 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 EPROM to determine exactly what register failed.

If all parts of the test are completed correctly, then the test passes.

```
A      MPU Register test.......................Running ----------> PASSED
```

G.6.6 MPU D - Exception Processing Test MPU D

G.6.6.1 Description. This command tests many of the exception processing routines of the MC68020, but not the interrupt auto vectors or any of the floating-point coprocessor vectors.

G.6.6.2 Command Input.

```
133Diag>MPU D
```

G.6.6.3 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 $XX
```

Here $XX is the hexadecimal exception vector offset, as explained in the MC68020 User's Manual, MC68020UM.

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
```

If all parts of the test are completed correctly, then the test passes.

```
D     MPU Exception Processing Test...........Running ----------> PASSED
```

G

G.6.5 MPU C - Address Mode Test

G.6.5.1 Description. This command tests the various addressing modes of the MC68020 chip. These include absolute address, address indirect, address indirect with postincrement, and address indirect with index modes.

G.6.5.2 Command Input.

```
133Diag>MPU C
```

G.6.5.3 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
Test failed 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 EPROM to determine exactly what address mode failed.

The only other possible error message display is:

```
C     MPU Address Mode test....................Running ---------->..... FAILED
Unexpected Bus Error
```

If all parts of the test are completed correctly, then the test passes.

```
C     MPU Address Mode test....................Running ----------> PASSED
```

G.7.3 CA20 F - Basic Caching Test CA20 F

G.7.3.1 Description. This command tests the basic caching function of the MC68020 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 MC68020 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.

G.7.3.2 Command Input.

```
133Diag>CA20 F
```

G.7.3.3 Response/Messages. After the command has been issued, the following line is printed:

```
F     CA20 Basic 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     CA20 Basic 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     CA20 Basic caching ......................Running ----------> PASSED
```

G.7 MC68020 ONCHIP CACHE TESTS - Command "CA20"

G.7.1 General Description

This paragraph details the diagnostics provided to test the MC68020 cache.

TABLE G-2. MC68020 Cache Diagnostic Tests

MONITOR COMMAND	TITLE	PARAGRAPH
CA20 F	Basic Caching	G.7.3
CA20 G	Unlike Function Codes	G.7.4
CA20 H	Disable Test	G.7.5
CA20 I	Clear Test	G.7.6

The normal procedure for fixing an MC68020 cache error is to replace the MPU.

G.7.2 Hardware Configuration

The following hardware is required for using these tests:

MVME133 - Module being tested
VME chassis
Video display terminal.

G.7.5 CA20 H - Disable Test

G.7.5.1 Description. In the MC68020 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.

G.7.5.2 Command Input.

```
133Diag>CA20 H
```

G.7.5.3 Response/Messages. After the command has been issued, the following line is printed:

```
H     CA20 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     CA20 Disable test ......................Running ---------->..... FAILED
1 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     CA20 Disable test ......................Running ----------> PASSED
```

G.7.4 CA20 G - Unlike Function Codes Test

G.7.4.1 <u>Description</u>. This command tests the ability of the onchip cache to recognize 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 MC68020 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.

G.7.4.2 <u>Command Input</u>.

```
133Diag>CA20 G
```

G.7.4.3 <u>Response/Messages</u>. After the command has been issued, the following line is printed:

```
G      CA20 Unlike 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      CA20 Unlike 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      CA20 Unlike fn. codes ...................Running ----------> PASSED
```

G

G.8 MEMORY TESTS - Command "MT"

G.8.1 General Description

This set of tests accesses random access memory (read/write) that may or may not reside on the MVME133 module. Default is the onboard RAM. To test offboard RAM, change Start and Stop Addresses per MT B and MT C as described following.

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 G-3. Memory Diagnostic Tests

MONITOR COMMAND	TITLE	PARAGRAPH
MT A	Set Function Code	G.8.3
MT B	Set Start Address	G.8.4
MT C	Set Stop Address	G.8.5
MT D	Set Bus Data Width	G.8.6
MT E	March Address Test	G.8.7
MT F	Walk a Bit Test	G.8.8
MT G	Refresh Test	G.8.9
MT H	Random Byte Test	G.8.10
MT I	Program Test	G.8.11
MT J	TAS Test	G.8.12

G.8.2 Hardware Configuration

The following hardware is required to perform these tests.

MVME133 - Module being tested
VME chassis
Video display terminal
Optional offboard memory.

G.7.6 CA20 I - Clear Test

G.7.6.1 <u>Description</u>. A control bit is included in the MC68020 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.

G.7.6.2 <u>Command Input</u>.

```
133Diag>CA20 I
```

G.7.6.3 <u>Response/Messages</u>. After the command has been issued, the following line is printed:

```
I     CA20 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     CA20 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     CA20 Clear test ........................Running ----------> PASSED
```

G.8.4 MT B - Set Start Address

G.8.4.1 Description. This command allows the user to select the start address used by all of the memory tests. For a system with one MVME133, it is suggested that address $00004000 be used. For a system with two MVME133's, the address $00008000 should be used. Other addresses may be used, but extreme caution should be used when attempting to test memory below these addresses.

G.8.4.2 Command Input.

```
133Diag>MT B [<new value>] <CR>
```

G.8.4.3 Response/Messages. If the user supplied the optional new value, then the display appears as follows:

```
133Diag>MT B [<new value>] <CR>
Start Addr.=<new value>
133Diag>
```

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.=00003000, which is for onboard RAM.

```
133Diag>MT B <CR>
Start Addr.=<current value> ?[<new value>] <CR>
Start Addr.=<new value>
133Diag>
```

This command may be used to display the current value without changing it by pressing a carriage return <CR> without entering the new value.

```
133Diag>MT B <CR>
Start Addr.=<current value> ?<CR>
Start Addr.=<current value>
133Diag>
```

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.

G.8.3 MT A - Set Function Code

G.8.3.1 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".

G.8.3.2 Command Input.

```
133Diag>MT A [<new value>] <CR>
```

G.8.3.3 Response/Messages. If the user supplied the optional new value, then the display appears as follows:

```
133Diag>MT A [<new value>] <CR>
Function Code=<new value>
133Diag>
```

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.

```
133Diag>MT A <CR>
Function Code=<current value> ?[<new value>] <CR>
Function Code=<new value>
133Diag>
```

This command may be used to display the current value without changing it by pressing a carriage return <CR> without entering the new value.

```
133Diag>MT A <CR>
Function Code=<current value> ?<CR>
Function Code=<current value>
133Diag>
```

G.8.6 MT D - Set Bus Data Width MT D

G.8.6.1 Description. This command is used to select either 16- or 32-bit bus data accesses during the MVME133 MT memory tests. The width is selected by entering zero for 16 bits or one for 32 bits.

G.8.6.2 Command Input.

```
133Diag>MT D [<new value: 0 for 16, 1 for 32>] <CR>
```

G.8.6.3 Response/Messages. If the user supplied the optional new value, then the display appears as follows:

```
133Diag>MT D [<new value>] <CR>
Bus Width (32=1/16=0) =<new value>
133Diag>
```

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 value is Bus Width (32=1/16=0) =0, which is for onboard RAM.

```
133Diag>MT D <CR>
Bus Width (32=1/16=0) =<current value> ?[<new value>] <CR>
Bus Width (32=1/16=0) =<new value>
133Diag>
```

This command may be used to display the current value without changing it by pressing a carriage return <CR> without entering the new value.

```
133Diag>MT D <CR>
Bus Width (32=1/16=0) =<current value> ?<CR>
Bus Width (32=1/16=0) =<current value>
133Diag>
```

G.8.5 MT C - Set Stop Address

G.8.5.1 <u>Description</u>. This command allows the user to select the stop address used by all of the memory tests.

G.8.5.2 <u>Command Input</u>.

```
133Diag>MT C [<new value>] <CR>
```

G.8.5.3 <u>Response/Messages</u>. If the user supplied the optional new value, then the display appears as follows:

```
133Diag>MT C [<new value>] <CR>
Stop Addr.=<new value>
133Diag>
```

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.=000FFFFC, which is for onboard RAM.

```
133Diag>MT C <CR>
Stop Addr.=<current value> ?[<new value>] <CR>
Stop Addr.=<new value>
133Diag>
```

This command may be used to display the current value without changing it by pressing a carriage return <CR> without entering the new value.

```
133Diag>MT C <CR>
Start Addr.=<current value> ?<CR>
Start Addr.=<current value>
133Diag>
```

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.

G.8.8 MT F - Walk a Bit Test

G.8.8.1 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.

G.8.8.2 Command Input.

```
133Diag>MT F <CR>
```

G.8.8.3 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 (refer to paragraph G.8.13).

```
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
```

G.8.7 MT E - March Address Test MT E

G.8.7.1 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.

G.8.7.2 Command Input.

```
133Diag>MT E <CR>
```

G.8.7.3 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 (refer to paragraph G.8.13).

```
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
```

G.8.10 MT H - Random Byte Test

G.8.10.1 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.

G.8.10.2 Command Input.

```
133Diag>MT H <CR>
```

G.8.10.3 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 (refer to paragraph G.8.13).

```
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
```

G.8.9 MT G - Refresh Test

G.8.9.1 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.
```

G.8.9.2 Command Input.

```
133Diag>MT G <CR>
```

G.8.9.3 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 (refer to paragraph G.8.13).

```
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
```

G.8.12 MT J - TAS Test

G.8.12.1 __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).
```

G.8.12.2 __Command Input__.

```
133Diag>MT J <CR>
```

G.8.12.3 __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 (refer to paragraph G.8.13).

```
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
```

G.8.11 MT I - Program Test

MT I

G.8.11.1 **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 G.8.13).

G.8.11.2 **Command Input.**

```
133Diag>MT I <CR>
```

G.8.11.3 **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 (refer to paragraph G.8.13).

```
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
```

G.9 REAL-TIME CLOCK TEST - Command "RTC" RTC

G.9.1 Description

This command tests the MM58274A Real-Time Clock (RTC). The RTC is started and, after five seconds, provides a level 4 interrupt to the MC68020 MPU via the onboard interrupt handler. The RTC is then checked for roll-over.

G.9.2 Command Input

```
133Diag>RTC
```

G.9.3 Response/Messages

After the command has been issued, the following line is printed:

```
RTC   Real Time Clock Test.....................Running ---------->
```

If there is no interrupt, then the display appears as follows.

```
RTC   Real Time Clock Test.....................Running ---------->..... FAILED
RTC did not interrupt
```

If an interrupt is generated, but it does not reset PWRUP* (bit 13 of the onboard Module Status Register = MSR), then the display appears as follows. (Refer to Appendix A for details of the MSR.)

```
RTC   Real Time Clock Test.....................Running ---------->..... FAILED
Interrupt flag not set in Status Reg.
```

If any digit is wrong in roll-over, then the test fails and the appropriate error message appears as one of the following:

```
Unit Sec. greater than 1          Unit Days <> 1
Tens Sec. <> 0                    Tens Days <> 0
Unit Min. <> 0                    Unit Month <> 1
Tens Min. <> 0                    Tens Month <> 0
Unit Hours <> 0                   Unit Years <> 0
Tens Hours <> 0                   Tens Years <> 0
                                  Day not 1
```

If a bus error occurs, the error message is:

```
Unexpected Bus Error
```

If both parts of the test are completed correctly, then the test passes.

```
RTC   Real Time Clock Test.....................Running ----------> PASSED
```

G.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 J.

The error reporting code is designed to conform to two rules:

1. The first time an error occurs, headings are printed out prior to the printing of the values.

2. 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----  FFFFEFFF  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.

G.11 FLOATING POINT COPROCESSOR (MC68881) TEST - Command "FPC" FPC

G.11.1 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).

G.11.2 Command Input

```
133Diag>FPC
```

G.11.3 Response/Messages

After the command has been issued, the following line is printed:

```
FPC   Floating Pnt. Coprocessor Test...........Running ---------->
```

If there is no FPC on the MVME133 module, then the test fails and 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 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
```

G.10 BUS ERROR TEST - Command "BERR"

G.10.1 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

G.10.2 Command Input

```
133Diag>BERR
```

G.10.3 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 six parts of the test are completed correctly, then the test passes.

```
BERR  Bus Error Test...........................Running ----------> PASSED
```

G.13 Z8530 FUNCTIONALITY TEST - Command "SCC"

G.13.1 Description

This command initializes the Z8530 chip for Tx and Rx interrupts, and local loopback mode. Using interrupt handlers, it transmits, receives, and verifies data until all transmitted data is verified or a time-out occurs.

G.13.2 Command Input

```
133Diag>SCC
```

G.13.3 Response/Messages

After the command has been issued, the following line is printed:

```
SCC   Z8530 Functionality Test.................Running ---------->
```

If any part of the test fails, a time-out eventually occurs, and then the test fails and the display appears as follows.

```
SCC   Z8530 Functionality Test.................Running ---------->..... FAILED
Error Code = $XXXX
```

$XXXX is defined below. Multiple errors are shown as the sum of individual errors; for example, $0003 means RS-485 transmit and external/status errors.

$0001	RS-485 transmit error	error code 0
$0002	RS-485 external/status change	error code 1
$0004	RS-485 receive error	error code 2
$0008	RS-485 special Rx condition	error code 3
$0010	RS-485 transmit time-out	error code 4
$0020	RS-485 receive time-out	error code 5
$0040	not used	error code 6
$0080	not used	error code 7
$0100	RS-232C transmit error	error code 8
$0200	RS-232C external/status change	error code 9
$0400	RS-232C receive error	error code 10
$0800	RS-232C special Rx condition	error code 11
$1000	RS-232C transmit time-out	error code 12
$2000	RS-232C receive time-out	error code 13
$4000	not used	error code 14
$8000	not used	error code 15

The only other possible error message is:

```
Unexpected Bus Error
```

If all parts of the test are completed correctly, then the test passes.

```
SCC   Z8530 Functionality Test.................Running ----------> PASSED
```

G.12 MFP (MC68901) FUNCTIONALITY TEST - Command "MFP" MFP

G.12.1 Description

This command tests three aspects of the MC68901 Multi-Function Peripheral (MFP). Interrupts are tested in both the serial I/O (SIO) test and in the timer test. Second, the SIO test outputs a message through the port using the SIO interrupts. Finally, the timer test cascades timers D, A, and B for a watchdog timer output but stops the watchdog from timing out to prevent system reset. (It may time out if the device fails to interrupt on two separate channels.)

G.12.2 Command Input

```
133Diag>MFP
```

G.12.3 Response/Messages

After the command has been issued, the following line is printed:

```
MFP   MFP Functionality Test..................Running ---------->
```

If any part of the test fails , then the display appears as follows.

```
MFP   MFP Functionality Test..................Running ---------->.... FAILED
(error message)
```

Here, (error message) is one of the following:

```
Unexpected Bus Error
Unexpected MFP Interrupt(interrupt was disabled)
Tx/Rx Problem; did not get expected interrupt
SIO Receive Error Code $XX
Bad SIO Xfer: expected $YY,received $ZZ
Did not receive interrupt from Timer A
Did not receive interrupt from Timer D
```

Here $XX, $YY, and $ZZ are hexadecimal numbers. The receive error code, $XX, is the contents of the Receiver Status Register, and is described in detail in the MC68901 data sheet, ADI-984.

If all three parts of the test are completed correctly, then the test passes.

```
MFP   MFP Functionality Test..................Running ----------> PASSED
```

INDEX

G

THIS PAGE INTENTIONALLY LEFT BLANK.

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

SUGGESTION/PROBLEM REPORT

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Inc.
Microcomputer Division
2900 S. Diablo Way
Tempe, Arizona 85282
Attention: Publications Manager
Maildrop DW164

Product: ______________________ Manual: ______________________

COMMENTS: ______________________

(For additional comments use other side)

Please Print

Name ______________	Title ______________
Company ______________	Division ______________
Street ______________	Mail Drop ______________
City ______________	Phone ______________
State ________ Zip ________	Country ______________

For Additional Motorola Publications
Literature Distribution Center
616 West 24th Street
Tempe, AZ 85282
(602) 994-6561

Motorola Field Service Division/Customer Support
(800) 528-1908
(602) 438-3100

MOTOROLA

COMMENTS:

MOTOROLA

MOTOROLA INC.

Microcomputer Division
2900 South Diablo Way
Tempe, Arizona 85282
P.O. Box 2953
Phoenix, Arizona 85062

Motorola is an Equal Employment
Opportunity/Affirmative Action Employer

Motorola and are registered
f Motorola, Inc.