MVME135BUG/D2

MVME135 Debug Monitor
135Bug Debugging Package

MOTOROLA

PRELIMINARY

MVME135BUG/D2

MARCH 1988

MVME135 DEBUG MONITOR

135Bug DEBUGGING PACKAGE

Second Edition

First Edition April 1987

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
GENERAL INFORMATION

1.1 Description of 135Bug

The 135Bug package (MVME135BUG) is a powerful evaluation and debugging tool for systems built around the MVME135 processor module. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 135Bug 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 135Bug routines that handle I/O, data conversion, and string functions are available to user programs through the TRAP #15 handler.

135Bug 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 VME135 hardware, described in the MVME135 Diagnostic Firmware User's Guide (Motorola Publication MVME135DIAG) and hereafter referred to as the **diagnostics**, and (3) a user interface which accepts commands from the system console terminal.

When using 135Bug the user will either operate out of the debugger directory or out of the diagnostic directory. If the user is in the debugger directory then the debugger prompt **135Bug>** , will be displayed and the user will have all of the debugger commands at his disposal. If the user is in the diagnostic directory then the diagnostic prompt **135Diag>** , will be displayed and the user will have all of the diagnostic commands at his/her disposal as well as all of the debugger commands. The user may switch between directories by using the **SD** command, described in Chapter 3, or may examine the commands in the particular directory that he/she is currently in by using the **HE** command, also described in Chapter 3.

Since 135Bug 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 135Bug. When a command is entered, 135Bug will execute the command and the prompt will reappear. However, if a command is entered which causes execution of user target code (i.e., **GO**) then control may or may not return to 135Bug, depending on the outcome of the user program.

Those users who have used one or more of Motorola's other debugging packages will find 135Bug very similar. There are two noticeable differences. Many of the commands are more flexible and powerful. Also, the debugger in general is more "user-friendly", with more detailed error messages and an expanded on-line help facility.

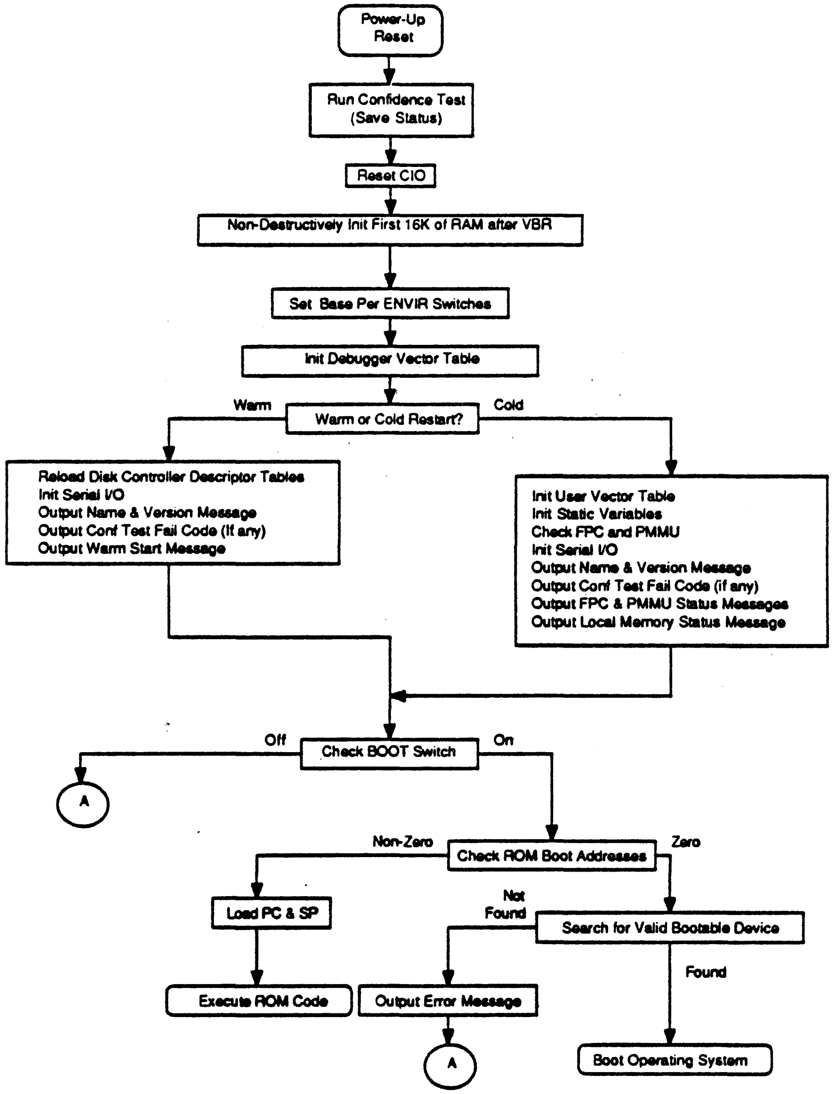

FIGURE 1-1. FLOW DIAGRAM OF 135Bug OPERATION MODE

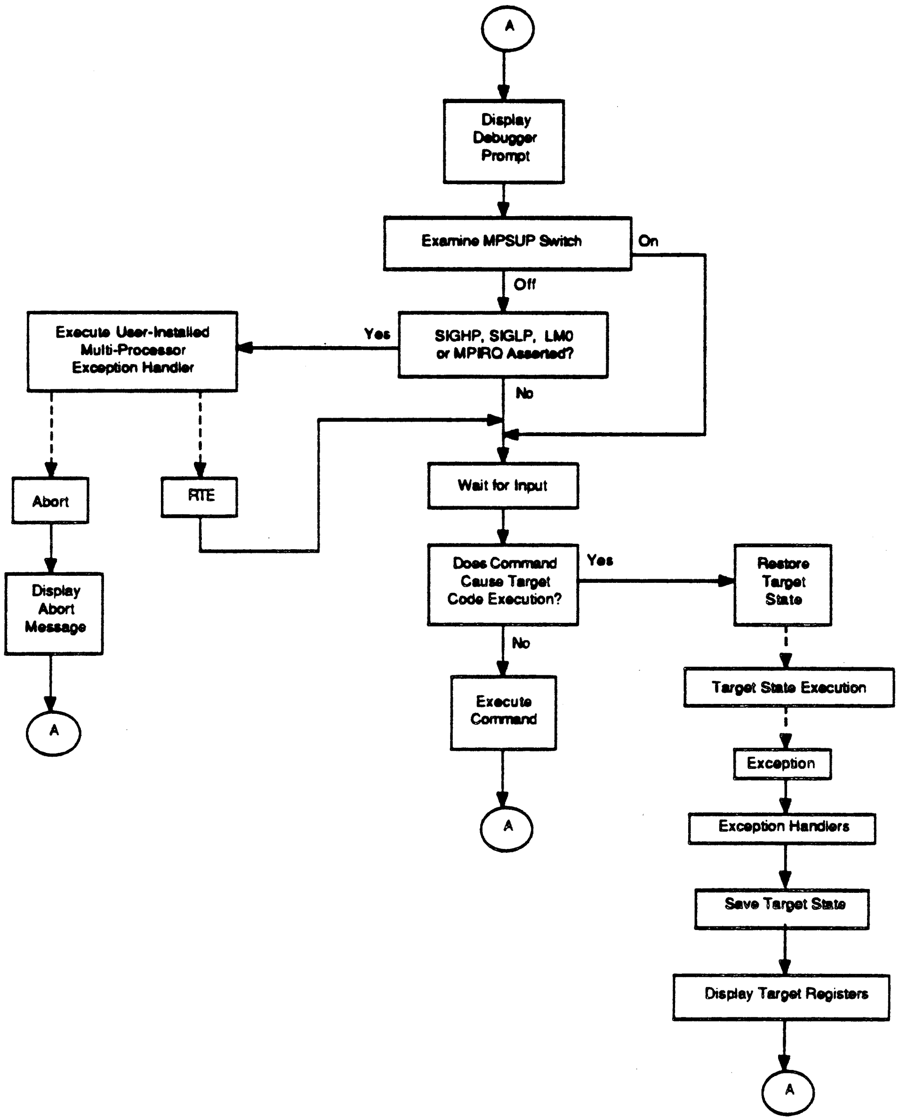

FIGURE 1-1. FLOW DIAGRAM OF 135Bug OPERATION MODE (cont.)

1.2 How To Use This Manual

If the user has never used a debugging package before, then he/she should read all of Chapters 1 and 2 before attempting to use 135Bug. This will give an idea of 135Bug's structure and capabilities.

Section 1.3, entitled "Installation and Start-up", describes a step-by-step procedure to follow to power up the module and obtain the 135Bug prompt on the terminal screen.

For a question about syntax or operation of a particular 135Bug command, the user may turn to the entry for that particular command in the section describing the command set (Chapter 3).

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

NOTE: In the examples shown, all user input is given in bold script. This is done for clarity in understanding the examples (to distinguish between character input by the user and character output by 135Bug). The symbol **<CR>** represents the carriage return key on the user's terminal keyboard. Whenever this symbol appears it means that a carriage return should be entered by the user.

1.3 Installation and Start-Up

To enable 135Bug to operate properly with the MVME135 module, the following set-up procedure must be followed:

CAUTION

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

1. Refer to the MVME135 User's Manual (Motorola Publication Number MVME135) and configure the mini-jumpers on the module as required for the user's particular application. The only mini-jumper configuration which is specifically dictated by 135Bug is J7. Jumper J7 must be configured with a jumper pin across pins (2-3).

   NOTE: This jumper block configures the EPROM sockets at U54 (odd byte) and U56 (even byte) to accept 64K x 8 devices. This is the configuration of the MVME135 module as shipped from the factory.

2. Configure status switches S3 and S4 on the MVME135 as required for the user's particular application. Refer to Appendices E and F for configuration details. Configure the **BOOT** switch (S4-1Ø), the **MPSUP** switch (S4-9), and the **ENVØ**, **ENV1** switches (S4-3,4) to select the desired power-up/reset mode. These switches are described in detail in section 1.4.1.
3. Be sure that the two 135Bug EPROM's are installed in locations U54 and U56 of the MVME135 module.
4. Refer to the set-up procedure for the user's particular chassis or system for details concerning the installation of the MVME135.
5. Connect the terminal which is to be used as the 135Bug's system console to the connector labled **Ser Port 1** on the MVME135. Set up the terminal as follows:

   Step A - Eight bits per character.

   Step B - One stop bit per character.

   Step C - Parity disabled.

   Step D - Baud rate for the terminal connected to MVME135 port 1 must be set to 96ØØ. After power-up, the baud rates as well as other port characteristics may be changed via software using the debugger's **PF** (Port Format) command.

   NOTE: In order for high-baud rate serial communication between 135Bug 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/she should check the terminal to make sure XON/XOFF handshaking is enabled.

6. If it is desired to connect up some device (i.e., a host computer system or a serial printer) to port 2, connect the RS-232 cable for the device to the connector labled **Ser Port 2** on the MVME135. The characteristics for this port may be reconfigured later using 135Bug's **PF** command.
7. Power up the system. 135Bug will execute some self-checks and display the debugger prompt **135Bug>** . The messages displayed will vary depending on the system configuration and integrity. These messages are explained below:

a) A Confidence Test is unconditionally run at power up/reset. If the Confidence Test passes, a message is not displayed, but if any section of the test fails, the message, followed by a code indicating the failure mode is displayed as follows:

Confidence Test Failed, Code XX

Refer to Appendix G for an explanation of the Confidence Test failure codes. The code is also available in the MP Comm Byte of the MP-CSR. A non-zero value indicates a failure. The board FAIL light will remain lit, on Confidence Test failure.

b) If the MVME135 contains a MC68881 Floating Point Co-Processor, a FPC Confidence Test will be executed. If the FPC Confidence test executes without error, the message ....

FPC passed

.... is displayed, otherwise the message ....

FPC failed

.... is displayed. If a FPC is not detected, the following message is displayed.

No FPC detected

c) If the MVME135 contains a MC68851 Paged Memory Management Unit, a PMMU Confidence Test will be executed. If the PMMU Confidence test executes without error, the message ....

PMMU passed

.... is displayed, otherwise the message ....

PMMU failed

.... is displayed. If a PMMU is not detected, the following message is displayed.

No PMMU detected

d) Automatic sizing of local memory is performed to determine if the MVME135 contains a 1- or 4-Megabyte DRAM. Depending on the amount of local memory found, the following message will be displayed.

```
Local Memory size is 1 MEG (4 MEG)
```

e) If the local memory fails to respond correctly, the following message will be displayed.

```
Local Memory Failure
```

An example of the display from power up/reset for a healthy MVME135 containing a FPC, PMMU, and 1MEG of local DRAM follows:

```
VME135 Debugger/Diagnostics Release Version V.r - MM/DD/YY
FPC passed
PMMU passed
Local Memory size is 1 MEG
135Bug>
```

Messages pertaining to the AUTOBOOT function are explained in section 1.6 Autoboot.

1.4 MVME135 Board Operation With 135Bug

This section describes all switch and jumper settings used by 135Bug that directly affect its operation. The terminal port assignments are also defined in this section. All component initialization critical to the function of 135Bug is discussed in the last portion of this section.

1.4.1 MVME135 Switch Settings

The 135Bug will read the user switch settings and initialize the hardware control registers accordingly. Four switch positions are used to alter the operation of the 135Bug. These are the **BOOT**, **ENVØ**, **ENV1**, and the **MPSUP** switches.

1.4.1.1 BOOT Switch

DIP switch S4 on the MVME135 contains a mode control switch, **BOOT**, at switch position 1Ø. This switch controls the autoboot function of 135Bug. When the switch is in the **ON** position, the 135Bug is in manual boot mode. In manual boot mode, the debugger is entered after appropriate start-up (see next section), and the user is presented with a prompt. To bootstrap an operating system, the explicit boot commands (**BO** or **BH**) must be entered by the user. When this switch is in the **OFF** position, the 135Bug is in automatic boot mode. In automatic boot mode, the 135Bug attempts to boot from a pre-programmed location in the EPROM or to read a proper boot block from any devices connected in the system. The autoboot procedure is defined in detail in section 1.6.

In either BOOT mode, the user needs to be aware that all address information used by the Boot procedure, mainly the target program counter and stack pointer, is required to be accessible over the VMEbus. For example, when using OPTØ or OPT1, local memory addresses must be translated to the address accessible over VME.

NOTE: Translations are done unconditionally when using the 135Bug Disk I/O routines.

1.4.1.2 ENVØ and ENV1 Switches

DIP switch S4 on the MVME135 contains the mode control switches, **ENVØ** and **ENV1**, at switch positions 3 and 4, respectively. This switch selects one of four possible operating environments, or "options" of 135Bug. 135Bug sets up certain default conditions at power-up or restart based on these switches. In particular, the **ENVØ**, **ENV1** switches dictate the location of 135Bug's vector table and reserved workspace (see section 1.5, "Memory Requirements" for more details on 135Bug memory allocation).

The settings for this switch are shown in the following table. BASE refers to the first address of allocated RAM as seen by 135Bug.

TABLE 1-1. 135Bug ENVIRONMENT OPTIONS

OPT	ENVØ	ENV1	Restart Function
Ø	ON	ON	135Bug operates locally at BASE = Ø.
1	ON	OFF	135Bug operates locally at High memory, BASE = $FFXØØØØØ.
2	OFF	ON	135Bug operates over VMEbus BASE, BASE = Ø + OFFSET, with the OFFSET calculated by (BOARD n - 1)* 16K. (n = 1, 2, 3, etc.)
3	OFF	OFF	135Bug operates in first off-board VMEbus memory, with the OFFSET calculated by ID byte * 16K.

Refer to Appendices E and F, MVME135 Status Register STAT1 (S4) and the Board ID-Mapping switch (S3), for specifics on setting these switches to obtain the desired power-up/restart conditions.

1.4.1.3 MPSUP Switch

DIP switch S4 on the MVME135 contains a mode control switch, **MPSUP**, at switch position 9. This switch enables/disables the feature which allows a MVME135 or other CPU board to transfer control from the 135Bug currently executing to a previously specified location. When this mode is enabled, continuous polling occurs of several MP-CSR bits, as well as MPIRQ, resulting in a psuedo-interrupt under certain conditions into target code. This is explained in complete detail in section 1.7.

1.4.2 MVME135 Port Configurations

Some 135Bug commands give the user the option of choosing the port which will be used for input or output. The valid port numbers which may be used for these commands are:

Ø - MVME135 Terminal Port (MVME135 " Serial Port 1")

1 - MVME135 Host/Printer Port (MVME135 " Serial Port 2")

NOTE: These logical port numbers (Ø and 1) are referred to as "Serial Port 1" and "Serial Port 2", respectively, by the MVME135 hardware documentation.

For example, the command **DUØ** (Dump S-Records to Port Ø) would actually output data to the device connected to the serial port labeled "Ser Port 1" on the MVME135 front panel.

1.4.3 Z8Ø36 CIO Timer Registers

The Z8Ø36 CIO counter-timer device on-board the MVME135 module contains 48 internal registers. Some of these registers are used by the MVME135 hardware to maintain status and control information. 135Bug uses two 8-bit registers of the CIO to store the upper word of the address of its workspace memory. These registers would normally hold the interrupt vectors to be returned by the CIO (they are unused because the vectors are returned by a PROM on the MVME135 module). The value in these registers is used by 135Bug to locate its vector table, variables, and stack.

If the user elects to use these particular CIO registers containing the workspace start address then 135Bug will not operate. Using other CIO registers will impair MVME135 hardware operation.

If the user wishes to take advantage of the broadcast IRQ mechanism in the 135Bug (S4 position 9, MPSUP = OFF, as explained in section 1.7), then the CIO PORT B operation must remain configured as per the 135Bug setup. The Interrupt Pending bit in the Port B Control and Status register, is used for BRIRQ polling and should never be set when the bug is in operation unless a BRIRQ operation is desired.

The following table summarizes the CIO registers.

TABLE 1-2. RESERVED CIO REGISTERS

Address	CIO Register	MVME135/135Bug Function
$FFFBØØØ2	Port A IRQ Vector	Used by 135Bug to store upper byte of workspace memory start address.
$FFFBØØØ3	Port B IRQ Vector	Used by 135Bug to store upper-mid byte of workspace memory start address.
$FFFBØØØD	Port A Data	MVME135 Status Register STAT1.
$FFFBØØØE	Port B Data	MVME135 Control Register CNT1.

1.5 Memory Requirements

The program portion of 135Bug is approximately 128K bytes of code. In addition, 135Bug requires a minimum of 16K bytes of read/write memory to operate.

1.5.1 EPROM Mapping

The EPROM sockets on-board the MVME135 module are mapped at locations $FFFØØØØØ to $FFF1FFFF. The 135Bug code is position-independent and will execute anywhere in memory.

1.5.2 RAM Allocation

135Bug requires a minimum of 16K bytes of read/write memory to operate. This memory may either be an off-board system memory (i.e., on an external memory board such as the MVME2Ø4, MVME2Ø4-1 or MVME2Ø4-2) or 135Bug may utilize its own or another MVME135's on-board read/write memory.

On power-up or restart, 135Bug examines the setting of the **ENVØ**, **ENV1** switches of the MVME135's STAT1 status register (refer to Appendix E) to determine if the user desires to run out of MVME135 memory or from system memory.

Four environment options are available for selecting the location for the 135Bug's stack and work area. The MVME135 allows local memory to be accessed at either a high or low memory address, as well as over the VMEbus. Deciding which option to use depends on the system memory available, and whether the application requires the use of local memory to run more efficiently.

The four options as previously mentioned in section 1.4.1.2 are described below. BASE refers to the first address of allocated memory as seen by 135Bug.

In the first two options, 135Bug may see local DRAM at a different address range than seen by other VMEbus masters.

OPTION Ø: Locate the 135Bug locally at the low memory BASE address $Ø.

OPTION 1: Locate the 135Bug locally at the high memory address. With 1 MEG of local memory the BASE address is $FFEØØØØØ and for 4 MEG of local memory the BASE address is $FF8ØØØØØ.

The next two options allow the user to locate the 135Bug at one of two base addresses over the VMEbus. To allow multiple MVME135's to select the same option simultaneously, each of the 135Bug space must have a unique offset from the base address selected.

OPTION 2: Locate the 135Bug space at VMEbus address $Ø + offset. This mode assumes some type of memory mapped over VME at address $Ø, whether it is local or external memory. For Option 2, the offset is calculated by multiplying the VME135's board number - 1 (i.e., 1st, 2nd, 3rd ... from lowest to upper VMEbus address range; the lowest-mapped VME135 is board #1, the next lowest is board #2, etc.) by 16K. This is to enable multiple VME135's Bug stack and variable space to be continuous from address zero, even if the boards are not mapped contiguously over the VMEbus.

OPTION 3: Locate the 135Bug space at the first off-board system memory location plus the offset. The offset is calculated by multiplying the 5 least significant bits of the ID byte by 16K ((ID byte & $1F) * $4ØØØ). The ID byte is an image of the Board ID mapping switch S3. This mode assumes memory mapped contiguously following the MVME135's DRAM as it appears over the VMEbus.

NOTE: In order to accurately size past local memory using Option 3, all VME135's in the system must have the same local memory size.

Regardless of where the 16K bytes are located, the first 12K bytes are used for 135Bug stack and static variable space and the next 4K bytes are reserved as user space. Whenever the MVME135 is reset the target program counter is initialized to the address corresponding to the beginning of the user space and the target stack pointers are initialized to addresses within the user space, with the target ISP set to the top of the user space. The target VBR is set equal to the BASE plus the offset.

The following examples illustrate 135Bug memory allocation.

Example 1: Option Ø selected, with two 1 MEG MVME135's in the system, and a VME2Ø4-2 mapped at $2ØØØØØ over the VMEbus.

	ENVØ	ENV1	S3	135Bug Stack & Vars	Target PC	Target ISP
BD#1	ON	ON	ØØ	$ØØØØØØØØ-$ØØØØ2FFF	$ØØØØ3ØØØ	$ØØØØ4ØØØ
BD#2	ON	ON	Ø1	$ØØØØØØØØ-$ØØØØ2FFF	$ØØØØ3ØØØ	$ØØØØ4ØØØ

Example 2: Option 1 selected, with two 1 MEG MVME135's in the system, and a VME2Ø4-2 mapped at $2ØØØØØ over the VMEbus.

	ENVØ	ENV1	S3	135Bug Stack & Vars	Target PC	Target ISP
BD#1	ON	OFF	ØØ	$FFEØØØØØ-$FFEØ2FFF	$FFEØ3ØØØ	$FFEØ4ØØØ
BD#2	ON	OFF	Ø1	$FFEØØØØØ-$FFEØ2FFF	$FFEØ3ØØØ	$FFEØ4ØØØ

Example 3: Option 2 selected, with two 1 MEG MVME135's in the system, and a VME2Ø4-2 mapped at $2ØØØØØ over the VMEbus. In this example, both boards' bug space is allocated in BD#1's DRAM.

	ENVØ	ENV1	S3	135Bug Stack & Vars	Target PC	Target ISP
BD#1	OFF	ON	ØØ	$ØØØØØØØØ-$ØØØØ2FFF	$ØØØØ3ØØØ	$ØØØØ4ØØØ
BD#2	OFF	ON	Ø1	$ØØØØ4ØØØ-$ØØØØ7FFF	$ØØØØ7ØØØ	$ØØØØ8ØØØ

Example 4: Option 2 selected, with two 1 MEG MVME135's in the system, and a VME2Ø4-2 mapped at $Ø over the VMEbus. In this example, both boards' bug space is allocated on the VME2Ø4-2.

	ENVØ	ENV1	S3	135Bug Stack & Vars	Target PC	Target ISP
BD#1	OFF	ON	Ø2	$ØØØØØØØØ-$ØØØØ2FFF	$ØØØØ3ØØØ	$ØØØØ4ØØØ
BD#2	OFF	ON	Ø3	$ØØØØ4ØØØ-$ØØØØ6FFF	$ØØØØ7ØØØ	$ØØØØ8ØØØ

Example 5: Option 3 selected, with two 1 MEG MVME135's in the system, and a VME2Ø4-2 mapped at $2ØØØØØ over the VMEbus.

	ENVØ	ENV1	S3	135Bug Stack & Vars	Target PC	Target ISP
BD#1	OFF	OFF	ØØ	$ØØ2ØØØØØ-$ØØ2Ø2FFF	$ØØ2Ø3ØØØ	$ØØ2Ø4ØØØ
BD#2	OFF	OFF	Ø1	$ØØ2Ø4ØØØ-$ØØ2Ø6FFF	$ØØ2Ø7ØØØ	$ØØ2Ø8ØØØ

Example 6: Option 3 selected, with two 1 MEG MVME135's in the system, and a VME2Ø4-2 mapped at $Ø over the VMEbus.

	ENVØ	ENV1	S3	135Bug Stack & Vars	Target PC	Target ISP
BD#1	OFF	OFF	Ø2	$ØØØØ8ØØØ-$ØØØØAFFF	$ØØØØBØØØ	$ØØØØCØØØ
BD#2	OFF	OFF	Ø3	$ØØØØCØØØ-$ØØØØEFFF	$ØØØØFØØØ	$ØØØ1ØØØØ

Example 7: Option 3 selected, with two 4 MEG MVME135's in the system, and a VME2Ø4-2 mapped at $8ØØØØØ over the VMEbus.

	ENVØ	ENV1	S3	135Bug Stack & Vars	Target PC	Target ISP
BD#1	OFF	OFF	ØØ	$ØØ8ØØØØØ-$ØØ8Ø2FFF	$ØØ8Ø3ØØØ	$ØØ8Ø4ØØØ
BD#2	OFF	OFF	Ø1	$ØØ8Ø4ØØØ-$ØØ8Ø6FFF	$ØØ8Ø7ØØØ	$ØØ8Ø8ØØØ

1.6 AUTOBOOT

AUTOBOOT is a switch selectable function that provides an operator independent mechanism for booting from a pre-programmed location in the EPROM or an operating system. When enabled, **AUTOBOOT** will first determine if a preselected EPROM location is non-zero, and if so, control will be transfered to the address contained in that location. Otherwise, **AUTOBOOT** will scan for controllers and devices in a specified sequence until a valid bootable device is found or until the list is exhausted. If a valid bootable device is found, a boot from that device is started. The controller scanning sequence goes from the lowest controller Logical Unit Number (LUN) detected to the highest controller LUN detected. At the controller level, scanning goes from the lowest device LUN configured to the highest device LUN configured. Autoboot operation can be enabled or disabled with the **BOOT** switch as follows :

Switch **ON** = manual boot (Using **BO** or **BH** commands).

Switch **OFF** = auto boot or ROM boot (At power-on/reset).

Example 1: With the **BOOT** switch set to **ON**, the **RESET** pushbutton is pressed:

```
VME135 Debugger/Diagnostics Release Version V.r - MM/DD/YY
FPC  passed test
No PMMU detected
Local Memory Size is 1 MEG
135Bug>
```

Example 2: With the **BOOT** switch set to **OFF**, the **RESET** pushbutton is pressed. A ROM BOOT Stack Pointer and Program Counter have been preprogrammed in EPROM addresses $FFF1FFF4 and $FFF1FFF8, respectively. The following is displayed then control is transfered to the address in $FFF1FFF8.

```
VME135 Debugger/Diagnostics Release Version V.r - MM/DD/YY
FPC  passed test
No PMMU detected
Local Memory Size is 1 MEG
Booting from ROM address $XXXXXXXX
```

Example 3: With the **BOOT** switch set to **OFF**, the **RESET** pushbutton is pressed. EPROM location $FFF1FFF4 contains a zero value. The first bootable device is a streamer tape on the VME350, controller 4, device 0:

```
VME135 Debugger/Diagnostics Release Version V.r - MM/DD/YY
FPC  passed test
No PMMU detected
Local Memory Size is 1 MEG
Autoboot in progress... To abort hit <BREAK>
Booting from VME350 CLUN=4 DLUN=0
IPL loaded at: $00050000
```

NOTE: A vertical parity checksum word at $FFF1FFFE must be updated each time the 135Bug EPROMs are patched. The new checksum is calculated by performing the Boolean exclusive OR operation over the new contents for the EPROMs. A method for calculating the new checksum is described below.

1. Transfer the intended new contents for the EPROMs to system memory. One way is to download from development system, EPROM programmer, etc. into memory using 135Bug's LO command:

   ```
   135Bug>LO;x=COPY NEWPROMS> MX,# <cr>
   ```

   Another way is to copy the contents of the current EPROMs out into system memory with 135Bug's BM command and then make the desired changes. The following command sequence copies the EPROM code out to address $50000:

   ```
   135Bug>BM FFF00000:20000 50000;b
   Effective address: FFF00000
   Effective count  : &131072
   Effective address: 00050000
   135Bug>MM <addr to change> ; DI <cr>
   ```

2. Enter the following program segment at some location other than that containing the new EPROM contents. Running this program segment calculates the proper checksum for the new EPROM contents and leaves it in the lower word of register D1.

   ```
            LEA      <start addr of code>,A0  Point to new code.
            MOVE.L   #$1FFFE,D0               This is the byte count for
                                              loop.
            MOVEQ.L  #-1,D1                   Load initial checksum value.
   GETWORD  MOVE.W   (A0)+,D2                 Get a word.
            EOR.W    D2,D1                    Accumulate checksum.
            SUBQ.L   #2,D0
   ```

```
BNE.B     GETWORD
ANDI.L    #$ØØØØFFFF,D1          Mask off upper word.
SYSCALL   .RETURN                (D1.W contains checksum)
```

3. Run the program segment using 135Bug's GO command. Use 135Bug's RD command to view the checksum in the lower half of D1.

4. Install new checksum in last word of code.

5. Upload modified code to development system or EPROM programmer using 135Bug's DU command.

1.7 Multi-Processing Support (MPSUP)

There are four methods of transfering control to a target program from the 135Bug, in the Multi-Processing psuedo interrupt Support mode (MPSUP = OFF). Three bits in the MP-CSR are available for use, LMØ, SIGLP, and SIGHP, in addition to the MPIRQ bit in Control Register 1.

Since the 135Bug operates in non-interrupt mode, when the MPSUP mode is enabled, these bits will be polled regularly. When one of the four bits is asserted, it is processed as if an exception occurred, creating a normal four word stack frame, then jumping indirectly through the vector table. The polling operation is handled in the system console driver module.

Before setting any of the four bits, the location to which control will be transferred, must be loaded in the associated vector table address. The 135Bug's Interrupt Vector Base is $4ØØ offset from the target VBR value (Base + Offset). The vector table addresses for the four bits are as follows:

LMØ	135Bug VBR+$128	Location Monitor Ø
SIGLP	135Bug VBR+$12C	Low Priority Signal
SIGHP	135Bug VBR+$114	High Priority Signal
MPIRQ	135Bug VBR+$1Ø8	Broadcast IRQ

If the user plans to return to 135Bug using an "RTE" instruction after processing of the signal or broadcast has been completed, it is the user's responsibility to preserve the exception stack frame as well as 135Bug's register state.

Control can also be returned to the 135Bug by pressing the **ABORT** pushbutton.

Before 135Bug exits through the vector table to the pre-loaded target address, the bit causing the transfer of control will be negated. This is done to prevent an interrupt from occurring when the interrupt mask is lowered, and to prevent a re-transfer of control if 135Bug is re-entered.

The bits supported in the **MPSUP** mode, and how they operate is described below:

LMØ: This bit is low true, and can be set through the MP-CSR, or by a broadcast cycle to the associated location in Short I/O.

SIGLP(HP): This bit is high true, and is set by writting directly to the MP-CSR location.

MPIRQ: This bit is low true. In order to use this signal, the user must give up control of VME Interrupt Level 1, since the hardware uses this path for the BRIRQ cycle. When the **MPSUP** mode is selected, **VMSK1** is unconditionally enabled. Polling for the MPIRQ bit will not be done unless VMSK1 is enabled. Also, as previously mentioned, the Z8Ø36 CIO PORT B configuration must be programmed as it is in the 135Bug initialization.

Since the MPIRQ bit will be reset when BRIRQ0 goes away, polling will be done using the Z8Ø36 PORT B CSR Interrupt Pending bit.

1.8 Reference Documentation

The following publications may 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; telephone (6Ø2) 994-6561.

Document Title	Document Number
MVME135 Diagnostic Firmware User's Guide	MVME135DIAG
MVME135 32-Bit Multiprocessing Board User's Manual	MVME135
MVME2Ø4-1/-2 Dual Ported Dynamic Memory User's Manual	MVME2Ø4
VSB Device Specification	TBD
M68KVMMB851 Memory Management Board User's Manual	M68KVMMB851
MC68Ø2Ø 32-Bit Microprocessor User's Manual	MC68Ø2ØUM/AD
MC68851 Paged Memory Management Unit User's Manual	MC68851UM/AD
MC68881 Floating-Point Coprocessor User's Manual	MC68881UM/AD
MC68882 Enhanced Floating-Point Coprocessor Technical Summary	BR5Ø9/D
MVME319 Intelligent Disk/Tape Controller User's Manual	MVME319
MVME32Ø VMEbus Disk Controller Module User's Manual	MVME32Ø
MVME321 IPC Firmware User's Guide (Preliminary)	MVME321FW
MVME327 IPC Firmware User's Guide (Preliminary)	MVME327FW
MVME35Ø IPC Firmware User's Guide (Preliminary)	MVME35ØFW
MVME36Ø Storage Drive Disk Controller User's Manual	MVME36Ø

CHAPTER 2
USING THE 135Bug DEBUGGER

2.1 Entering Debugger Command Lines

135Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt **135Bug>** 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 section 2.2.

When a command is entered the debugger will execute the command and the prompt will reappear. However, if the command entered causes execution of user target code, (i.e., **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 will return to the debugger when the breakpoint is encountered during execution of the user program. Alternately, the user program could return control to the debugger by means of the TRAP #15 function **.RETURN** (described in Chapter 5). For more about this, refer to the description in Chapter 3 for the **GO** commands.

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

1. The command identifier (i.e., **MD** or **md** for the memory display command). Note that either upper- or lower-case is allowed.
2. A port number if the command is set up to work with more than one port.
3. At least one intervening space before the first argument.
4. Any required arguments, as specified by command.
5. 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 meta-symbols used are:

< > The angular brackets enclose a symbol, known as a syntactic variable, that is replaced in a command line by one of a class of symbols it represents.

[ ] Square brackets enclose a symbol that is optional.

| This symbol indicates that a choice is to be made. One of several symbols, separated by this symbol, should be selected.

/ The slash indicates that one or more of the symbols separated by this symbol can be selected.

{} These brackets enclose an optional symbol that may occur zero or more times.

2.1.1 Syntactic Variables

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

< DEL> - Delimiter; either a comma or a space.

< EXP> - Expression (described in detail in section 2.1.1.1).

< ADDR> - Address (described in detail in section 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 Expression as a Parameter

An expression can be one or more numeric values separated by the arithmetic operators plus (+) or 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.

Numeric value examples:

Base	Identifier	Examples
Hexadecimal	$	$FFFFFFFF
Decimal	&	&1974, &10-&4
Octal	@	@456
Binary	%	%1000110

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

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

String literal examples:

String Literal	Numeric Value (in Hex)
'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. Sub-expressions within parentheses are evaluated first. Nested parenthetical sub-expressions are evaluated from the inside out.

Examples of valid expressions are:

Expression	Result (in Hex)
FF0011	FF0011
45+99	DE
&45+&99	90
@35+@67+@10	5C
%10011110+%1001	A7
88<<44	880
AA&F0	A0

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 135Bug is similar to the one accepted by the 68020 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 summarized the address formats which are acceptable for address parameters in debugger command lines.

TABLE 2-1. DEBUGGER ADDRESS PARAMETER FORMATS

Format	Example	Description
N	14	Absolute address+contents of automatic offset register.
N+Rn	13Ø+R5	Absolute address+contents of the specified offset register (not an assembler-accepted syntax).
(An)	(A1)	Address register indirect.
(d,An) or d(An)	(12Ø,A1) 12Ø(A1)	Address register indirect with displacement (two formats accepted).
(d,An,Xn) or d(An,Xn)	(&12Ø,A1,D2) &12Ø(A1,D2)	Address register indirect with index and displacement (two formats accepted).
([bd,An,Xn],od)	([C,A2,A3],&1ØØ)	Memory indirect pre-indexed.
([bd,An],Xn,od)	([12,A3],D2,&1Ø)	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 (Ø to 7).
Rn	- Offset register n.

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

NOTE: Relative addresses are limited to 1 megabyte (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:

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

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

The above program was loaded at address 0001327C.

The disassembled code is shown next:

```
135Bug>MD 1327C;DI CR>
ØØØ1327C 48E78Ø8Ø                  MOVEM.L  DØ/AØ,-(A7)
ØØØ1328Ø 428Ø                      CLR.L    DØ
ØØØ13282 1Ø18                      MOVE.B   (AØ)+,DØ
ØØØ13284 534Ø                      SUBQ.W   #1,DØ
ØØØ13286 12D8                      MOVE.B   (AØ)+,(A1)+
ØØØ13288 51C8FFFC                  DBF      DØ,$13286
ØØØ1328C 4CDFØ1Ø1                  MOVEM.L  (A7)+,DØ/AØ
ØØØ1329Ø 4E75                      RTS
135Bug>
```

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

```
135Bug>OF RØ CR>
RØ =ØØØØØØØØ ØØØØØØØØ? 1327C:16. <CR>
135Bug>MD Ø+RØ;DI <CR>
ØØØØØ+RØ 48E78Ø8Ø                  MOVEM.L  DØ/AØ,-(A7)
ØØØØ4+RØ 428Ø                      CLR.L    DØ
ØØØØ6+RØ 1Ø18                      MOVE.B   (AØ)+,DØ
ØØØØ8+RØ 534Ø                      SUBQ.W   #1,DØ
ØØØØA+RØ 12D8                      MOVE.B   (AØ)+,(A1)+
ØØØØC+RØ 51C8FFFC                  DBF      DØ,$A+RØ
ØØØ1Ø+RØ 4CDFØ1Ø1                  MOVEM.L  (A7)+,DØ/AØ
ØØØ14+RØ 4E75                      RTS
135Bug>
```

2.2 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 the Control or **CTRL** key must be held down while striking the character key.

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

When observing output from any 135Bug 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 135Bug 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.

2.3 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 **MM** (Memory Modify) 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 135Bug Assembler/Disassembler.

Another way to enter a program is to download an object file from a host system (i.e., an EXORmacs). The program must be in S-Record format (described in Appendix A) and may have been assembled or compiled on the host system. Alternately, the program may have been previously created using the 135Bug **MM** command as outlined above and stored to the host using the **DU** (Dump) command. If a communication link exists between the host system and the VME135, then the file can be downloaded into memory via the debugger's **LO** command.

Another way is by reading in the program from disk, using one of the disk commands (i.e., **BO**, **BH**, or **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.

2.4 System Utility Calls 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 135Bug routines via the MC68Ø2Ø 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.

2.5 Restarting the System

There are three methods available to the user of initializing the system to a known state. Each has characteristics which make it more appropriate than another in certain situations.

2.5.1 Reset

Pressing and releasing the **RESET** pushbutton on the front panel of the VME135 will initiate an on-board reset. Two reset modes are available: COLD and WARM. By default, 135Bug is in COLD mode (refer to the **RESET** command description). During COLD reset, a total system initialization takes place, as if the VME135 module had just been powered up. All static variables are restored to their default states.

On-board serial ports are reconfigured to their default state. The breakpoint table is cleared. The offset registers are cleared. The target registers are invalidated. Input and output character queues are cleared. On-board devices (timer, serial ports, etc) are reset.

During WARM reset, 135Bug variables and tables are preserved, as well as the target state registers and breakpoints. If the particular VME135 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 VME135's illuminated **HALT** LED) for example, after a double bus fault, or if the 135Bug environment is ever lost (vector table is destroyed, etc).

2.5.2 Abort

Abort is invoked by pressing and releasing the **ABORT** pushbutton on the VME135 front panel. Whenever Abort is invoked while running target code, a "snapshot" of the processor state is captured and stored in the target registers. 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 will 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 pushbutton was pushed, will be displayed to the screen. Any breakpoints installed in the user code will be removed and the breakpoint table will remain intact. Control will be returned to the debugger.

2.5.3 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 the debugger console. Break will remove any breakpoints in the user code and will keep the breakpoint table intact. Break does not, however, take a snapshot of the machine state nor does it display the target registers. It is useful to terminate debugger commands that output large blocks of data before completion.

2.6 Preserving Debugger Operating Environment

This section explains how to avoid contaminating the operating environment of the debugger. 135Bug uses certain of the VME135's on-board resources and uses on-board memory to contain temporary variables, exception vectors, etc. If the user disturbs resources which 135Bug depends on, then the debugger may function unreliably or not at all.

2.6.1 135Bug Vector Table and Workspace

As described in section 1.5, "Memory Requirements", 135Bug needs 14.5K bytes of read/write memory to operate and also allocates another 1.5K bytes as user space for a total of 16K bytes allocated. On power-up or reset, 135Bug decides where this memory will be. Starting at this point, 135Bug reserves a 1Ø24-byte area for a user program vector table area and then allocates another 1Ø24-byte area and builds an exception vector table for the debugger itself to use. Next, 135Bug reserves space for static variables and initializes these static variables to predefined default values. After the static variables, 135Bug allocates space for the system stack, then initializes the system stack pointer to the top of this area.

With the exception of the first 1Ø24-byte vector table area, the user must be extremely careful not to use the above-mentioned memory areas for other purposes. The user should refer to section 1.5.2 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's program counter, causing a system crash.

2.6.2 Exception Vectors Used By 135Bug

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

TABLE 2-2. EXCEPTION VECTORS USED BY 135Bug

Vector Offset	Exception	135Bug Facility
$Ø8	Bus Error	Retries accesses when conflict bit active and RMC cycle caused error.
$1Ø	Illegal Instruction	Breakpoints (Used by GO, GN, GT)
$24	Trace	Trace operations (such as T)
$7C	Level 7 Interrupt	ABORT pushbutton
$BC	TRAP #15	System calls (See Chapter 5)

When the debugger handles one of the exceptions listed in Table 2-2, the target stack pointer is left pointing past the bottom of the exception stack frame created; that is, it reflects the system stack pointer values just before the exception occurred. In this way, the operation of the debugger facility (through an exception) is transparent to the user.

Example: Trace one instruction using debugger.

```
135Bug>RD <CR>
PC   =00003E00 SR   =2700=TR:OFF_S._7_.....
USP  =00003830 MSP  =00003C18 ISP* =00004000 VBR  =00000000
SFC  =0=F0     DFC  =0=F0     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
00003E00 203900100000          MOVE.L  ($100000).L,D0
135Bug>T <CR>
PC   =00003E06 SR   =2700=TR:OFF_S._7_.....
USP  =00003830 MSP  =00003C18 ISP* =00004000 VBR  =00000000
SFC  =0=F0     DFC  =0=F0     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
00003E06 D280                  ADD.L   D0,D1
135Bug>
```

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 135Bug or it may create a separate exception vector table of its own. The two following sections detail these two methods.

2.6.2.1 Using 135Bug's Target Vector Table

135Bug initializes and maintains a vector table area for target programs. A target program is any user program started by the bug, either manually with **GO** or **TRace** type commands or automatically with the **BOot** command. The start address of this target vector table area is the base address of the VME135 module, determined as described in section 1.5.2. This address is loaded into the target-state VBR at power-up and cold-start reset and can be observed by using the **RD** command to display the target-state registers immediately after power-up.

135Bug initializes the target vector table with the debugger vectors listed in Table 2-2 and fills the other vector locations with the address of a generalized exception handler (refer to section 2.6.2.3). 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 2-2 are overwritten then the accompanying debugger functions will be lost.

135Bug maintains a separate vector table for its own use in a 1K-byte 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.

2.6.2.2 Creating a New Vector Table

A user program may create a separate vector table in memory to contain its exception vectors. If this is done, 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 135Bug vector table into the corresponding vector locations in the user vector table.

The vector for the 135Bug generalized exception handler (described in detail in section 2.6.2.3) may be copied from offset $80 (Trap #0 vector) in the target vector table to all locations in the user's 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 the exception.

The following is an example of 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   $80(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
        BMI.S    LOOP                 Initialize entire vector table.
        MOVE.L   $8(A0),$8(A1)        Copy bus error vector.
        MOVE.L   $10(A0),$10(A1)      Copy breakpoints vector.
        MOVE.L   $24(A0),$24(A1)      Copy trace vector.
        MOVE.L   $BC(A0),$BC(A1)      Copy system call vector.
        LEA.L    COPROCC(PC),A2       Get user exception vector.
        MOVE.L   A2,$2C(A1)           Install as F-Line handler.
        MOVEC.L  A1,VBR               Change VBR to new table.
        RTS
        END
```

It may turn out that the user program uses one or more of the exception vectors that are required for debugger operation. Debugger facilities may still be used, however, if the user's 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 (i.e., **ABORT**) the user's 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 135Bug target program vector table (which the user program saved), yielding the address of the 135Bug exception vector. The user program then jumps to the address stored at this vector location (i.e., which is the address of the 135Bug 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 for the particular exception before jumping to the address of the exception handler.

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

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

2.6.2.3 135Bug Generalized Exception Handler

135Bug has a generalized exception handler which it uses to handle all of the exceptions not listed in Table 2-2. For all these exceptions, the target stack pointer is left pointing to the top of the exception stack frame created. In this way, if an unexpected exception occurs during execution of 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 $FØØØØØ. It is assumed for this example that an access of memory location $FØØØØØ will initiate Bus Error exception processing.

```
135Bug>RD <CR>
PC   =ØØØØ3EØØ SR   =27ØØ=TR:OFF_S._7_.....
USP  =ØØØØ383Ø MSP  =ØØØØ3C18 ISP* =ØØØØ4ØØØ VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØØØØØØ D1   =ØØØØØØØØ D2   =ØØØØØØØØ D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØ4ØØØ
ØØØØ3EØØ 2Ø39ØØFØØØØØ       MOVE.L  ($FØØØØØ).L,DØ
135Bug>T <CR>

Exception: Long Bus Error
Format/Vector=BØØ8
SSW=Ø145 Fault Addr. =ØØFØØØØØ Data In=ØØØØØØØØ Data Out=ØØØØ2ØØ6
PC   =ØØØØ3EØ6 SR   =A7ØØ=TR:ALL_S._7_.....
USP  =ØØØØ383Ø MSP  =ØØØØ3C18 ISP* =ØØØØ3FA4 VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØØØØØØ D1   =ØØØØØØØØ D2   =ØØØØØØØØ D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØ3FA4
ØØØØ3EØØ 2Ø39ØØFØØØØØ       MOVE.L  ($FØØØØØ).L,DØ
135Bug>
```

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.

```
135Bug>MD (A7):&44 <CR>
ØØØØ3FA4 A7ØØ ØØØØ 2ØØØ BØØ8  3E2C Ø145 ØØØØ ØØ27   '... .Ø.>,.E...'
ØØØØ3FB4 ØFØØ ØØØØ ØFØØ ØØØØ  ØØØØ 1BCC 2Ø39 ØØØØ   .p...p.....L 9..
ØØØØ3FC4 ØØØØ 2ØØA ØØØØ 2ØØ8  ØØØØ 2ØØ6 ØØØØ ØØØØ   .. ... ... .....
ØØØØ3FD4 ØØFØ ØØØØ 1ØØF Ø487  ØØØØ A7ØØ 4ØØ3 ØØØØ   .p........'.@...
ØØØØ3FE4 ØØØØ 7FFF ØØØØ ØØØØ  CØ1Ø ØØØØ ØØØØ 4ØØØ   ........@.....@.
ØØØØ3FF4 ØØØØ ØØØØ FFF8 Ø86C                         .....x.l
135Bug>
```

2.7 Disk I/O Support

135Bug can initiate disk Input/Output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 135Bug consist of command-level disk operations, disk I/O system calls (via the TRAP #15 instruction) for use by user programs, and automatic bootstrap at power-up or reset. 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 135Bug. Default values for these parameters are assigned at power-up and cold-start reset, but may be altered as described in section 2.7.5.

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

2.7.1 Blocks Versus Sectors

The logical block defines the unit of information for disk devices. A disk is viewed by 135Bug as a storage area divided in 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 will vary 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. 135Bug 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.

2.7.2 Disk I/O via 135Bug Commands

These following 135Bug commands are provided for disk I/O. Detailed instructions for their use may be found in Chapter 3. When a command is issued to a particular controller LUN and device LUN, these LUNs are remembered by 135Bug so that the next disk command will default to use the same controller and device.

2.7.2.1 IOP (Physical I/O to Disk)

This command 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.

2.7.2.2 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.

2.7.2.3 IOC (I/O Control)

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

2.7.2.4 BO (Bootstrap Operating System)

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

2.7.2.5 BH (Bootstrap and Halt)

BH reads an operating system or control program from the specified device into memory, and then returns control to 135Bug. It is used as a debugging tool.

2.7.3 Disk I/O via 135Bug System Calls

All operations that actually access the disk are done directly or indirectly by 135Bug 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 have been provided to allow user programs to do disk I/O:

.DSKRD	Disk Read. System call to read blocks from a disk into memory.
.DSKWR	Disk Write. System call to write blocks from memory onto a disk.
.DSKCFIG	Disk Configure. This function allows the user to change the configuration of the specified device.
.DSKFMT	Disk format. This function allows the user to send a format command to the specified device.
.DSKCTRL	Disk Control. This function is used to implement any special device control functions that can not be accomodated easily with any of the other disk functions.

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

To perform a disk operation, 135Bug must eventually present a particular disk controller module with a controller command packet which has been especially prepared for that type of controller 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 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.

2.7.4 Default 135Bug Controller and Device Parameters

135Bug initializes the parameter tables for a default configuration of controllers and devices (refer to Appendix C). If the system needs to be configured differently than this default configuration (for example, to use a 7Ø-Megabyte Winchester drive where the default is a 4Ø-Megabyte 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 (refer to Appendix B for more information on the disk's configuration area). This is a temporary change. If a cold-start reset occurs then the default parameter information will be written back into the tables.

Alternately, the **IOT** command may be used to manually reconfigure the parameter table for any controller and/or device that is different from the default. This is also a temporary change and will be overwritten if a cold-start reset occurs. Finally, the user may change the configuration files and rebuilt 135Bug so that it has different defaults. This last option is described in detail in the 135Bug Customer Letter. Refer to Appendix C for disk controller data.

2.7.5 Disk I/O Error Codes

135Bug returns an error code if an attempted disk operation is unsuccessful. Refer to Appendix D for an explanation of disk I/O error codes.

2.8 Additional Support Features

In addition to the features already discussed, the 135Bug supports other specialized functions of the VME135 module. These features are detailed in the following sections.

2.8.1 Function Code Support

The function codes identify the address space being accessed on any given bus cycle, and in general, they are an extension of the address. This becomes more obvious when using a memory management unit like the MC68851, where two identical logical addresses can be made to map to two different physical addresses. In this case, the function codes provide the additional information required to find the proper memory location.

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

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

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

<ADDR> ^<FC>

The valid function code mnemonics are:

Function Code	Mnemonic	Description
Ø	FØ	Unasigned, reserved
1	UD	User Data
2	UP	User Program
3	F3	Unassigned, reserved
4	F4	Unassigned, reserved
5	SD	Supervisor Data
6	SP	Supervisor Program
7	CS	CPU Space Cycle

Example: To change data at location $5ØØØ in the user data space.

135Bug>m **5ØØØ^ud** <CR>
ØØØØ5ØØØ^UD ØØØØ ? **1234.** <CR>
135Bug>

2.8.2 Diagnostic Facilities

As part of the 135Bug debugging package, the MVME135DIAG Diagnostic Firmware User's Guide provides a complete set of hardware diagnostics intended for the testing and troubleshooting of the VME135. 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 **SD** for "switch directories". The diagnostic prompt **135Bug>** should appear. Refer to the MVME135DIAG Diagnostic Firmware User's Guide 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 positioning of switches.

2.8.3 Floating Point Coprocessor Support

The **MC68881** (Floating Point Coprocessor) and the **MC68882** (Enhanced Floating Point Coprocessor) are supported in this version of 135Bug. An MC6888X confidence check is run at reset time to verify that the part is present and that all registers can be accessed. It also insures that a context switch can be done sucessfully. 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 MC6888X target state is saved and restored along with the processor state as required when switching between the target program and 135Bug.

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 timeout is received then the test is aborted and the message "No FPC detected" is displayed. Otherwise the test continues. If an error is detected the test is aborted and the message "FPC failed test" is displayed. If the test runs without errors then the message "FPC passed test" is displayed and an internal flag is set. This flag is later checked 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 operations 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:

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_21Ø3_123456789ABCDEFØ1	Packed Decimal Real Format
-3.123456789Ø12345Ø1_E+123	Scientific Notation Format (Decimal)

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

1. The sign field is the first field and is a binary field.
2. The exponent field is the second field and is a hexadecimal field.

3. The mantissa field is the last field and is a hexadecimal field.
4. 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).
5. Each field must be separated from adjacent fields by an underscore.
6. All the digit positions in the sign and exponent fields must be present.

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: The 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.

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:
- FPØ-FP7 - Floating-point Data Registers

For more information about the MC68881 coprocessor, refer to the MC68881 Floating Point Coprocessor User's Manual.

2.8.4 Paged Memory Management Unit Coprocessor Support

The Paged Memory Management Unit Coprocessor (MC68851) is supported in this version of 135Bug. An MC68851 confidence check is run at reset time to verify that the part is present and that all registers can be accessed. It also insures that a context switch can be done sucessfully. The commands **RD**, **RM**, **MD**, and **MM** have been extended to allow display and modification of **PMMU** data in registers and in memory. PMMU instructions can be assembled/disassembled with the DI option of the **MD/MM** commands. In addition, the MC68851 target state is saved and restored along with the processor state as required when switching between the target program and 135Bug. Finally, there is a set of diagnostics to test functionality of the PMMU.

At power-up/reset a PMMU confidence check is executed. Initially, a read of one of the PMMU registers is attempted. If a bus error timeout is received then the test is aborted and the message "No PMMU detected" is displayed. Otherwise the test continues. If an error is detected the test is aborted and the message "PMMU failed test" is displayed. If the test runs without errors then the message "PMMU passed test" is displayed and an internal flag is set. This flag is later checked by the bug when doing a task switch. The PMMU state will be saved and restored only if this flag is set. This allows proper bug operations in systems that do not have a PMMU.

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

The MC68851 registers are shown below, along with their data types in parentheses:

Address Translation Control Registers:

CRP	- CPU Root Pointer	(DL)
SRP	- Supervisor Root Pointer	(DL)
DRP	- DMA Root Pointer	(DL)
TC	- Translation Control	(L)

Status Information Registers:

PCSR	- PMMU Cache Status Register	(W)
PSR	- PMMU Status Register	(W)

Protection Mechanism Control Registers:

CAL	- Current Access Level	(B)
VAL	- Validate Access Level	(B)
SCC	- Stack Change Control	(B)
AC	- Access Control	(W)

Breakpoint Registers:

BADØ-BAD7	Breakpoint Acknowledge Data Registers	(W)
BACØ-BAC7	Breakpoint Acknowledge Control Registers	(W)

For more information about the MC68851 coprocessor, refer to the MC68851 Paged Memory Management Unit User's Manual.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3
THE 135Bug DEBUGGER COMMAND SET

3.1 Introduction

This chapter contains descriptions of each of the debugger commands and provides one or more examples of each. Table 3-1 summarizes the 135Bug debugger commands.

TABLE 3-1. DEBUGGER COMMANDS

Command Mnemonic	Title	Section
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
DC	Data Conversion	3.10
DU	Dump S-Records	3.11
GD	Go Direct (Ignore Breakpoints)	3.12
GN	Go to Next Instruction	3.13
GO	Go Execute User Program	3.14
GT	Go To Temporary Breakpoint	3.15
HE	Help	3.16
IOC	I/O Control for Disk	3.17
IOP	I/O Physical (Direct Disk Access)	3.18
IOT	I/O "TEACH" for Disk Configuration	3.19
LO	Load S-Records from Host	3.20
MA/NOMA	Macro Define/Display/Delete	3.21
MAE	Macro Edit	3.22
MAL/NOMAL	Enable/Disable Macro Expansion Listing	3.23
MAW/MAR	Save/Load Macros	3.24
MD	Memory Display	3.25
MM	Memory Modify	3.26
MS	Memory Set	3.27
OF	Offset Registers Display/Modify	3.28

TABLE 3-1. DEBUGGER COMMANDS (cont.)

Command Mnemonic	Title	Section
PA/NOPA	Printer Attach/Detach	3.29
PF	Port Format	3.30
RD	Register Display	3.31
RESET	Cold/Warm Reset	3.32
RM	Register Modify	3.33
SD	Switch Directories	3.34
T	Trace	3.35
TC	Trace On Change of Control Flow	3.36
TM	Transparent Mode	3.37
TT	Trace To Temporary Breakpoint	3.38
VE	Verify S-Records Against Memory	3.39

Each command is described in the following text. The command's syntax is shown using the symbols explained in section 2.1. In the examples shown, all user input is shown in bold font. This is done for clarity in understanding the examples (i.e., to distinguish between character input by the user and character output by 135Bug). The symbol **<CR>** represents the carriage return key on the user's terminal keyboard. Whenever this symbol appears it means that a carriage return should be entered by the user.

3.2 Block of Memory Fill **BF**

BF < RANGE> < DEL> < data> [< increment>] [; B|W|L]

where:

< data> and < increment> are both expression parameters

options:

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 will be 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 will be printed stating the increment which was actually used.

It 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 will ever be disturbed in any case. The "Effective address" messages displayed by the command will show exactly where data was stored.

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

```
135Bug>BF 20000,2001F 4E71 <CR>
Effective address: 00020000
Effective address: 0002001F
135Bug>MD 20000:18 <CR>
00020000 4E71 4E71 4E71 4E71  4E71 4E71 4E71 4E71    NqNqNqNqNqNqNqNq
00020010 4E71 4E71 4E71 4E71  4E71 4E71 4E71 4E71    NqNqNqNqNqNqNqNq
00020020 0000 0000 0000 0000  0000 0000 0000 0000    ................
135Bug>
```

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

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

```
135Bug>BF 20000,10 4E71 ;B <CR>
Effective address: 00020000
Effective count  : &16
Data = $71
135Bug>MD 20000:30;B <CR>
00020000 7171 7171 7117 7171  7171 7171 7171 7171    qqqqqqqqqqqqqqqq
00020010 0000 0000 0000 0000  0000 0000 0000 0000    ................
00020020 0000 0000 0000 0000  0000 0000 0000 0000    ................
135Bug>
```

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

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

```
135Bug>BF 20000,20006 12345678 ; L <CR>
Effective address: 00020000
Effective address: 00020003
135Bug>MD 20000:C;L <CR>
00020000 1234 5678 0000 0000  0000 0000 0000 0000    .4Vx............
00020010 0000 0000 0000 0000  0000 0000 0000 0000    ................
00020020 0000 0000 0000 0000  0000 0000 0000 0000    ................
135Bug>
```

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 to $2002F is clear).

```
135Bug>BF 20000,18 0 1 <CR>            ( default size is Word       )
Effective address: 00020000
Effective count  : &24
135Bug>MD 20000:18 <CR>
00020000 0000 0001 0002 0003  0004 0005 0006 0007    ................
00020010 0008 0009 000A 000B  000C 000D 000E 000F    ................
00020020 0010 0011 0012 0013  0014 0015 0016 0017    ................
135Bug>
```

3.3 Bootstrap Operating System and Halt BH

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

Device LUN	- Is the logical unit number of the device to boot from. Defaults to LUN Ø.
Controller LUN	- Is the logical unit number of the controller to which the above device is attached. Defaults to LUN Ø.
< 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.

The **BH** command 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. Since control is retained by 135Bug, all the 135Bug facilities are available for debugging the loaded program if necessary.

Examples:

```
135Bug>BH 1,Ø <CR>                ( Boot and halt from device LUN 1, )
135Bug>                           ( controller Ø.                     )

135Bug>BH A,3,test2;d <CR>        ( Boot and halt from device LUN $A,)
135Bug>                           ( controller 3, 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.

3.4 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 four with an option of L would mean to move four longwords (or 16 bytes) to the new location. If an option field is specified without a count in the range an error results.

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

```
135Bug>MD 21000:10 <CR>
00021000 5448 4953 2049 5320  4120 5445 5354 2121    THIS IS A TEST!!
00021010 0000 0000 0000 0000  0000 0000 0000 0000    ................
135Bug>
```

```
135Bug>BM 21000 2100F 20000 <CR>
Effective address: 00021000
Effective address: 0002100F
Effective address: 00020000
135Bug>
```

```
135Bug>MD 20000:10 <CR>
00020000 5448 4953 2049 5320  4120 5445 5354 2121    THIS IS A TEST!!
00020010 0000 0000 0000 0000  0000 0000 0000 0000    ................
135Bug>
```

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...

```
135Bug>MD 20000:5;DI <CR>
00020000 D480                  ADD.L     D0,D2
00020002 E2A2                  ASR.L     D1,D2
00020004 2602                  MOVE.L    D2,D3
00020006 4E4F0021              SYSCALL   .OUTSTR
0002000A 4E71                  NOP
135Bug>
```

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.

```
135Bug>BM 20002 2000B 20004 <CR>
Effective address: 00020002
Effective address: 0002000B
Effective address: 00020004
135Bug>
```

```
135Bug>MD 20000:6;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 4E4F0021              SYSCALL   .OUTSTR
0002000C 4E71                  NOP
135Bug>
```

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

```
135Bug>MM 20002;DI <CR>
00020002 E2A2              ASR.L    D1,D2 ? NOP <CR>
00020002 4E71              NOP
00020004 E2A2              ASR.L    D1,D2 ? . <CR>
135Bug>
```

```
135Bug>MD 20000:6;DI <CR>
00020000 D480              ADD.L    D0,D2
00020002 4E71              NOP
00020004 E2A2              ASR.L    D1,D2
00020006 2602              MOVE.L   D2,D3
00020008 4E4F0021          SYSCALL  .OUTSTR
0002000C 4E71              NOP
135Bug>
```

3.5 Bootstrap Operating System BO

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

Device LUN - Is the logical unit number of the device to boot from. Defaults to LUN Ø.

Controller LUN - Is the logical unit number of the controller to which the above device is attached. Defaults to LUN Ø.

< 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.

The **BO** command 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 Ø of the device LUN specified. The device and controller configurations used when **BO** is initiated can be examined and changed via the **IOT** command.

The following sequence of events occur when **BO** is invoked:

1. Block Ø of the device LUN and controller LUN specified is read into memory.

2. Locations $F8(248) to $FF(255) of block Ø are checked to contain the string "MOTOROLA" or "EXORMACS".

3. The following information is extracted from block Ø:

   $9Ø(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 Ø contain the necessary information to initiate this transfer:

   $14(2Ø)-$17(23) : Block number of first sector to load from disk.

   $18(24)-$19(25) : Number of blocks to load from disk.

   $1E(3Ø)-$21(33) : Starting memory location to load.

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

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

   4-7: Initial value for target program counter. 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 = $27ØØ.

   DØ = Device LUN.

   D1 = Controller LUN.

   D4 = 'IPLx', with x=$ØC ($495Ø4CØC).

   The ASCII string 'IPL' indicates that this is the Initial Program Load sequence; the code $ØC indicates TRAP #15 support with stack parameter passing and TRAP #15 disk support.

   AØ = 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:

```
135Bug>BO <CR>                  ( Boot from device LUN Ø,            )
                                ( controller Ø.                      )

135Bug>BO 3 <CR>                ( Boot from device LUN 3,            )
                                ( controller 3.                      )

135Bug>BO , 3 <CR>              ( Boot from device LUN Ø,            )
                                ( controller 3.                      )

135Bug>BO 8 Ø,test <CR>         ( Boot from device LUN 8,            )
                                ( controller Ø, and pass the string  )
                                ( "test" to the booted program.      )
```

3.6 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 Ø count is found, the target code state is saved in the target registers and control is returned back to 135Bug. 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** are 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 135Bug.

NOBR is used for deleting breakpoints from the breakpoint table. If an address is specified then that address will be removed from the breakpoint table. If **NOBR** < **CR**> is entered then all entries will be deleted from the breakpoint table and the empty table will be displayed.

Example:

```
135Bug>BR 14ØØØ,142ØØ 147ØØ:&12 <CR>     ( Set some breakpoints.       )
BREAKPOINTS
ØØØ14ØØØ          ØØØ142ØØ
ØØØ147ØØ:C

135Bug>NOBR 142ØØ <CR>                  ( Delete one breakpoint.      )
BREAKPOINTS
ØØØ14ØØØ          ØØØ147ØØ:C

135Bug>NOBR <CR>                        ( Delete all breakpoints.     )
BREAKPOINTS
135Bug>
```

3.7 Block of Memory Search BS

BS < RANGE> < DEL> < TEXT> [;B|W|L] or

BS < RANGE> < DEL> < data> < DEL> [< mask>] [;B|W|L,N,V]

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

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

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

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 one's is used (all bits will be 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 or addresses and data will be done only when the memory contents do NOT match the user-specified pattern. Otherwise this mode is identical to Mode 2.

For all three modes, informations 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 of 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 will be output and a message will be output stating that the last match does not lie entirely within the range. The user may search non-contiguous memory with this command without causing a Bus Error.

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

```
00030004 0000 0045 7272 6F72  2053 7461 7475 733D    ...Error Status=
00030010 3446 2F2F 436F 6E66  6967 5461 626C 6563    4F//ConfigTableS
00030020 7461 7274 3A00 0000  0000 0000 0000 0000    tart:..........
```

```
135Bug>BS 30000 3002F 'Task Status' <CR>
Effective address: 00030000                Mode 1: the string is not
Effective address: 0003002F                found, so a message is
-not found-                                output.
135Bug>
```

```
135Bug>BS 30000 3002F 'Error Status' <CR>
Effective address: 00030000                Mode 1: the string is found,
Effective address: 0003002F                and the address of its first
00030003                                   byte is output.
135Bug>
```

```
135Bug>BS 30000 3001F 'ConfigTableStart' <CR>
Effective address: 00030000
Effective address: 0003001F
00030014
-last match extends over range boundary-
135Bug>
```

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.

```
135Bug>BS 30000:30 't' ; B <CR>
Effective address: 00030000
Effective count  : &48
0003000A   0003000C   00030020   00030023
135Bug>
```

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

```
135Bug>BS 30000:18,2F2F <CR>
Effective address: 00030000
Effective count  : &24
00030012|2F2F
135Bug>
```

Mode 2, using <RANGE> with count: count is displayed in decimal, and the data pattern is found and displayed.

```
135Bug>BS 30000,3002F 3D34 <CR>
Effective address: 00030000
Effective address: 0003002F
-not found-
135Bug>
```

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

```
135Bug>BS 30000,3002F 3D34 ;N <CR>
Effective address: 00030000
Effective address: 0003002F
0003000F|3D34
135Bug>
```

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

```
135Bug>BS 3ØØØØ:3Ø 6Ø,FØ ;B <CR>
Effective address: ØØØ3ØØØØ
Effective count  : &48
ØØØ3ØØØ6|6F    ØØØ3ØØØB|61    ØØØ3ØØ15|6F    ØØØ3ØØ16|6E
ØØØ3ØØ17|66    ØØØ3ØØ18|69    ØØØ3ØØ19|67    ØØØ3ØØ1B|61
ØØØ3ØØ1C|62    ØØØ3ØØ1D|6C    ØØØ3ØØ1E|65    ØØØ3ØØ21|61
135Bug>
```

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

3.8 Block of Memory Verify BV

BV < RANGE> < DEL> < data> [< increment>][;B|W|L]

where:

< data> and < increment> are both expression parameters

options:

B - Byte

W - Word

L - Longword

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

If the user-entered data or increment (if specified) does not fit into the data field size then leading bits are truncated to make the fit. If truncation occurs, then a message will be 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 verified, then data is verified to the last boundary before the upper address. No address outside of the specified range will be read from in any case. The "Effective address" messages displayed by the command will show exactly the extent of the area read from.

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

```
135Bug>MD 20000:18 <CR>
00020000 4E71 4E71 4E71 4E71  4E71 4E71 4E71 4E71    NqNqNqNqNqNqNqNq
00020010 4E71 4E71 4E71 4E71  4E71 4E71 4E71 4E71    NqNqNqNqNqNqNqNq
00020020 4E71 4E71 4E71 4E71  4E71 4E71 4E71 4E71    NqNqNqNqNqNqNqNq
135Bug>BV 20000 2001F 4E71 <CR>        (default size is Word        )
Effective address: 00020000
Effective address: 0002001F
135Bug>                                (verify successful, nothing printed)
```

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

```
135Bug>MD 20000:30;B <CR>
00020000 0000 0000 0000 0000  0000 0000 0000 0000    ................
00020010 0000 0000 0000 0000  0000 0000 0000 0000    ................
00020020 0000 0000 0000 0000  0000 4AFB 4AFB 4AFB    ..........J.J.J.
135Bug>BV 20000:30,0;B <CR>
Effective address: 00020000
Effective count  : &48
0002002A|4A   0002002B|FB   0002002C|4A   0002002D|FB
0002002E|4A   0002002F|FB
135Bug>                                (mismatches are printed out     )
```

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

```
135Bug>MD 20000:18 <CR>
00020000 0000 0001 0002 0003  0004 0005 0006 0007    ................
00020010 0008 FFFF 000A 000B  000C 000D 000E 000F    ................
00020020 0010 0011 0012 0013  0014 0015 0016 0017    ................
135Bug>BV 20000:18,0,1 <CR>            (default size is Word        )
Effective address: 00020000
Effective count  : &24
00020012|FFFF
135Bug>                                (mismatch is printed out)
```

3.9 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 MC68Ø2Ø addressing mode.

Examples:

```
135Bug>DC 1Ø <CR>
        ØØØØØØ1Ø = $1Ø = &16
135Bug>

135Bug>DC &1Ø-&2Ø <CR>
SIGNED  : FFFFFFF6 = -$A = -&1Ø
UNSIGNED: FFFFFFF6 = $FFFFFFF6 = &4294967286
135Bug>

135Bug>DC 123+&345+@67+%11ØØØØ1 <CR>
        ØØØØØ314 = $314 = &788
135Bug>

135Bug>DC (2*3*8) /4 <CR>
        ØØØØØØØC = $C = &12
135Bug>

135Bug>DC 55&F <CR>
        ØØØØØØØ5 = $5 = &5
135Bug>
```

135Bug> **DC 55> > 1** <CR>

ØØØØØØ2A = $2A = &42

135Bug>

The subsequent examples assume AØ=ØØØ3ØØØØ and the following data resides in memory:

ØØØ3ØØØØ 11111111 22222222 33333333 44444444 ..."""3333DDDD

135Bug>**DC (AØ)** <CR>
ØØØØ3ØØØ = $3ØØØØ = &1966Ø8
135Bug>

135Bug>**DC ([,AØ])** <CR>
11111111 = $11111111 = &286331153
135Bug>

135Bug>**DC (4,AØ)** <CR>
ØØØ3ØØØ4 = $3ØØØ4 = &196612
135Bug>

135Bug>**DC ([4,AØ])** <CR>
22222222 = $22222222 = &5726623Ø6
135Bug>

3.10 Dump S-Records DU

DU [<port>]<DEL><RANGE><DEL>[<TEXT><DEL>][<ADDR>][<OFFSET>][;B|W|L]

The **DU** command outputs data from memory in the form of Motorola S-Records to a port specified by the user. If port is not specified then the S-Records are sent to the 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's 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 A 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.

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

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

```
135Bug>DU 20000 2002F <CR>
Effective address: 00020000
Effective address: 0002002F
135Bug>
```

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

```
135Bug>DU0 30000:&10 <CR>
Effective address: 00020000
Effective count  : &10
S0030000FC
S20E03000026025445535466084E4F7B
S9030000FC
135Bug>
```

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.

```
135Bug>DU 20000 2002F 'TEST' 2000A <CR>
Effective address: 00020000
Effective address: 0002002F
135Bug>
```

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 will create with the VERSAdos utility **UPLOADS**.

```
135Bug>TM <CR>                ( Go into transparent mode to establish    )
Escape character: $01=^A      ( communication with the EXORmaces.        )

<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 will at this point display 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 the escape character to return )
                            ( to 135Bug.                          )
```

Now enter the command for 135Bug to dump the S-Records to the port.

```
135Bug>DU 20000 2000F 'FILE1' <CR>
Effective address: 00020000
Effective address: 0002000F
135Bug>

135Bug>TM <CR>              ( Go into transparent mode again.     )
Escape character: $10=^A

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

The **UPLOADS** utility will now display 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 )
135Bug>                        ( to 135Bug.                           )
```

3.11 Go Direct (Ignore Breakpoints) GD

GD [< ADDR>]

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

Once execution of target code has begun, control may be returned to 135Bug by various conditions:

1. The user presses the **ABORT** or **RESET** pushbuttons on the VME135 front panel.
2. An unexpected exception occurs.
3. By execution of the **.RETURN** TRAP #15 function.

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

```
135Bug>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
135Bug>RM D0 <CR>
```

Initialize D0 and start target program:

```
D0    =00000000 ? 52A9C. <CR>
135Bug>GD 10000 <CR>
Effective address: 00010000
```

To exit target code, press **ABORT** pushbutton.

```
Exception: Abort
Format Vector = Ø1ØØ
PC   =ØØØ1ØØØE SR   =2711=TR:OFF_S._7_X...C
USP  =ØØØØF83Ø MSP  =ØØØØFC18 ISP* =ØØØØFFF8 VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØ52A9C D1   =ØØØØØØØØ D2   =ØØØØØØFF D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØFFF8
ØØØ1ØØØE                    BRA.B   $1ØØØE
135Bug>
```

Set PC to start of program and restart target code:

```
135Bug>RM PC <CR>
PC   =ØØØ1ØØØE ? 1ØØØØ. <CR>
135Bug>GD <CR>
Effective address: ØØØ1ØØØØ
```

3.12 Go To Next Instruction GN

GN

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

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 $6ØØØØ.

```
135Bug>MD 6ØØØØ:4;DI <CR>
ØØØØ6ØØØ 7ØØ3               MOVE.L  #3,DØ
ØØØØ6ØØ2 72Ø1               MOVEQ.L #1,D1
ØØØØ6ØØ4 61ØØFFA            BSR.W   $7ØØØ
ØØØØ6ØØ8 26ØØ               MOVE.L  DØ,D3
135Bug>
```

The following simple subroutine resides at address $7ØØØ.

```
135Bug>MD 7ØØØØ:2;DI <CR>
ØØØØ7ØØØ DØ81               ADD.L   D1,DØ
ØØØØ7ØØ2 4E75               RTS
135Bug>
```

Execute up to the BSR instruction.

```
135Bug>RM PC <CR>
PC    =00000000 ? 6000. <CR>
135Bug>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=F0      DFC   =0=F0      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
135Bug>
```

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

```
135Bug>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=F0      DFC   =0=F0      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
135Bug>
```

3.13 Go Execute User Program GO

GO [< ADDR>]

The **GO** command (alias **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 135Bug by various conditions:

1. A breakpoint with a count of zero is found.
2. The user presses the **ABORT** or **RESET** pushbuttons on the VME135 front panel.
3. An unexpected exception occurs.
4. By execution of the **.RETURN** TRAP #15 function.

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

```
135Bug>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
135Bug>
```

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

```
135Bug>RM DØ <CR>
DØ   =ØØØØØØØØ ? 52A9C. <CR>
135Bug>BR 1ØØØØ,1ØØØE <CR>
BREAKPOINTS
ØØØ1ØØØØ            ØØØ1ØØØE
135Bug>GO 1ØØØØ <CR>
Effective address: ØØØ1ØØØØ
At Breakpoint
PC   =ØØØ1ØØØE SR   =2Ø11=TR:OFF_S._Ø_X...C
USP  =ØØØØF83Ø MSP  =ØØØØFC18 ISP* =ØØØ1ØØØØ VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØ52A9C D1   =ØØØØØØØØ D2   =ØØØØØØFF D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØ1ØØØØ
ØØØ1ØØØE 6ØFE              BRA.B   $1ØØØE
135Bug>
```

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

Continue target program execution.

```
135Bug>G <CR>
Effective address: ØØØ1ØØØE
At Breakpoint
PC   =ØØØ1ØØØE SR   =2Ø11=TR:OFF_S._Ø_X...C
USP  =ØØØØF83Ø MSP  =ØØØØFC18 ISP* =ØØØ1ØØØØ VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØ52A9C D1   =ØØØØØØØØ D2   =ØØØØØØFF D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØ1ØØØØ
ØØØ1ØØØE 6ØFE              BRA.B   $1ØØØE
135Bug>
```

Remove breakpoints and restart target code.

```
135Bug>NOBR <CR>
BREAKPOINTS
135Bug>GO 1ØØØØ <CR>
Effective address: ØØØ1ØØØØ
```

To exit target code, press the **ABORT** pushbutton.

```
Exception: Abort
Format Vector = Ø1ØØ
PC   =ØØØ1ØØØE SR   =2Ø11=TR:OFF_S._Ø_X...C
USP  =ØØØØF83Ø MSP  =ØØØØFC18 ISP* =ØØØ1ØØØØ VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØ52A9C D1   =ØØØØØØØØ D2   =ØØØØØØFF D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØFFF8
ØØØ1ØØØE 6ØFE                BRA.B   $1ØØØE
135Bug>
```

3.14 Go To Temporary Breakpoint GT

GT < ADDR>

The **GT** command allows the user to set a temporary breakpoint and then start target code execution. A count may be specified with the temporary breakpoint. Control is given at the target PC address. All previously set breakpoints are enabled. The temporary breakpoint is removed when any breakpoint with Ø 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 135Bug by various conditions:

1. A breakpoint with a count of zero is found.
2. The user presses the **ABORT** or **RESET** pushbuttons on the VME135 front panel.
3. An unexpected exception occurs.
4. By execution of the **.RETURN** TRAP #15 function.

Example: (The following program resides at $1ØØØØ).

```
135Bug>MD 1ØØØØ;DI <CR>
ØØØ1ØØØØ 22ØØ                    MOVE.L  DØ,D1
ØØØ1ØØØ2 4282                    CLR.L   D2
ØØØ1ØØØ4 D4Ø1                    ADD.B   D1,D2
ØØØ1ØØØ6 E289                    LSR.L   #1,D1
ØØØ1ØØØ8 66FA                    BNE.B   $1ØØØ4
ØØØ1ØØØA E2ØA                    LSR.B   #1,D2
ØØØ1ØØØC 55C2                    SCS     D2
ØØØ1ØØØE 6ØFE                    BRA.B   $1ØØØE
135Bug>
```

Initialize DØ and set a breakpoint:

```
135Bug>RM DØ <CR>
DØ   =ØØØØØØØØ ? 52A9C. <CR>
135Bug>BR 1ØØØE <CR>
BREAKPOINTS
ØØØ1ØØØE
135Bug>
```

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

```
135Bug>RM PC <CR>
PC    =ØØØ1ØØØE ? 1ØØØØ. <CR>
135Bug>
```

```
135Bug>GT 1ØØØ6 <CR>
Effective address: ØØØ1ØØØ6
Effective address: ØØØ1ØØØØ
At Breakpoint
PC    =ØØØ1ØØØ6 SR    =2711=TR:OFF_S._7_X...C
USP   =ØØØØ383Ø MSP   =ØØØØ3C18 ISP* =ØØØØ4ØØØ VBR   =ØØØØØØØØ
SFC   =Ø=FØ     DFC   =Ø=FØ     CACR =Ø=..     CAAR  =ØØØØØØØØ
DØ    =ØØØ52A9C D1    =ØØØØØØ29 D2   =ØØØØØØØ9 D3    =ØØØØØØØØ
D4    =ØØØØØØØØ D5    =ØØØØØØØØ D6   =ØØØØØØØØ D7    =ØØØØØØØØ
AØ    =ØØØØØØØØ A1    =ØØØØØØØØ A2   =ØØØØØØØØ A3    =ØØØØØØØØ
A4    =ØØØØØØØØ A5    =ØØØØØØØØ A6   =ØØØØØØØØ A7    =ØØØØ4ØØØ
ØØØ1ØØØ6 E289                   LSR.L   #1,D1
135Bug>
```

Set another temporary breakpoint at $1ØØØ2 and continue the target program execution.

```
135Bug>GT 1ØØØ2 <CR>
Effective address: ØØØ1ØØØ6
At Breakpoint
PC    =ØØØ1ØØØ6 SR    =2711=TR:OFF_S._7_X...C
USP   =ØØØØ383Ø MSP   =ØØØØ3C18 ISP* =ØØØØ4ØØØ VBR   =ØØØØØØØØ
SFC   =Ø=FØ     DFC   =Ø=FØ     CACR =Ø=..     CAAR  =ØØØØØØØØ
DØ    =ØØØ52A9C D1    =ØØØØØØØØ D2   =ØØØØØØFF D3    =ØØØØØØØØ
D4    =ØØØØØØØØ D5    =ØØØØØØØØ D6   =ØØØØØØØØ D7    =ØØØØØØØØ
AØ    =ØØØØØØØØ A1    =ØØØØØØØØ A2   =ØØØØØØØØ A3    =ØØØØØØØØ
A4    =ØØØØØØØØ A5    =ØØØØØØØØ A6   =ØØØØØØØØ A7    =ØØØØ4ØØØ
ØØØ1ØØØE 6ØFE                   BRA.B   $1ØØØE
135Bug>
```

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

3.15 Help HE

HE [< COMMAND>]

HE is the 135Bug help facility. **HE < CR>** displays the command name of all available commands along with its appropriate title. **HE < COMMAND>** displays only the command name and title for that particular command.

Examples:

```
135Bug>HE <CR>
BF      Block Fill
BH      Boot Operating System and Halt
BI      Block Initialize
BM      Block Move
BO      Boot Operating System
BR      Breakpoint Insert
NOBR    Breakpoint Delete
BS      Block Search
BV      Block Verify
DC      Data Conversion and Expression Evaluation
DU      Dump S-Records
GD      Go Direct (no breakpoints)
GN      Go and Stop after Next Instruction
GO      Go to Target Code
G       " Alias" for previous command
GT      Go and Insert Temporary Breakpoint
HE      Help facility
IOC     I/O Control for Disk
IOP     I/O to Disk
IOT     I/O " Teach"
LO      Load S-Records
MA      Macro Define/Display
NOMA    Macro Delete
MAE     Macro Edit
MAL     Enable Macro Expansion Listing
NOMAL   Disable Macro Expansion Listing
MAW     Save Macros
MAR     Load Macros
```

MD	Memory Display
MM	Memory Modify
MS	Memory Set
OF	Offset Registers
PA	Printer Attach
NOPA	Printer Detach
PF	Port Format
RD	Register Display
RESET	Cold/Warm Reset
RM	Register Modify
SD	Switch Directory
T	Trace Instruction
TC	Trace on Change of Flow
TM	Transparent Mode
TT	Trace to Temporary Breakpoint
VE	Verify S-Records

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

```
135Bug>HE T <CR>
T       Trace Instruction
135Bug>
```

3.16 I/O Control For Disk **IOC**

IOC

The **IOC** command allows a user to send command packets directly to a disk controller. The packet to be sent must already reside in memory and must follow the packet format of the particular disk controller.

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.

The default controller LUN and device LUN when **IOC** is invoked are those most recently specified for **IOP**, **IOT**, or a previous invocation of **IOC**. The same special characters used by the **MM** command to access a previous field (^), reopen the same location (=), or exit (.), can be used with **IOC**. The power-up default for the packet address is the area which is also used by the **BO** and **IOP** commands for building packets.

Example: Send the packet at $1ØØØØ to a VME319 controller board configured as CLUN #Ø. Specify an operation to the hard disk which is at DLUN #1.

```
135Bug>IOC <CR>
Controller LUN    =ØØ? <CR>
Device LUN        =ØØ? 1 <CR>
Packet address    =ØØØØ12BC? 1ØØØØ <CR>
ØØØ1ØØØØ Ø219 15ØØ 1ØØ1 ØØØ2  Ø1ØØ 3DØØ 3ØØØ ØØØØ   ..........=.Ø...
ØØØ1ØØ1Ø ØØØØ ØØØØ Ø3ØØ ØØØØ  ØØØØ Ø2ØØ Ø3         ................
Send Packet=Y (Y/N)? <CR>
135Bug>
```

3.17 I/O Physical to Disk IOP

IOP

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

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

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

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

1. Controller LUN =ØØ?

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

2. Device LUN =ØØ?

   The Logical Unit Number of the device to access is specified in this field.

3. Read/Write/Format =R?

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

   a. R for Read. This will read blocks of data from the selected device into memory.

   b. W for Write. This will write blocks of data from memory to the selected device.

c. F for Format. This will format the selected device. For disk devices, either a track or the whole disk can be selected by a subsequent field.

4. Memory Address =00003000?

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

5. Starting Block =00000000?

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

6. Number of Blocks =0002?

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

7. Address Modifier =00?

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

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

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

9. File Number =0000?

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

10. Flag Byte =00?

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

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

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

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

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

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

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

Erase: This will completely clear the tape of previous data and at the same time will retension the tape.

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

Example 1: Read 25 blocks starting at block 37Ø from device 2 of controller Ø into memory beginning at address $5ØØØØ.

```
135Bug>IOP <CR>
Controller LUN    =ØØ? <CR>
Device LUN        =ØØ? 2 <CR>
Read/Write/Format=R? <CR>
Memory Address    =ØØØØ3ØØØ? 5ØØØØ <CR>
Starting Block    =ØØØØØØØØ? &37Ø <CR>
Number of Blocks =ØØØ2? &25 <CR>
Address Modifier =ØØ? <CR>
135Bug>
```

Example 2: Write 14 blocks starting at memory location $7ØØØ to file 6 of device Ø, controller 4. Append a filemark at the end of the file.

```
135Bug>IOP <CR>
Controller LUN    =ØØ? 4 <CR>
Device LUN        =Ø2? Ø <CR>
Read/Write/Format=R? W <CR>
Memory Address    =ØØØ5ØØØØ? 7ØØØ <CR>
File Number       =ØØØØØ172? 6 <CR>
Number of Blocks =ØØ19? e <CR>
Flag Byte         =ØØ? %Ø1 <CR>
Address Modifier =ØØ? <CR>
135Bug>
```

3.18 I/O Teach Disk Configuration IOT

IOT [;[H][A]]

The **IOT** command allows the user to "teach" a new disk configuration to 135Bug 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 since 135Bug commands that access the disk use the TRAP #15 disk functions, changes in the descriptor tables will 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 the **H** (Help) option specified. This option instructs **IOT** to list the disk controllers which are currently available in the system.

Example:

```
135Bug>IOT;H <CR>
   Disk Controllers Available
Lun   Type     Address    # dev
 Ø   VME32Ø  $FFFFBØØØ     4
 4   VME35Ø  $FFFF5ØØØ     1
135Bug>
```

IOT may be invoked with the **A** (All) option specified. This option instructs **IOT** to list all the disk controllers which are currently supported in 135Bug. When invoked without options, the **IOT** command enters an interactive sub-command mode where the descriptor table values currently in effect are displayed one-at-a-time on the console for the operator to examine. The operator may change the displayed value by entering a new value or may leave it unchanged by typing only a carriage return. The same special characters used by the **MM** command to access a previous field (^), reopen the same location (=), or exit (.), can be used with **IOT**. All numerical values are interpreted as hexadecimal numbers. Decimal values may be entered by preceding the number with an "&".

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

If the Controller LUN and Device LUN selected do not correspond to a valid controller and device, then IOT will output the message "Invalid LUN" and the user will be 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 will begin displaying the values in the attribute fields, allowing the user to enter changes if desired.

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

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

1. Sector Size:

   ```
   Ø-128  1-256
   2-512  3-1Ø24       =Ø1?
   ```

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

2. Block Size:

   ```
   Ø-128  1-256
   2-512  3-1Ø24       =Ø1?
   ```

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

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

3. Sectors/Track =ØØ2Ø?

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

4. Starting Head =10?

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

5. Number of Heads =05?

   This field specifies the number of heads on the drive.

6. Number of Cylinders =0337?

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

   48 TPI - 40 Cylinders
   96 TPI - 80 Cylinders

7. Precomp. Cylinder =0000?

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

8. Reduced Write Current Cylinder =0000?

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

9. Interleave Factor =00?

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

10. Spiral Offset =00?

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

11. ECC Data Burst Length=ØØ?

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

12. Step Rate Code =ØØ?

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

The encoding is as follows:

Step Rate Code(Hex)	Winchester Hard Disks	5-1/4" Floppy	8" Floppy
ØØ	Ø msec	12 msec	6 msec
Ø1	6 msec	6 msec	3 msec
Ø2	1Ø msec	12 msec	6 msec
Ø3	15 msec	2Ø msec	1Ø msec
Ø4	2Ø msec	3Ø msec	15 msec

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

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

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

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

48 TPI - Single Track Density
96 TPI - Double Track Density

15. Single/Equal_in_all Track zero density =S (S/D)?

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

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

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

S = 25ØkHz data rate
F = 5ØØkHz data rate

17. Gap 1 =Ø7?

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

18. Gap 2 =Ø8?

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

19. Gap 3 =ØØ?

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

2Ø. Gap 4 =ØØ?

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

21. Spare Sectors Count =ØØ?

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

22. Reserved Area Units:Tracks/Cylinders =T (T/C)?

This field specifies the units (tracks or cylinders) used for the next two fields.

23. <UNITS> Reserved for Alternates=ØØØØ?

This field specifies the number of <UNITS> reserved for the alternate mapping area on the disk. The token <UNITS> is replaced by the word "Tracks" or the word "Cylinders", as specified by the "Reserved Area Units" field.

24. < UNITS> Reserved for Controller=ØØØØ?

This field specifies the number of < UNITS> reserved for use by the controller. The token < UNITS> is replaced by the word "Tracks" or the word "Cylinders", as specified by the "Reserved Area Units" field.

Example 1: Examining the default parameters of a 5-1/4" Floppy Disk.

```
135Bug>IOT <CR>
Controller LUN        = ØØ? <CR>
Device LUN            = ØØ? 2 <CR>
Sector Size:
Ø-128 1-256
2-512 3-1Ø24          = Ø1? <CR>
Block Size:
Ø-128 1-256
2-512 3-1Ø24          = Ø1? <CR>
Sectors/Track         = ØØ1Ø? <CR>
Number of Heads       = Ø2? <CR>
Number of Cylinders   = ØØ5Ø? <CR>
Precomp. Cylinder     = ØØ28? <CR>
Step Rate Code        = ØØ? <CR>
Single/Double DATA Density  =D (S/D)? <CR>
Single/Double TRACK Density =D (S/D)? <CR>
135Bug>
```

Example 2: Changing from a 4Ø Megabyte Winchester to a 7Ø Megabyte Winchester. (Note that reconfiguration such as this is only necessary when a user wishes to read or write a disk which is different than the default using the **IOP** command. Reconfiguration is normally done automatically by the **BO** or **BH** command when booting from a disk which is different from the default).

```
135Bug>IOT <CR>
Controller LUN          = ØØ? <CR>
Device LUN              = ØØ? 2 <CR>
Sector Size:
Ø-128 1-256
2-512 3-1Ø24            = Ø1? <CR>
Block Size:
Ø-128 1-256
2-512 3-1Ø24            = Ø1? <CR>
Sectors/Track           = ØØ2Ø? <CR>
Starting Head           = ØØ? <CR>
Number of Heads         = Ø6? 8 <CR>
Number of Cylinders     = Ø33E? 4ØØ <CR>
Precomp. Cylinder       = ØØØØ? 4Ø1 <CR>
Reduced Write Current Cylinder= ØØØØ? <CR>
Interleave Factor       = Ø1?  ØB <CR>
Spiral Offset           = ØØ? <CR>
ECC Data Burst Length= ØØØØ? ØØØB <CR>
135Bug>
```

Example 3: Changing from Fuji 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.

```
135Bug>IOT <CR>                                   (Fixed Disk)
Controller LUN        = ØØ? 2 <CR>
Device LUN            = ØØ? <CR>
Sector Size:
Ø-128 1-256
2-512 3-1Ø24          = Ø2? 1 <CR>
Block Size:
Ø-128 1-256
2-512 3-1Ø24          = Ø1? <CR>
Sectors/Track         = ØØ4Ø? <CR>
Starting Head         = ØØ? 1Ø <CR>
Number of Heads       = ØA? 5 <CR>
Number of Cylinders   = Ø337? <CR>
Interleave Factor     = Ø1? <CR>
Spiral Offset         = ØØ? <CR>
Gap 1                 = 1Ø? 7 <CR>
Gap 2                 = 2Ø? 8 <CR>
Spare Sectors Count   = ØØ? <CR>
135Bug>
```

```
135Bug>IOT <CR>                            (Removable Disk)
Controller LUN        = Ø2? <CR>
Device LUN            = ØØ? 1 <CR>
Sector Size:
Ø-128 1-256
2-512 3-1Ø24          = Ø1? <CR>
Block Size:
Ø-128 1-256
2-512 3-1Ø24          = Ø1? <CR>
Sectors/Track         = ØØ4Ø? <CR>
Starting Head         = ØØ? <CR>
Number of Heads       = ØØ? 1 <CR>
Number of Cylinders   = Ø337? <CR>
Interleave Factor     = Ø1? <CR>
Spiral Offset         = ØØ? <CR>
Gap 1                 = 7? <CR>
Gap 2                 = 8? <CR>
Spare Sectors Count   = ØØ? <CR>
135Bug>
```

3.19 Load S-Records From Host LO

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 VME135 module. The **LO** command accepts serial data from the host and loads it into memory.

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 will be 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 will cause the records to be stored to memory at different locations then 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 $Ø 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 (=), will be sent to the host before 135Bug 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. The 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 will also be echoed back to the host port by the host and will appear 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** will keep looking for a <LF> character from the host, signifying the end of the echoed command. No data records will be processed until this <LF> is received. If the host system does not echo characters, **LO** will still keep 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 will be processed.

The other options have the following effects:

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

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

T option - TRAP #15 code. This option causes LO to set the target register D4 = 'LO'x, with x = $ØC ($4C4F2ØØC). The ASCII string 'LO ' indicates that this is the LO command; the code $ØC 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, since some Motorola debuggers use conventions different from 135Bug, and they will set a different code in D4.

The S-Record format (refer to Appendix A) 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) will be 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 will be printed to the screen and 135Bug's error handler will be invoked to point to the faulty character.

As mentioned, if the embedded checksum of a record does not agree with the checksum calculated by 135Bug AND if the checksum comparison has not been disabled via the "-C" option then an error condition exists. A message will be 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 is 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 will cause the command to abort.

Since 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:

```
                          * Test Program.
                          *
          65040000                  ORG       $65040000

65040000 7001                       MOVEQ.L   #1,D0
65040002 D088                       ADD.L     A0,D0
65040004 4A00                       TST.B     D0
65040006 4E75                       RTS
                                    END
****** TOTAL ERRORS      0--
****** TOTAL WARNINGS    0--
```

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

```
S00F000054455354533353372020010 15E
S30D650400007001D0884A004E75B3
S70565040000091
```

Load this file into the VME135's memory for execution at address $4ØØØØ as follows:

```
135Bug>TM <CR>                  ( Go into transparent mode to establish  )
Escape character: $Ø1=^A        ( communication with the VME/1Ø.         )

<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    )
                                ( 135Bug prompt.                         )
```

```
135Bug>LO -65ØØØØØØ ;X=COPY TEST.MX,# <CR>
COPY TEST.MX,#
SØØFØØØØ5445535453335337202ØØ1Ø15E
S3ØD65Ø4ØØØØ7ØØ1DØ884AØØ4E75B3
S7Ø565Ø4ØØØØ91
135Bug>
```

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

The offset address of -65ØØØØØØ was added to the addresses of the records in FILE.MX and caused the program to be loaded to memory starting at $4ØØØØ. The text "COPY TEST.MX,#" is a VERSAdos command line that caused the file to be copied by VERSAdos to the VME/1Ø port which is connected with the VME135's host port.

```
135Bug>MD 4ØØØØ:4;DI <CR>
ØØØ41ØØØ 7ØØ1                 MOVEQ.L #1,DØ
ØØØ4ØØØ2 DØ88                 ADD.L   AØ,DØ
ØØØ4ØØØ4 4AØØ                 TST.B   DØ
ØØØ4ØØØ6 4E75                 RTS
135Bug>
```

The target PC now contains the entry point of the code in memory ($4ØØØØ).

3.2Ø Macro Define/Display/Delete **MA** **NOMA**

MA [<name>]
NOMA [<name>]

<name> : any combination of 1-8 alphanumeric characters

The **MA** command allows the user to define a complex command consisting of any number of 135Bug primitive commands with optional parameter specifications.

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

Entering **MA** without specifying a macro name causes 135Bug to list all currently defined macros and their definitions.

When **MA** is invoked with the name of a currently defined macro, that macro's definition will be displayed. Line numbers are shown when displaying macro definitions to facilitate editing via **MAE** (see section 3.21).

If **MA** is invoked with a valid macro name that does not currently have a definition, then 135Bug will enter the macro definition mode. In response to each macro definition prompt "M=", enter a 135Bug command, including a carriage return. Commands entered are not checked for syntax until the macro is invoked. To exit the macro definition mode, enter only a carriage return (null line) in response to the prompt. If the macro contains errors, it can either be deleted and redefined or it can be edited with the **MAE** command. A macro containing no primitive 135Bug commands (i.e., no definition) will not be accepted.

Macro definitions are stored in a string pool of fixed size. If the string pool becomes full while in the definition mode, the offending string will be discarded, a message "STRING POOL FULL, LAST LINE DISCARDED" will be printed and the user will be returned to the 135Bug command prompt. This will also happen if the string entered would cause the string pool to overflow. The string pool has a capacity of 255 characters. The only way to add or expand macros when the string pool is full is to either edit or delete macro(s).

135Bug commands contained in macros may reference arguments supplied at invocation time. Arguments are denoted in macro definitions by embedding a back slash character "\" followed by a numeral. Up to 1Ø arguments are permitted. A definition containing a back slash followed by a zero would cause the first argument to that macro to be inserted in place of the "\Ø" characters.

The second argument would be used wherever the sequence "\1" occurred. Entering "ARGUE 3ØØØ 1 ;B" on the debugger command line would invoke the macro named "ARGUE" with the text strings "3ØØØ", "1", and ";B" replacing "\Ø", "\1", and "\2" (respectively) within the body of the macro.

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

Examples:

```
135Bug>MA ABC <CR>
M=MD 3ØØØ <CR>
M=GO \Ø <CR>
M= <CR>
135Bug>
```

Define macro ABC

```
135Bug>MA DIS <CR>
M=MD \Ø:17;DI <CR>
M= <CR>
135Bug>
```

Define macro DIS

```
135Bug>MA <CR>
MACRO ABC
Ø1Ø MD 3ØØØ
Ø2Ø GO \Ø
MACRO DIS
Ø1Ø MD \Ø:17;DI
135Bug>
```

List macro definitions

```
135Bug>MA ABC <CR>
MACRO ABC
Ø1Ø MD 3ØØØ
Ø2Ø GO \Ø
135Bug>
```

List definition of macro ABC

```
135Bug>NOMA DIS <CR>
135Bug>
```

Delete macro DIS

```
135Bug>MA ASM <CR>                     Define macro ASM
M=MM \Ø;DI <CR>
M=<CR>
135Bug>

135Bug>MA <CR>                         List all macros
MACRO ABC
Ø1Ø MD 3ØØØ
Ø2Ø GO \Ø
MACRO ASM
Ø1Ø MD \Ø;DI
135Bug>

135Bug>NOMA <CR>                       Delete all macros
135Bug>

135Bug>MA <CR>                         List all macros
NO MACROS DEFINED
135Bug>
```

3.21 Macro Edit **MAE**

MAE <name> <line #> [<string>]

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

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

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

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

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

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

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

Examples:

```
135Bug>MA ABC <CR>          List definition of macro ABC
MACRO ABC
Ø1Ø MD 3ØØØ
Ø2Ø GO \Ø
135Bug>
```

```
135Bug>MAE ABC 15 RD <CR>             Add a line to macro ABC
MACRO ABC
Ø1Ø MD 3ØØØ
Ø2Ø RD                                This line was inserted
Ø3Ø GO \Ø
135Bug>

135Bug>MAE ABC 1Ø MD 1Ø+RØ <CR>       Replace line 1Ø
MACRO ABC
Ø1Ø MD 1Ø+RØ                          This line was overwritten
Ø2Ø RD
Ø3Ø GO \Ø
135Bug>

135Bug>MAE ABC 3Ø <CR>                Delete line 3Ø
MACRO ABC
Ø1Ø MD 1Ø+RØ
Ø2Ø RD
135Bug>
```

3.22 Enable/Disable Macro Expansion Listing MAL NOMAL

MAL
NOMAL

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

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

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

3.23 Save/Load Macros **MAW** **MAR**

MAW [<Device LUN>][<DEL>[<Controller LUN>][<DEL><Block #>]]
MAR [<Device LUN>][<DEL>[<Controller LUN>][<DEL><Block #>]]

Device LUN - Is the logical unit number of the device to save/load macros to/from. Initially defaults to LUN Ø.

Controller LUN - Is the logical unit number of the controller to which the above device is attached. Initially defaults to LUN Ø.

DEL - Is a field delimiter: Comma (,) or spaces ( ).

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

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

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

NOTE: **MAR** will discard all currently defined macros before loading from disk/tape.

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

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

Example: (Assume that device 2, controller Ø is accessable).

```
135Bug>MAR 2,Ø,3 <CR>                  Load macros from block 3
135Bug>

135Bug>MA <CR>                         List macros
MACRO ABC
Ø1Ø MD 3ØØØ
Ø2Ø GO \Ø
135Bug>

135Bug>MA ASM <CR>                     Define macro ASM
M=MM \Ø;DI <CR>
M= <CR>
135Bug>

135Bug>MA <CR>                         List all macros
MACRO ABC
Ø1Ø MD 3ØØØ
Ø2Ø GO \Ø
MACRO ASM
Ø1Ø MD \Ø;DI
135Bug>

135Bug>MAW ,,8 <CR>                    Save macros to block 8, previous
                                       device
WRITING TO BLOCK $8 ON CONTROLLER $Ø, DEVICE $2
OK to proceed (y/n)? Y                 Carriage return not needed
135Bug>
```

3.24 Memory Display **MD**

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

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 one line MC68020 assembler/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 will cause the command to re-execute, displaying an equal number of data items or lines beginning at the next address.

Example 1:

```
135Bug>MD 12000 <CR>
00012000 2800 1942 2900 1942  2800 1842 2900 2846    (..B)..B(..B).(F
135Bug><CR>
00012010 FC20 0050 ED07 9F61  FF00 000A E860 F060    |..Pm..a....h'p'
135Bug>
```

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

```
135Bug>MD (A2,D5):&19;B <CR>
00013627 4F82 00C5 9B10 337A  DF01 6C3D 4B50 0F0F    0..E..3z_.1=KP..
00013637 31AB 80                                     +1.
135Bug>
```

Example 3: To display memory at location 50008 with disassembly enabled, the user enters the following.

```
135Bug>MD 50008;DI <CR>
00050008 46FC2700                   MOVE.W   $2700,SR
0005000C 61FF0000023E               BSR.L    #5024C
00050012 4E7AD801                   MOVEC.L  VBR,A5
00050016 41ED7FFC                   LEA.L    $7FFC(A5),A0
0005001A 5888                       ADDQ.L   #4,A0
0005001C 2E48                       MOVE.L   A0,A7
0005001E 2C48                       MOVE.L   A0,A6
00050020 13C7FFFB003A               MOVE.B   D7,($FFFB003A).L
135Bug>
```

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

```
135Bug>MD 50008;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 0_3FF_0000000000000= 1.0000000000000000_E+0000
00005028 0_000_00000FFFFFFFF= 2.1219957904712067_E+0314
00005030 0_44D_FDE9F10A8D361= 6.0200000000000000_E+0023
00005038 0_3C0_79CA10C924223= 1.5999999999999999_E+0019
135Bug>
```

3.25 Memory Modify **MM**

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

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 (alias **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 will be closed and the next memory location will be 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 will be 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** will back up and open the previous memory location.

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

. - Terminates **MM** command. Control will return to 135Bug.

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

Example 1:

```
135Bug>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
135Bug>
```

Example 2:

```
135Bug>MM 10001;LA <CR>                 Longword access to location 10001
00010001 CD432187? <CR>                 (Alternate location accesses)
00010009 00068010? 68010+10= <CR>       Modify and re-open location
00010009 00068020? <CR>
00010009 00068020? . <CR>               Exit MM
135Bug>
```

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 will be disassembled and displayed and the user will be prompted with a question mark ("?") for input. At this point the user has three options:

1. Enter <CR>. This will close the present location and will continue with disassembly of next instruction.

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

3. Enter . <CR>. This will close the present location and will exit the **MM** command.

If a new source line is entered (#2 above), the present line will be 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 **PF** command. In the hardcopy mode, a line feed will be done instead of erasing the line.

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

Refer to Chapter 4 for additional information about the assembler.

The examples below were made in the hardcopy mode.

Example 3:

Assemble a new source line.

```
135Bug>MM 1000C;DI <CR>
0001000C 46FC2400          MOVE.W  $2400,SR ? DIVS.W -(A2),D2 <CR>
0001000C 85E2              DIVS.W  -(A2),D2
0001000E 2400              MOVE.L  D0,D2 ?
135Bug> .
```

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 ?
135Bug>
```

Example 5:

Step to next location and exit MM.

```
135Bug>M 1000C;DI <CR>
0001000C 000000FF          OR.B    #255,D0 ? <CR>
00010010 20C9              MOVE.L  A1,(A0)+ ? . <CR>
135Bug>
```

Example 6:

```
135Bug>M 7ØØØ;X <CR>
ØØØØ7ØØØ Ø_ØØØØ_FFFFFFFFØØØØØØØØ? 1_3C1Ø_84782 <CR>
ØØØØ7ØØC 1_7FFF_ØØØØØØØØFFFFFFFF? Ø_ØØ1A_F <CR>
ØØØØ7Ø18 Ø_ØØØØ_FFFFFFFFØØØØØØØØ? 6.Ø2E23= <CR>
ØØØØ7Ø18 Ø_4Ø4D_FEF4F885469BØ88Ø? ^ <CR>
ØØØØ7ØØC Ø_ØØ1A_FØØØØØØØØØØØØØØØ? <CR>
ØØØØ7ØØØ 1_3C1Ø_8478ØØØØØØØØØØØØ? . <CR>
135Bug>
```

3.26 Memory Set MS

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

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

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

Example: Assume that memory is initially cleared:

```
135Bug>MS 25000 0123456789abcDEF 'This is ''a test'' 23456 <CR>
135Bug>MD 25000:20;B <CR>
00025000 0123 4567 89AB CDEF  5468 6973 2069 7320    .#Eg.+MoThis is
00025010 2761 2074 6573 7427  2345 6000 0000 0000    'a test'#E'....
135Bug>
```

3.27 Offset Registers Display/Modify OF

OF [Rn[;A]]

The **OF** command 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 offset registers in section 2.1.1.2.2).

There are 8 offset registers (RØ through R7), but only RØ through R6 can be changed. R7 always has both base and top addresses set to Ø. This allows the automatic register function to be effectively disabled by selecting 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 1-megabyte 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 of the **RM** (Register Modify) and **MM** (Memory Modify) 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 Ø. This effectively disables them.
3. R7 always has both base and top addrèsses set to Ø, it cannot be changed.
4. Any offset register can be set as the automatic register.
5. The automatic register is always added to every absolute address argument of every 135Bug command where there is not an offset register explicitly called out.
6. There is always an automatic register. Note that a convenient way to disable the effect of the automatic register is by setting R7 as the automatic register. This is the default condition.

Examples:

Display of offset registers.

```
135Bug>OF <CR>
RØ = ØØØØØØØØ ØØØØØØØØ  R1 = ØØØØØØØØ ØØØØØØØØ
R2 = ØØØØØØØØ ØØØØØØØØ  R3 = ØØØØØØØØ ØØØØØØØØ
R4 = ØØØØØØØØ ØØØØØØØØ  R5 = ØØØØØØØØ ØØØØØØØØ
R6 = ØØØØØØØØ ØØØØØØØØ  R7*= ØØØØØØØØ ØØØØØØØØ
135Bug>
```

Modify some offset registers.

```
135Bug>OF RØ <CR>
RØ = ØØØØØØØØ ØØØØØØØØ? 2ØØØØ 2ØØFF <CR>
R1 = ØØØØØØØØ ØØØØØØØØ? 25ØØØ:2ØØ^ <CR>
RØ = ØØØ2ØØØØ ØØØ2ØØFF? . <CR>
135Bug>
```

Look at location $20000.

```
135Bug>M 20000;DI <CR>
00000+R0 41F95445 5354          LEA.L   ($54455354).L,A0 . <CR>
135Bug>M R0;DI <CR>
00000+R0 41F95445 5354          LEA.L   ($54455354).L,A0 . <CR>
135Bug>
```

Set R0 as the automatic register.

```
135Bug>OF R0;A <CR>
R0*=00020000 000200FF? . <CR>
135Bug>
```

To look at location $20000.

```
135Bug>M 0;DI <CR>
00000+R0 41F95445 5354          LEA.L   ($54455354).L,A0 . <CR>
135Bug>
```

To look at location 0, override the automatic offset.

```
135Bug>M 0+R7;DI <CR>
00000000 FFF8                   DC.W    $FFF8 . <CR>
135Bug>
```

3.28 Printer Attach/Detach **PA** **NOPA**

PA [n]

NOPA [n]

These two commands "attach" or "detach" a printer to the specified port. When the printer is attached, everything that appears on the system console terminal is also echoed to the "attached" port's printer. **PA** is used to attach, **NOPA** is used to detach. If no port is specified, **PA** will attach port 1 by default, **NOPA** will detach all attached ports.

If the port number specified is not currently assigned, **PA** will display an error message. If **NOPA** is attempted on a port that is not currently attached, an error message will be displayed.

The port being attached must already be configured. This is done using the **PF** (Port Format) command. On the VME135, it is necessary to disable the hardware handshake mechanism. This is done by executing the following sequence prior to "PA1".

```
135Bug>PF1 <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? <CR>
Async Mono, Bisync, Gen, SDLC, or HDLC [A,M,B,G,S,H] = A? <CR>
Sync1 = $00? <CR>
Sync2 = $00? <CR>
DTE or DCE [T,C] = C? <CR>
Auto Xmit enable on CTS* [Y,N] = Y? N. <CR>
135Bug>
```

RECOVERING FROM A "HUNG" PRINTER: attached ports are not detached by exceptions (bus errors, abort, etc). If printer attach is invoked to an incorrectly set-up device, or a fault such as a paper jam occurs, the only means of recovery is the **RESET** switch on the VME135 module.

Examples:

```
CONSOLE DISPLAY:                     PRINTER OUTPUT:
135Bug>PA <CR>
(attaching port 1 by default         (printer now attached)

135Bug>HE NOPA <CR>                  135Bug>HE NOPA
NOPA      Printer detach             NOPA      Printer detach

135Bug>NOPA <CR>                     135Bug>NOPA
(detach all attached printers)       (printer now detached)
135Bug>
```

3.29 Port Format PF

PF[n]

The **PF** command 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 EIGHT PORTS MAY BE ASSIGNED AT ANY GIVEN TIME. PORT #'s MUST BE RANGE Ø to $1F.

3.29.1 Listing Current Port Assignments

PF will list the names of the board and port for each assigned port number (LUN) when the command is invoked with the port number omitted.

Example:

```
135Bug>PF <CR>
Current port assignments:  (Port #: Board name, Port name)
ØØ: VME135, " 1"    Ø1: VME135, " 2"
135Bug>
```

3.29.2 Configuring a Port

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

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

> Only listed values are accepted when a list is shown. The sole exception is that upper or lower case 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 "^A").

The caret, when entered by itself or following a value, will cause **PF** to issue the previous prompt after each entry.

v Either an upper or lowercase "v" will cause **PF** 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 will cause **PF** to issue the same prompt again. This is supported to be consistent with the operation of other debugger commands. To assume 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 **PF** to exit from the interactive mode and issue the "OK to proceed (Y/N)?".

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

Example: Changing number of stop bits on port number 1.

```
135Bug>PF1 <CR>
Baud rate [11Ø,3ØØ,6ØØ,12ØØ,24ØØ,48ØØ,96ØØ,192ØØ] = 96ØØ? <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          (Note: Carriage return not required)
135Bug>
```

3.29.3 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 boards with adjustable base addressing, i.e., the VMEØ5Ø. Entering no value will select the default address shown.

Baud rate:

The user may choose from the following: 11Ø,3ØØ,6ØØ,12ØØ, 24ØØ,48ØØ,96ØØ,192ØØ.

Parity type:

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

Character width:

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

Number of stop bits:

Only 1 and 2 stop bits are supported.

Synchronization type:

As the debugger is a polled serial input/output environment, most users will use only asynchronous communication. Synchronous modes are permitted but no synchronous protocols are supported by 135Bug.

Synchronization character values:

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

Data equipment type:

Driver authors may require knowledge of the port's data equipment type. Types DTE (Data Terminal Equipment) and DCE (Data Communication Equipment) are permitted but ignored by current drivers.

Automatic hardware hardshake:

Some devices and connection circuitry support hardware handshake. Transmitters may be set up to enable only when the RS-232 signal Clear-to-send is asserted. Receivers may be set up to negate the RS-232 signal Request-to-send when the receiver's FIFO (First-In/First-Out) buffer is full.

Automatic software handshake:

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

3.29.4 Assigning a New Port

PF supports a set of drivers for a number of different boards and the ports on each. To assign one of these to a previously unassigned port number, invoke the command with that number. A message will then be printed to indicate that the port is unassigned and a prompt will be issued to request the name of the board (i.e., VME135, VMEØ5Ø, etc). Presing **RETURN** at this point will cause **PF** to list the currently supported boards and ports. Once the name of the board has been entered, a prompt will be issued for the name of the port. After the port name has been entered, **PF** will attempt 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 **RETURN** will retain the base address shown.

If the configuration of the new port is not fixed, then the interactive configuration mode is entered. Refer to section 3.26.2 above regarding configuring assigned ports. If the new port does have a fixed configuration, then **PF** will issue the "OK to proceed (Y/N)?"prompt immediately.

PF will not initialize any hardware until the user has responded with the letter "Y" to prompt "OK to proceed (Y/N)?". Pressing **BREAK** any time prior to this step or responding with the letter "N" at the prompt will leave the port unassigned. This is only true of ports not previously assigned.

Example: Assigning port 2 to the VMEØ5Ø printer port.

```
135Bug>PF 2 <CR>
Logical unit $Ø2 unassigned
Name of board? <CR>                      (cause PF to list supported boards,
                                         ports)

Boards and ports supported:
VME135:  1, 2
VMEØ5Ø:  1, 2, PTR
Name of board? VMEØ5Ø <CR>               (Note: Upper or lowercase accepted)
Port base address = $FFFF1Ø8Ø? <CR>      (Note: Interactive mode is not
                                         entered as hardware has fixed
                                         configuration)

OK to proceed (Y/N)? Y
135Bug>
```

3.30 Register Display RD

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

The **RD** command is used to display the target state, that is, the register state associated with the target program (refer to the **GO** command). The instruction pointed to by the target PC is also disassembled and displayed. Internally, a register mask specifies which registers will be displayed when **RD < CR>** is executed. At reset time this mask is set to display all **MPU** registers. This register mask can be changed with the **RD** command. The optional arguments allow the user the capability to enable or disable the display of any register or group or registers. This is useful for showing only the registers of interest, minimizing unnecessary data on the screen, and also to save screen space, which is reduced particularly when coprocessor registers are displayed.

The arguments are as follows:

+	is a qualifier indicating that a device or register range is to be added.
-	is a qualifier indicating that a device or register range is to be removed, except when used between two register names. In this case it indicates a register range.
=	is a qualifier indicating that a device or register range is to be set.
/	is a delimiter between device names and register ranges.
< REG1>	is the first register in a range of registers.
< REG2>	is the last register in a range of registers.
< DNAME>	is a device name. This is used to quickly enable or disable all the registers of a device. The available device names are:

MPU	Microprocessor Unit
FPC	Floating Point Coprocessor
PMMU	Paged Memory Management Unit

The following notes should be observed when specifying any arguments in the command line:

1. The qualifier is applied to the next register range only.
2. If no qualifier is specified, a + qualifier is assumed.
3. All device names should appear before any register names.
4. The command line arguments are parsed from left to right, with each field being processed after parsing, thus, the sequence in which qualifiers and registers are organized has an impact on the resultant register mask.
5. When specifying a register range, <REG1> and <REG2> do not have to be of the same class.
6. The register mask used by **RD** is also used by all the exception handler routines, including the trace and breakpoint exception handlers.

The **MPU** registers in ordering sequence are:

Number of registers		
1Ø	System Registers	(PC,SR,USP,MSP,ISP,VBR,SFC,DFC,CACR,CAAR)
8	Data Registers	(DØ-D7)
8	Address Registers	(AØ-A7)

The **FPC** registers in ordering sequence are:

Number of registers		
3	System Registers	(FPCR,FPSR,FPIAR)
8	Data Registers	(FPØ-FP7)

The **PMMU** registers in ordering sequence are:

Number of registers		
4	Address Translation Control	(CRP,SRP,DRP,TC)
6	Control/Status/Access Level	(PCSR,PSR,AC,CAL,VAL,SCC)
8	Breakpoint Acknowledge Data	(BADØ-BAD7)
8	Breakpoint Acknowledge Control	(BACØ-BAC7)

Example 1:

```
135Bug>RD <CR>
PC   =ØØØØ3ØØØ SR  =27ØØ=TR:OFF_S._7_.....
USP  =ØØØØF83Ø MSP =ØØØØ3C18 ISP* =ØØØØ4ØØØ VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØØØØØØ D1  =ØØØØØØØØ D2   =ØØØØØØØØ D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5  =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1  =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5  =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØ4ØØØ
ØØØØ3ØØØ 424F             DC.W    $424F
135Bug>
```

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	TØ	Mnemonic	Description
Ø	Ø	TR:OFF	Trace off
Ø	1	TR:CHG	Trace on change of flow
1	Ø	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 Ø 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 it cleared.

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

Function Code	Mnemonic	Description
Ø	FØ	Undefined
1	UD	User Data
2	UP	User Program
3	F3	Undefined
4	F4	Undefined
5	SD	Supervisor Data
6	SP	Supervisor Program
7	CS	CPU Space

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

Example 2: To set the display to D6 and A3 only.

```
135Bug>RD =D6/A3 <CR>
D6    =ØØØØØØØØ A3    =ØØØØØØØØ
ØØØØ3ØØØ 4AFC             ILLEGAL
135Bug>
```

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

Example 3: To restore all the MPU registers.

```
135Bug>RD +MPU <CR>
PC    =ØØØØ3ØØØ SR    =27ØØ=TR:OFF_S._7_.....
USP   =ØØØØ383Ø MSP   =ØØØØ3C18 ISP* =ØØØØ4ØØØ VBR  =ØØØØØØØØ
SFC   =Ø=FØ     DFC   =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ    =ØØØØØØØØ D1    =ØØØØØØØØ D2   =ØØØØØØØØ D3   =ØØØØØØØØ
D4    =ØØØØØØØØ D5    =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ    =ØØØØØØØØ A1    =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4    =ØØØØØØØØ A5    =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØ4ØØØ
ØØØØ3ØØØ 4AFC             ILLEGAL
135Bug>
```

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

Example 4:

```
135Bug>RD +FPC <CR>
PC   =00003000 SR   =2700=TR:OFF_S._7_.....
USP  =00003830 MSP  =00003C18 ISP* =00004000 VBR  =00000000
SFC  =0=F0     DFC  =0=F0     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
135Bug>
```

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

Example 5: To display only the **PMMU** registers.

```
135Bug>RD =PMMU <CR>
CRP  =00000000_00000000      SRP  =00000000_00000000
DRP  =00000000_00000000      TC   =00000000
PCSR =0000-.._0              PSR  =0000-........._0
AC   =0000     CAL  =00      VAL  =00        SCC  =00
BAD0 =0000     BAD1 =0000    BAD2 =0000      BAD3 =0000
BAD4 =0000     BAD5 =0000    BAD6 =0000      BAD7 =0000
BAC0 =0000     BAC1 =0000    BAC2 =0000      BAC3 =0000
BAC4 =0000     BAC5 =0000    BAC6 =0000      BAC7 =0000
135Bug>
```

The PCSR and PSR registers above include a mnemonic portion. For the PCSR register, the bits are:

F	Flush bit
LW	Lock Warning bit
TA	Task Alias field (3 bits)

For the PSR register, the bits are:

B	Bus Error
L	Limit Violation
S	Supervisor Only
A	Access Level Violation
W	Write Protected
I	Invalid
M	Modified
G	Gate
C	Globally Sharable
N	Number of Levels (3 bits)

3.31 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** pushbutton on the VME135's front panel.

Two RESET levels are available:

COLD - This is the standard mode of operation, and is the one defaulted on power on. 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, the port configurations, and any other static variables in the system.

NOTE: If the VME135 is the system controller, pressing the **RESET** pushbutton will reset all the modules in the system, including disk controllers like the VME320 or VME360. This may cause the disk controller configuration to be out of phase with respect to the disk configuration tables in memory.

Example:

```
135Bug>RESET <CR>
Cold/Warm Start = C (C/W)? W          Set to warm start
135Bug>
                                      Press the RESET pushbutton
VME135 Debugger/Diagnostic Version 2.0 - 3/2/88
Warm Start
135Bug>
```

3.32 Register Modify RM

RM < REG>

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

Example 1:

```
135Bug>RM D4 <CR>
D4    =12345678? ABCDEF^ <CR>      Modify register and backup
D3    =00000000? 3000. <CR>        Modify register and exit
135Bug>
```

Example 2:

```
135Bug>RM SFC <CR>
SFC  =7=CS    ? 1= <CR>            Modify register and re-open
SFC  =1=UD    ? . <CR>             Exit
135Bug>
```

The **RM** command is also used to modify the Floating Point Coprocessor registers (the MC68881).

Example 3:

```
135Bug>RM FPSR <CR>
FPSR =ØØØØØØØØ-(CC=....    ) ? FØØØØØØ <CR>
FPIAR=ØØØØØØØØ ? <CR>
FPØ  =Ø_7FFF_FFFFFFFFFFFFFFFF= Ø.FFFFFFFFFFFFFFFF_E-ØFFF? Ø_1234_5 <CR>
FP1  =Ø_7FFF_FFFFFFFFFFFFFFFF= Ø.FFFFFFFFFFFFFFFF_E-ØFFF? 1.25E3 <CR>
FP2  =Ø_7FFF_FFFFFFFFFFFFFFFF= Ø.FFFFFFFFFFFFFFFF_E-ØFFF? 1_7F_3FF <CR>
FP3  =Ø_7FFF_FFFFFFFFFFFFFFFF= Ø.FFFFFFFFFFFFFFFF_E-ØFFF? 11ØØ_9261_3 <CR>
FP4  =Ø_7FFF_FFFFFFFFFFFFFFFF= Ø.FFFFFFFFFFFFFFFF_E-ØFFF? &564 <CR>
FP5  =Ø_7FFF_FFFFFFFFFFFFFFFF= Ø.FFFFFFFFFFFFFFFF_E-ØFFF? Ø_5FF_FØAB <CR>
FP6  =Ø_7FFF_FFFFFFFFFFFFFFFF= Ø.FFFFFFFFFFFFFFFF_E-ØFFF? 3.1415 <CR>
FP7  =Ø_7FFF_FFFFFFFFFFFFFFFF= Ø.FFFFFFFFFFFFFFFF_E-ØFFF? -2.74638369E-36. <CR>
135Bug>

135Bug>RD +FPC <CR>
PC   =ØØØØ2ØØØ SR   =27ØØ=TR:OFF_S._7_.....
USP  =ØØØØ383Ø MSP  =ØØØØ3C18 ISP* =ØØØØ4ØØØ VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØØØØØØ D1   =ØØØØØØØØ D2   =ØØØØØØØØ D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØ4ØØØ
FPCR =ØØØØØØØØ FPSR =ØFØØØØØØ-(CC=NZI[NAN])  FPIAR=ØØØØØØØØ
FPØ  =Ø_1234_5ØØØØØØØØØØØØØØØ= 6.6258385370745493_E-353Ø
FP1  =Ø_4ØØ9_9C4ØØØØØØØØØØØØØ= 1.25ØØØØØØØØØØØØØØ_E-ØØØ3
FP2  =1_3FFF_BFFØØØØØØØØØØØØØ=-1.49951171875ØØØØØ_E-ØØØØ
FP3  =1_3C9D_BCEECF12DØ61BED9=-3.ØØØØØØØØØØØØØØØØ_E-Ø261
FP4  =Ø_4ØØ8_8DØØØØØØØØØØØØØØ= 5.64ØØØØØØØØØØØØØØ_E-ØØØ2
FP5  =Ø_41FF_F8558ØØØØØØØØØØØ= 2.6Ø12612226385672_E-Ø154
FP6  =Ø_4ØØØ_C9ØE56Ø4189374BC= 3.1415ØØØØØØØØØØØØ_E-ØØØØ
FP7  =1_3F88_E9A2FØB8D678C318=-2.74638369ØØØØØØØØ_E-ØØ36
ØØØØ2ØØØ ØØØØØØØØ             OR.B    #Ø,DØ
135Bug>
```

The RM command is also used to modify the Paged Memory Management Unit registers (the MC68851).

Example 4:

```
135Bug>RM CRP <CR>
CRP  =00000000_00000000        ? <CR>
SRP  =00000000_00000000        ? <CR>
DRP  =00000000_00000000        ? 12345678_12345678 <CR>
TC   =00000000 ? 87654321 <CR>
PCSR =0000-.._0? <CR>
PSR  =0000-........._0? <CR>
AC   =0000 ? <CR>
CAL  =00 ? <CR>
VAL  =00 ? <CR>
SCC  =00 ? <CR>
BAD0 =0000      ? <CR>
BAD1 =0000      ? <CR>
BAD2 =0000      ? <CR>
BAD3 =0000      ? <CR>
BAD4 =0000      ? <CR>
BAD5 =0000      ? <CR>
BAD6 =0000      ? <CR>
BAD7 =0000      ? <CR>
BAC0 =0000      ? <CR>
BAC1 =0000      ? <CR>
BAC2 =0000      ? <CR>
BAC3 =0000      ? <CR>
BAC4 =0000      ? <CR>
BAC5 =0000      ? <CR>
BAC6 =0000      ? <CR>
BAC7 =0000      ? . <CR>
135Bug>
```

```
135Bug>RD +PMMU <CR>
PC   =00002000 SR   =2700=TR:OFF_S._7_.....
USP  =00003830 MSP  =00003C18 ISP* =00004000 VBR  =00000000
SFC  =0=F0     DFC  =0=F0     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
CRP  =00000000_00000000       SRP  =00000000_00000000
DRP  =12345678_12345678       TC   =87654321
PCSR =0000-.._0               PSR  =0000-........._0
AC   =0000     CAL  =00       VAL  =00       SCC  =00
BAD0 =0000     BAD1 =0000     BAD2 =0000     BAD3 =0000
BAD4 =0000     BAD5 =0000     BAD6 =0000     BAD7 =0000
BAC0 =0000     BAC1 =0000     BAC2 =0000     BAC3 =0000
BAC4 =0000     BAC5 =0000     BAC6 =0000     BAC7 =0000
00002000 00000000           OR.B    #0,D0
135Bug>
```

3.33 Switch Directories

SD

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

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

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

Example 1:

```
135Bug>SD <CR>
135Diag>                    ( The user has changed from the debugger)
                            ( directory to the diagnostic directory,)
                            ( as can be seen by the " 135Diag> "    )
                            ( prompt.                               )
```

Example 2:

```
135Diag>SD <CR>
135Bug>                     ( The user is now back in the debugger  )
                            ( directory.                            )
```

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 135Bug.

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 Ø count is encountered, control will be returned to 135Bug.

The trace functions are implemented with the trace bits (TØ, T1) in the MC68Ø2Ø status register; therefore, these bits should not be modified by the user while using the trace commands.

Example: (The following program resides at location $1ØØØØ)

```
135Bug>MD 1ØØØØ;DI <CR>
ØØØ1ØØØØ 22ØØ             MOVE.L   DØ,D1
ØØØ1ØØØ2 4282             CLR.L    D2
ØØØ1ØØØ4 D4Ø1             ADD.B    D1,D2
ØØØ1ØØØ6 E289             LSR.L    #1,D1
ØØØ1ØØØ8 66FA             BNE.B    $1ØØØ4
ØØØ1ØØØA E2ØA             LSR.B    #1,D2
ØØØ1ØØØC 55C2             SCS      D2
ØØØ1ØØØE 6ØFE             BRA.B    $1ØØØE
135Bug>
```

Initialize PC and DØ:

```
135Bug>RM PC <CR>
PC    =ØØØØ8ØØØ ? 1ØØØØ. <CR>
135Bug>RM DØ <CR>
DØ    =ØØØØØØØØ ? 8F41C. <CR>
135Bug>
```

Display target registers and trace one instruction:

```
135Bug>RD <CR>
PC    =00010000 SR    =2700=TR:OFF_S._7_.....
USP   =0000382C MSP   =00003C14 ISP* =00004000 VBR  =00000000
SFC   =0=F0     DFC   =0=F0     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
135Bug>T <CR>
PC    =00010000 SR    =2700=TR:OFF_S._7_.....
USP   =0000382C MSP   =00003C14 ISP* =00004000 VBR  =00000000
SFC   =0=F0     DFC   =0=F0     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
135Bug>
```

Trace next instruction:

```
135Bug><CR>
PC    =00010004 SR    =2704=TR:OFF_S._7_..Z..
USP   =0000382C MSP   =00003C14 ISP* =00004000 VBR  =00000000
SFC   =0=F0     DFC   =0=F0     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
135Bug>
```

Trace the next two instructions:

```
135Bug>T 2 <CR>
PC   =00010000 SR   =2700=TR:OFF_S._7_.....
USP  =0000382C MSP  =00003C14 ISP* =00004000 VBR  =00000000
SFC  =0=F0     DFC  =0=F0     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   =00010000 SR   =2700=TR:OFF_S._7_.....
USP  =0000382C MSP  =00003C14 ISP* =00004000 VBR  =00000000
SFC  =0=F0     DFC  =0=F0     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   410004
135Bug>
```

3.35 Trace On Change Of Control Flow **TC**

TC [< COUNT>]

The **TC** command will start execution at the address in the target PC and will begin tracing upon the detection of an instruction that causes a change of control flow, such as JSR, BSR, RTS, etc. This means that execution will be 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 135Bug.

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 will recognize a breakpoint only if it is at a change of flow instruction. In all cases, if a breakpoint with Ø count is encountered, control will be returned to 135Bug.

The trace functions are implemented with the trace bits (TØ, T1) in the MC68Ø2Ø status register, therefore, these bits should not be modified by the user while using the trace commands.

Example: (The following program resides at location $1ØØØØ)

```
135Bug>MD 1ØØØØ;DI <CR>
ØØØ1ØØØØ 22ØØ                    MOVE.L    DØ,D1
ØØØ1ØØØ2 4282                    CLR.L     D2
ØØØ1ØØØ4 D4Ø1                    ADD.B     D1,D2
ØØØ1ØØØ6 E289                    LSR.L     #1,D1
ØØØ1ØØØ8 66FA                    BNE.B     $1ØØØ4
ØØØ1ØØØA E2ØA                    LSR.B     #1,D2
ØØØ1ØØØC 55C2                    SCS       D2
ØØØ1ØØØE 6ØFE                    BRA.B     $1ØØØE
135Bug>
```

Initialize PC and DØ:

```
135Bug>RM PC <CR>
PC    =00008000 ? 10000. <CR>
135Bug>RM D0 <CR>
D0    =00000000 ? 8F41C. <CR>
135Bug>
```

Trace on change of flow:

```
135Bug>TC <CR>
00010008 66FA                BNE.B   $10004
PC   =00010004 SR   =2700=TR:OFF_S._7_.....
USP  =0000382C MSP  =00003C14 ISP* =00004000 VBR  =00000000
SFC  =0=F0     DFC  =0=F0     CACR =0=..     CAAR =00000000
D0   =0008F41C D1   =00047A0E D2   =0000001C D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00004000
00010004 D401                ADD.B   D1,D2
135Bug>
```

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

3.36 Transparent Mode TM

TM [n] [< ESCAPE>]

The **TM** command essentially connects the console serial port and the host port together, allowing the user to communicate with a host computer. A message displayed by **TM** shows the current escape character, i.e., the character used to exit the transparent mode. The two ports remain "connected" until the escape character is received by the console port. The escape character is not transmitted to the host and at power up or reset is initialized to $Ø1=^A.

The optional port number "n" allows the user to specify which port will be the "host" port. If omitted, port 1 will be 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 rates use the **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 :	$Ø3	Set escape character to ^ C
ASCII character :	'c	Set escape character to c
control 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:

135Bug>**TM** <**CR**>	Enter **TM**
Escape character: $Ø1=^ A	Exit code is always displayed
<**^A**>	Exit transparent mode
135Bug>	

Example 2:

135Bug>**TM** ^**G** <**CR**>	Enter **TM** and set escape character
Escape character: $Ø7=^ G	to ^ G
<**^G**>	Exit transparent mode
135Bug>	

3.37 Trace To Temporary Breakpoint TT

TT < ADDR>

The **TT** command will set a temporary breakpoint at the specified address and will trace until a breakpoint with Ø count is encountered. The temporary breakpoint is then removed (TT is analogous to the **GT** command) and control is returned to 135Bug. 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 Ø count is encountered, control will be returned to 135Bug.

The trace functions are implemented with the trace bits (TØ, T1) in the MC68Ø2Ø status register; therefore, these bits should not be modified by the user while using the trace commands.

Example: (The following program resides at location $1ØØØØ)

```
135Bug>MD 1ØØØØ;DI <CR>
ØØØ1ØØØØ 22ØØ                    MOVE.L    DØ,D1
ØØØ1ØØØ2 4282                    CLR.L     D2
ØØØ1ØØØ4 D4Ø1                    ADD.B     D1,D2
ØØØ1ØØØ6 E289                    LSR.L     #1,D1
ØØØ1ØØØ8 66FA                    BNE.B     $1ØØØ4
ØØØ1ØØØA E2ØA                    LSR.B     #1,D2
ØØØ1ØØØC 55C2                    SCS       D2
ØØØ1ØØØE 6ØFE                    BRA.B     $1ØØØE
135Bug>
```

Initialize PC and DØ:

```
135Bug>RM PC <CR>
PC   =ØØØØ8ØØØ ? 1ØØØØ. <CR>
135Bug>RM DØ <CR>
DØ   =ØØØØØØØØ ? 8F41C. <CR>
135Bug>
```

Trace to temporary breakpoint:

```
135Bug>TT 1ØØØ6 <CR>
PC   =ØØØ1ØØØ2 SR   =27ØØ=TR:OFF_S._7_.....
USP  =ØØØØ382C MSP  =ØØØØ3C14 ISP* =ØØØØ4ØØØ VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØ8F41C D1   =ØØØ8F41C D2   =ØØØØØØØØ D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØ4ØØØ
ØØØ1ØØØ2 4282               CLR.L   D2
PC   =ØØØ1ØØØ4 SR   =27Ø4=TR:OFF_S._7_..Z..
USP  =ØØØØ382C MSP  =ØØØØ3C14 ISP* =ØØØØ4ØØØ VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØ8F41C D1   =ØØØ8F41C D2   =ØØØØØØØØ D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØ4ØØØ
ØØØ1ØØØ4 D4Ø1               ADD.B   D1,D2
At Breakpoint
PC   =ØØØ1ØØØ2 SR   =27ØØ=TR:OFF_S._7_.....
USP  =ØØØØ382C MSP  =ØØØØ3C14 ISP* =ØØØØ4ØØØ VBR  =ØØØØØØØØ
SFC  =Ø=FØ     DFC  =Ø=FØ     CACR =Ø=..     CAAR =ØØØØØØØØ
DØ   =ØØØ8F41C D1   =ØØØ8F41C D2   =ØØØØØØ1C D3   =ØØØØØØØØ
D4   =ØØØØØØØØ D5   =ØØØØØØØØ D6   =ØØØØØØØØ D7   =ØØØØØØØØ
AØ   =ØØØØØØØØ A1   =ØØØØØØØØ A2   =ØØØØØØØØ A3   =ØØØØØØØØ
A4   =ØØØØØØØØ A5   =ØØØØØØØØ A6   =ØØØØØØØØ A7   =ØØØØ4ØØØ
ØØØ1ØØØ6 E289               LSR.L   #1,D1
135Bug>
```

3.38 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 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 will be 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 will cause the records to be compared to memory at different locations then 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 $Ø 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 (=), will be sent to the host before 135Bug 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. The 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 will also be echoed back to the host port by the host and will appear 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** will keep looking for a <LF> character from the host, signifying the end of the echoed command. No data records will be processed until this <LF> is received. If the host system does not echo characters, **VE** will still keep 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 **VE** out of the loop so that data records will be processed.

The other options have the following effects:

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

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

During a verify operation, an S-Record's data is compared to memory beginning with the address contained in the S-Record's 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 will be printed to the screen and 135Bug's error handler will be invoked to point to the faulty character.

As mentioned, if the embedded checksum of a record does not agree with the checksum calculated by 135Bug AND if the checksum comparison has not been disabled via the "-C" option then an error condition exists. A message will be 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.

```
                               *  Test Program.
                               *
           65Ø4ØØØØ                        ORG        $65Ø4ØØØØ

  65Ø4ØØØ 7ØØ1                             MOVEQ.L    #1,DØ
  65Ø4ØØ2 DØ88                             ADD.L      AØ,DØ
  65Ø4ØØ4 4AØØ                             TST.B      DØ
  65Ø4ØØ6 4E75                             RTS
                                           END
******  TOTAL ERRORS     Ø--
******  TOTAL WARNINGS   Ø--
```

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

```
SØØFØØØØ544553545333533372Ø2ØØ1Ø15E
S3ØD65Ø4ØØØØ7ØØ1DØ884AØØ4E75B3
S7Ø565Ø4ØØØØ91
```

This file was downloaded into memory at address $4ØØØØ. The program may be examined in memory using the **MD** (Memory Display) command.

```
135Bug>MD 4ØØØØ:4;DI <CR>
ØØØ41ØØØ 7ØØ1                  MOVEQ.L #1,DØ
ØØØ4ØØØ2 DØ88                  ADD.L   AØ,DØ
ØØØ4ØØØ4 4AØØ                  TST.B   DØ
ØØØ4ØØØ6 4E75                  RTS
135Bug>
```

Suppose that the user wants to make sure that the program has not been destroyed in memory. The **VE** command will be used to perform a verification.

```
135Bug>VE -65000000;X=COPY TEST.MX,# <CR>
S00F000054455354533353372020001015E
S30D65040000700 1D0884A004E75B3
S70565040000 91
Verify passes.
135Bug>
```

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.

```
135Bug>M 40002 <CR>
00040002 D088 ? D089. <CR>
135Bug>VE -65000000;X=COPY TEST.MX,# <CR>
S00F000054455354533353372020001015E
S30D650400007001D0884A004E75B3
S7056504000091
```

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

```
S30D65040000------88--------B3
135Bug>
```

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

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4
USING THE ONE-LINE ASSEMBLER/DISASSEMBLER

4.1 Introduction

Included as part of the 135Bug 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/MC68851/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 135Bug 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 MC68020 Assembly Language

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

- Operations
  - MC68020 machine-instruction operation code
  - Directives (pseudo-ops)
- Operators
- Special symbols

4.1.1.1 Machine-Instruction Operation Codes

The part of the assembly language that provides the mnemonic machine-instruction operation codes for the MC68020/MC68851/MC68881 machine instructions are described in the MC68020UM, MC68851UM, and MC68881UM Technical User's Manuals. Refer to these manuals for any question concerning operation codes.

4.1.1.2 Directives

Normally, assembly language can contain mnemonic directives which specify auxiliary actions to be performed by the assembler. The 135Bug assembler recognizes only two directives called DC.W (define constant) and SYSCALL. These two directives are used to define data within the program and to make calls to 135Bug utilities (refer to paragraphs 4.2.3 and 4.2.4, respectively).

4.1.2 Comparison with MC68020 Resident Structured Assembler

There are several major differences between the 135Bug 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 135Bug assembler processes each line of a program as an individual unit. Due mainly to this basic functional difference, the capabilities of the 135Bug assembler are more restricted:

1. Label and line numbers are not used. Labels are used to reference other lines and locations in a program. The one-line assembler has no knowledge of other lines and, therefore, cannot make the required association between a label and the label definition located on a separate line.

2. Source lines are not saved. In order to read back a program after it has been entered, the machine code is disassembled and then displayed as mnemonic and operands.

3. Only two directives (DC.W and SYSCALL) are accepted.

4. No macro operation capability is included.

5. No conditional assembly is used.

6. Several symbols recognized by the resident assembler are not included in the 135Bug assembler character set. These symbols include > and < . Three other symbols have multiple meaning to the resident assembler, depending on the context. These are:

   a. Asterisk (*) -- Multiply or current PC.
   b. Slash (/) -- Divide or delimiter in a register list.
   c. Ampersand (&) -- And or decimal number prefix.

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 135Bug assembler are acceptable to the resident assembler except as described above.

4.2 Source Program Coding

A source program is a sequence of source statements arranged in a logical way to perform a predetermined task. Each source statement occupies a line and must be either an executable instruction, a DC.W directive, or a SYSCALL assembler directive. Each source statement follows a consistent source line format.

4.2.1 Source Line Format

Each source statement is a combination of operation and, as required, operand fields. Line numbers, labels and comments are NOT used.

4.2.1.1 Operation Field

Since 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:

1. Operation codes -- Which correspond to the MC68020/MC68851/MC68881 instruction set.

2. Define Constant directive -- DC.W is recognized to define a constant in a word location.

3. System Call directive -- SYSCALL is used to call 135Bug 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	(AØ),A1	Longword size is assumed (.B, .W not allowed); this instruction loads the effective address of the first operand into A1.
ADD.B	(AØ),DØ	This instruction adds the byte whose address is (AØ) to the lowest order byte in DØ.
ADD	D1,D2	This instruction adds the low order word of D1 to the low order word of D2. (W is the default size code.)
ADD.L	A3,D3	This instruction adds the entire 32-bit (longword) contents of A3 to D3.

Example (illegal):

SUBA.B	#5,A1	Illegal size specification (.B not allowed on SUBA). This instruction would have subtracted the value 5 from the low order byte of A1; byte operations on address registers are not allowed.

4.2.1.2 Operand Field

If present, the operand field follows the operation filed and is separated from the operation field by at least one space. When two or more operand subfields appear within a statement, they must be separated by a comma. In an instruction like ' ADD D1,D2', the first subfield (D1) is called the source effective address field, and the second subfield (D2) is called the destination < EA> field. Thus, the contents on D1 are added to the contents of D2 and the result is saved in register D2. In the instruction ' MOVE D1,D2', the first subfield (D1) is the sending field and the second subfield (D2) is the receiving field. In other words, for most two-operand instructions, the format

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 offest. Otherwise, it is treated as a straight unsigned hexadecimal.

For example,

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

disassembles to

```
ØØØØ3ØØØ  21FCØØØØ 12345678     MOVE.L    #$1234,($5678).W
ØØØØ3ØØ8  21E8FFFC 5678         MOVE.L    -$4(AØ),($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 68Ø2Ø instruction mnemonics. Numbers are recognized as binary, octal, decimal, and hexadecimal, with hexadecimal as the default case.

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

    &12334
    -&987654321

b. Hexadecimal - is a string of hexadecimal digits (Ø to 9, A to 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.

```
ØØØØ3ØØØ  21FCØØØØ 12345678    MOVE.L   #$1234,($5678).W
ØØ5ØØØ    ØØ53                 DC.W     'S'
ØØ5ØØ2    223C41424344         MOVE.L   #'ABCD',D1
ØØ5ØØ8    3536                 DC.W     '56'
```

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

Pseudo Registers	
RØ-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.
DØ-D7	Data Registers.
AØ-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 in the status register (SR).

Floating Point Coprocessor Registers	
FPCR	Control Register.
FPSR	Status Register.
FPIAR	Instruction Address Register.
FPØ-FP7	Floating Point Data Registers.

Paged Memory Management Unit Coprocessor Registers	
PSR	Status Register.
PCSR	Cache Status Register.
AC	Access Control Register.
CRP	CPU Root Pointer.
SRP	Supervisor Root Pointer.
DRP	DMA Root Pointer.
TC	Translation Control Register.
BACØ-BAC7	Breakpoint Acknowledge Control Registers.
BADØ-BAD7	Breakpoint Acknowledge Data Registers.
CAL	Current Access Level.
VAL	Validate Access Level.
SCC	Stack Change Control.

4.2.1.5 Character Set

The character set recognized by the 135Bug assembler is a subset of ASCII, and is listed below:

1. The letters A through Z (uppercase and lowercase)

2. The integers Ø through 9

3. Arithmetic operators: + - * / << >> ! &

4. Parentheses ( )

5. 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.

6. Five separating characters:
   - Space
   - , (comma)
   - . (period)
   - / (slash)
   - - (dash)

7. The character * (asterisk) indicates current location.

4.2.2 Addressing Modes

Effective address modes, combined with operation codes, define the particular function to 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 135Bug one-line assembler.

TABLE 4-1. 135Bug ASSEMBLER ADDRESSING MODES

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

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

1) Binary numbers (%1Ø)
2) Octal numbers (@765..Ø)
3) Decimal numbers (&987..Ø)
4) Hexadecimal numbers ($FED..Ø)
5) String literals ('CHAR')
6) Offset registers (RØ-R7)
7) Program counter (*)

Allowed operators are:

1)	Addition	+
2)	Subtraction	-
3)	Multiply	*
4)	Divide	/
5)	Shift left	<<
6)	Shift right	>>
7)	Bitwise or	!
8)	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 via the use of parentheses.

Possible points of confusion:

1. 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     DØ      means CLR.W register DØ. On the other hand,

CLR     $DØ
CLR     ØDØ
CLR     +DØ
CLR     DØ+Ø    all mean CLR.W memory location $DØ.
```

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

   For parsing algebraic expressions, the order of parsing is

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

   with a possible left or right parenthesis.

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

1) ***	Means	PC	*	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.

1) Address register indirect with index, base displacement.
2) Address register memory indirect post-indexed.
3) Address register memory indirect pre-indexed.
4) Program counter indirect with index, base displacement.
5) Program counter memory indirect post-indexed.
6) 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:

1. The user may terminate the operand at any time by specifying ")". Example:

```
CLR     ( )        or
CLR     (,,)       is equivalent to
CLR     (Ø.N,ZAØ,ZDØ.W*1)
```

2. The user may skip a field by "stepping past" it with a comma. Example:

```
CLR     (D7)       is equivalent to
CLR     ($D7,ZAØ,ZDØ.W*1)
```

but

```
CLR     (,,D7)     is equivalent to
CLR     (Ø.N,ZAØ,D7.W*1)
```

3. If the user does not specify the base register, the default "ZAØ" is forced.

4. If the user does not specify the index register, the default "ZDØ.W*1" is forced.

5. Any unspecified displacements are defaulted to "Ø.N".

The rules for parsing the memory indirect addressing modes are the same as above with the following additions.

1. The subfield that begins with "[" must be terminated with a matching "]".

2. If the text given is insufficient to distinguish between the pre-indexed or post-indexed 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 if word (.W) size is specified. An ASCII string is recognized when characters are enclosed inside single quotes (' '). Each character (7 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 left justified. A maximum of two ASCII characters may be entered for each DC.W directive.

Examples are:

ØØØ1ØØ22	Ø4D2	DC.W	1234	Decimal number
ØØØ1ØØ24	AAFE	DC.W	&AAFE	Hexadecimal number
ØØØ1ØØ26	4142	DC.W	'AB'	ASCII String
ØØØ1ØØ28	5443	DC.W	'TB'+1	Expression
ØØØ1ØØ2A	ØØ43	DC.W	'C'	ASCII character is right justified

4.2.4 SYSCALL System Call Directive

The function of this directive is to aid the user in making the TRAP #15 calls to the system functions. The format for this directive is:

SYSCALL <function name >

For example, the following two pieces of code will produce identical results.

```
TRAP    #$F
DC.W    Ø
```

or

```
SYSCALL .INCHR
```

Refer to Chapter 5 (SYSTEM CALLS), for a complete listing of all the functions provided.

4.3 Entering and Modifying Source Program

User programs are entered into the memory using the one-line assembler/disassembler. The program is entered in assembly language statements on a line-by-line basis. The source code is not saved as it is converted immediately to machine code upon entry. This imposes several restrictions on the type of source line that can be entered.

Symbols and labels, other than the defined instruction mnemonics, are not allowed. The assembler has no means to store the associated values of the symbols and labels in lookup tables. This forces the programmer to use memory addresses and to enter data directly rather than use labels.

Also, editing is accomplished by retyping the entire new source line. Lines can be added or deleted by moving a block of memory data to free up or delete the appropriate number of locations (refer to the **BM** command).

4.3.1 Invoking the Assembler/Disassembler

The assembler/disassembler is invoked using the ;DI option of the **MM** (Memory Modify) and **MD** (Memory Display) commands:

MM <ADDR> ;DI

where

<**CR**> sequences to next instruction
.<**CR**> exits command

and

MD[S] <ADDR>[:<count>|<ADDR>];DI

The **MM** (;DI option) is used for program entry and modification. When this command is used, the memory contents at the specified location are disassembled and displayed. A new or modified line can be entered if desired.

The disassembled line can be an 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 may be entered immediately following the disassembled line, using the format discussed in paragraph 4.2.1:

```
135Bug>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:

```
135Bug>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 the **PF** command). In this case, the above example will look as follows:

```
135Bug>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 a "^" under the field suspected of causing the error and an error message.

The location being accessed is redisplayed:

```
135Bug>MM 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 ?
```

4.3.3 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's destination in the operand field of the instruction. The offset will be calculated by the assembler. The user must append the appropriate size extension to the branch instruction.

To reference a current location in an operand expression, the character "*" (asterisk) can be used. Examples are:

```
00030000    60004094             BRA *+$4096
00030000    60FE                 BRA.B *
00030000    4EF90003 0000        JMP *
00030000    4EF00130 00030000    JMP (*,A0,D0)
```

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

NOTE: Branch sizes must be entered as ".B" or ".W" as opposed to ".S" and ".L".

4.3.4 Assembler Output/Program Listings

A listing of the program is obtained using the **MD** (Memory Display) 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 will be equal to the line count.

To obtain a hard copy listing of a program, use the **PA** (Printer Attach) command to activate the Port 1 printer. A **MD** (Memory

Display) to the terminal will then cause 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 of an address register; all other numbers are displayed in unsigned hexadecimal.

CHAPTER 5
SYSTEM CALLS

5.1 Introduction

This chapter describes the 135Bug TRAP #15 handler, which allows system calls from user programs. The system calls can be used to access selected functional routines contained within 135Bug, including input and output routines. TRAP #15 may also be used to transfer control to 135Bug at the end of a user program (refer to the **.RETURN** function).

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 the VME135 board's console port. The defaults may be changed, however, using the **.REDIR_I** and **.REDIR_O** functions.

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 135Bug
DC.W $xxxx   Routine being requested (xxxx = code)
```

In some of the examples shown in the following descriptions, a SYSCALL macro is used. This macro automatically assembles the TRAP #15 call followed by the Define Constant for the function code. For clarity, the SYSCALL macro is as follows:

```
SYSCALL     MACRO
            TRAP      #15
            DC.W      \1
            ENDM
```

Using the SYSCALL macro, the system call would appear in the user program as follows:

```
SYSCALL     <routine name>
```

It is, of course, necessary to create an equate file with the routine names equated to their respective codes.

When using 135Bug's one-line assembler/disassembler, the SYSCALL macro and the equates are predefined. Simply write in "SYSCALL" followed by a space and the function, then the carriage return.

Example:

```
135Bug>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>
135Bug>
```

5.1.2 String Formats for I/O

Within the context of the TRAP #15 handler there are two formats for strings:

Pointer/Pointer Format - The string is defined by a pointer to the first character and a pointer to the last character + 1.

Pointer/Count Format - The string is defined by a pointer to a count byte which contains the count of characters in the string followed by the string itself.

A line is defined as a string followed by a carriage return and a line feed.

5.2 System Call Routines

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

TABLE 5-2. 135Bug SYSTEM CALL ROUTINES

Code	Function	Description
$0000	.INCHR	Input character
$0001	.INSTAT	Input serial port status
$0002	.INLN	Input line (pointer/pointer format)
$0003	.READSTR	Input string (pointer/count format)
$0004	.READLN	Input line (pointer/count format)
$0005	.CHKBRK	Check for break
$0010	.DSKRD	Disk read
$0011	.DSKWR	Disk write
$0012	.DSKCFIG	Disk configure
$0014	.DSKFMT	Disk format
$0015	.DSKCTRL	Disk control
$0020	.OUTCHR	Output character
$0021	.OUTSTR	Output string (pointer/pointer format)
$0022	.OUTLN	Output line (pointer/pointer format)
$0023	.WRITE	Output string (pointer/count format)
$0024	.WRITELN	Output line (pointer/count format)
$0025	.WRITDLN	Output line with data (pointer/count format)
$0026	.PCRLF	Output carriage return and line feed
$0027	.ERASLN	Erase line
$0028	.WRITD	Output string with data (pointer/count format)
$0029	.SNDBRK	Send break
$0040	.TM_INI	Timer initialization
$0041	.TM_STR0	Start timer at T = 0
$0042	.TM_RD	Read timer
$0043	.DELAY	Wait for the specified delay
$0060	.REDIR	Redirect I/O of a TRAP 15 function
$0061	.REDIR_I	Redirect input
$0062	.REDIR_O	Redirect output
$0063	.RETURN	Return to 135Bug
$0064	.BINDEC	Convert binary to Binary Coded Decimal (BCD)
$0067	.CHANGEV	Parse value
$0068	.STRCMP	Compare two strings (pointer/count format)
$0069	.MULU32	Multiply two 32-bit unsigned integers
$006A	.DIVU32	Divide two 32-bit unsigned integers

5.2.1 .INCHR Function .INCHR

TRAP FUNCTION: .INCHR - Input character routine-

CODE: $ØØØØ

DESCRIPTION: Will read 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)+,DØ    Load character in DØ
```

5.2.2 .INSTAT Function .INSTAT

TRAP FUNCTION: .INSTAT - Input serial port status-

CODE: $ØØØ1

DESCRIPTION: Used to see if there are character 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)+,(AØ)+  In buffer
        BRA.S     LOOP         Check for more
EMPTY
```

5.2.3 .INLN Function .INLN

TRAP FUNCTION: .INLN - Input line routine-

CODE: $ØØØ2

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 AØ contains the address where the string is to go:

```
SUBQ.L   #4,A7      Allocate space for result
PEA      (AØ)       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 included in the string. Control character processing as described in section 2.2, Terminal Input/Output Control, is in effect.

5.2.4 .READSTR Function .READSTR

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

CODE: $ØØØ3

DESCRIPTION: Used to read a string of characters from the default input port into a buffer. On entry, the first byte in the buffer indicates the maximum number of characters that can be placed in the buffer. Note that the buffer's size should be no less than this 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. All printable characters will be echoed to the default output port. The <CR> will not be echoed.

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 AØ contains the string buffer address;

```
PEA     (AØ)      Push buffer address
TRAP    #15       (May also invoke by SYSCALL
DC.W    3          macro (" SYSCALL .READSTR")
```

NOTES:

This routine allows the caller to dictate the maximum length of input up to 254 characters. If more than characters are entered, then the buffer input is truncated. Control character processing as described in section 2.2, Terminal Input/Output Control, is in effect.

5.2.5 .READLN Function .READLN

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

CODE: $ØØØ4

DESCRIPTION: Used to read a string of characters from the default input port. Characters are echoed to the default output port. A string consists of a count byte followed by the characters read from the input. The count byte indicates the number of characters 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 AØ points to a 256 byte buffer;

```
PEA       (AØ)       Long 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 section 2.2, Terminal Input/Output Control, is in effect.

5.2.6 .CHKBRK Function .CHKBRK

TRAP FUNCTION: .CHKBRK - Check for break-

CODE: $ØØØ5

DESCRIPTION: Returns "Zero" status in condition code register if break status detected at default input port.

ENTRY CONDITIONS:

No arguments or stack allocation required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Z flag set in CCR if break detected

EXAMPLE:

```
SYSCALL   .CHKBRK
BEQ       BREAK
```

5.2.7 .DSKRD, .DSKWR Function

.DSKRD
.DSKWR

TRAP FUNCTION: .DSKRD - Disk read function-

.DSKWR - Disk write function-

CODE: **$0010**

$0011

DESCRIPTION: These functions are used to read and write blocks of data to the specified disk device. Information about the data transfer is passed in a command packet which has been built somewhere in memory. The address of the packet is passed as an argument to the function. The same command packet format is used for **.DSKRD** and **.DSKWR**. These functions will automatically invoke **.DSKINIT** if the specified controller has not been previously initialized. They will also call **.DSKCFIG** if the specified device has not been previously configured. The command packet is eight words in length and is arranged as follows:

```
      F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
    +-------------------------------+-------------------------------+
$00 |        Controller LUN         |          Device LUN           |
    +-------------------------------+-------------------------------+
$02 |                          Status Word                          |
    +---------------------------------------------------------------+
$04 |                                                               |
    +-------------          Memory Address          ----------------+
$06 |                                                               |
    +---------------------------------------------------------------+
$08 |                      Block Number (Disk)                      |
    +-------------                or                ----------------+
$0A |                  File Number (Streamer tape)                  |
    +---------------------------------------------------------------+
$0C |                       Number of Blocks                        |
    +-------------------------------+-------------------------------+
$0E |           Flag Byte           |       Address Modifier        |
    +-------------------------------+-------------------------------+
```

Field descriptions:

Controller LUN	Logical Unit Number (LUN) of controller to use.
Device LUN	Logical Unit Number (LUN) of device to use.
Status Word	This status word will reflect the result of the operation. It will be zero if the command completed without errors. Refer to Appendix D for meanings of returned error codes.
Memory Address	Address of buffer in memory. On a disk read data will be written starting at this address. On a disk write data will be read starting at the address.
Block Number	For disk devices, this is the block number where the transfer will start. On a disk read data will be read starting at this block. On a disk write data will be written starting at this block.
File Number	For streamer tape devices, this is the file number where the transfer will start. This field is used if the **IFN** bit in the *Flag Byte* is cleared (refer to the Flag Byte description). On a disk read, data will be read starting at this file. On a disk write, data will be written starting at this file.
Number of Blocks	This field indicates the number of blocks to read from the disk (**.DSKRD**) or to write to the disk (**.DSKWR**). For streamer tape devices, the actual number of blocks transferred is returned in this field.
Flag Byte	The flag byte is used to specify variations of the same command, and to receive special status information. Bits Ø through 3 are used as command bits, bits 4 through 7 are used as status bits. For disk devices this field must be set to Ø. For streamer tape devices, the following bits are defined:

	Bit 7	File Mark flag. If 1, a file mark was detected at the end of the last operation.
	Bit 1	Ignore File Number flag. If Ø, the file number field is used to position the tape before any reads or writes are done. If 1, the file number field is ignored, and reads or writes start at the present tape position.
	Bit Ø	End Of File flag. If Ø, reads or writes are done until the specified block count is exhausted. If 1, reads are done until the count is exhausted or until a file mark is found. If 1, writes are terminated with a filemark.
Address Modifier	VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If non-zero, the specified value will be 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 will be written into memory as a result of .DSKRD function.
Data will be written to disk as a result of .DSKWR function.
Z(ero) = Set to 1 if no errors.

EXAMPLE:

If AØ, A1 point to packets formatted as specified above.

```
        PEA       (AØ)
        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
```

5.2.8 .DSKCFIG Function .DSKCFIG

TRAP FUNCTION: .DSKCFIG - Disk configure function-

CODE: $ØØ12

DESCRIPTION: This function allows the user to change the configuration of the specified device. It effectively performs an " IOT under program control". All the required parameters are passed in a command packet which has been built somewhere in memory. The address of the packet is passed as an argument to the function. This function is provided for use in special applications, since **.DSKCFIG** is invoked automatically the first time that a device is accessed by **.DSKRD**, **.DSKWR**, or **.DSKFMT**. The packet format is as follows:

```
         F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   Ø
       +--------------------------------+--------------------------------+
  $ØØ  |         Controller LUN         |           Device LUN           |
       +--------------------------------+--------------------------------+
  $Ø2  |                           Status Word                           |
       +-----------------------------------------------------------------+
  $Ø4  |                                                                 |
       +---------------          Memory Address           ---------------+
  $Ø6  |                                                                 |
       +-----------------------------------------------------------------+
  $Ø8  |                                                                 |
       +---------------                 Ø                 ---------------+
  $ØA  |                                                                 |
       +-----------------------------------------------------------------+
  $ØC  |                                Ø                                |
       +--------------------------------+--------------------------------+
  $ØE  |               Ø                |        Address Modifier        |
       +--------------------------------+--------------------------------+
```

Field descriptions:

Controller LUN   Logical Unit Number (LUN) of controller to use.

Device LUN   Logical Unit Number (LUN) of device to use.

Status Word	This status word will reflect the result of the operation. It will be zero if the command completed without errors. Refer to Appendix D for meanings of returned error codes.
Memory Address	Contains a pointer to a *Device Descriptor Packet* that contains the configuration information to be changed.
Address Modifier	VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If non-zero, the specified value will be used.

The Device Descriptor Packet is as follows:

```
        F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   Ø
      +-------------------------------+-------------------------------+
 $ØØ  |        Controller LUN         |          Device LUN           |
      +-------------------------------+-------------------------------+
 $Ø2  |                               Ø                               |
      +---------------------------------------------------------------+
 $Ø4  |                                                               |
      +--------              Parameters Mask                  --------+
 $Ø6  |                                                               |
      +---------------------------------------------------------------+
 $Ø8  |                                                               |
      +--------              Attributes Mask                  --------+
 $ØA  |                                                               |
      +---------------------------------------------------------------+
 $ØC  |                                                               |
      +--------              Attributes Flags                 --------+
 $ØE  |                                                               |
      +---------------------------------------------------------------+
 $1Ø  |                                                               |
      |                                                               |
      |                                                               |
      |                          Parameters                           |
      |                                                               |
      |                                                               |
      |                                                               |
      +---------------------------------------------------------------+
```

Most of the fields in the Device Descriptor Packet are equivalent to the fields defined in the **CFGA** *-Configuration Area* block, as described in Appendix B. In the field descriptions below, reference is made to the equivalent field in the **CFGA** whenever possible. For additional information on these fields, refer to Appendix B.

Controller LUN	Same as in command packet.
Device LUN	Same as in command packet.
Parameters Mask	Equivalent to the IOSPRM and IOSEPRM fields, with the *least significant* word equivalent to IOSPRM, and the *most significant* word equivalent to IOSEPRM.
Attributes Mask	Equivalent to the IOSATM and IOSEATM fields, with the *least significant* word equivalent to IOSATM, and the *most significant* word equivalent to IOSEATM.
Attributes Flags	Equivalent to the IOSATW and IOSEATW fields, with the *least significant* word equivalent to IOSATW, and the *most significant* word equivalent to IOSEATW.
Parameters	The parameters used for device reconfiguration are specified in this area. Most parameters have an *exact* CFGA equivalent. The following chart shows the field name, offset from start of packet, length, equivalent CFGA field, and short description of each field. Those parameters that do not have an *exact* equivalent are indicated with "*", and are explained after the chart.

Field Name	Offset (Bytes)	Length (Bytes)	CFGA Equiv.	Description
P_DDS*	$10	1	---	Device descriptor size
P_DSR	$11	1	IOSSR	Step rate
P_DSS*	$12	1	IOSPSM	Sector size (encoded)
P_DBS*	$13	1	IOSREC	Block size (encoded)
P_DST*	$14	2	IOSSPT	Sectors/track
P_DIF	$16	1	IOSILV	Interleave factor
P_DSO	$17	1	IOSSOF	Spiral Offset
P_DSH*	$18	1	IOSSHD	Starting head
P_DNH	$19	1	IOSHDS	Number of heads
P_DNCYL	$1A	2	IOSTRK	Number of cylinders
P_DPCYL	$1C	2	IOSPCOM	Precompensation cylinder
P_DRWCYL	$1E	2	IOSRWCC	Reduced write current cylinder
P_DECCB	$20	2	IOSECC	ECC data burst length
P_DGAP1	$22	1	IOSGPB1	Gap 1 size
P_DGAP2	$23	1	IOSGPB2	Gap 2 size
P_DGAP3	$24	1	IOSGPB3	Gap 3 size
P_DGAP4	$25	1	IOSGPB4	Gap 4 size
P_DSSC	$26	1	IOSSSC	Spare sectors count
P_DRUNIT	$27	1	IOSRUNIT	Reserved area units
P_DRCALT	$28	2	IOSRSVC1	Reserved count for alternates
P_DRCCTR	$2A	2	IOSRSVC2	Reserved count for controller

Chart Notes:

P_DDS This field is for internal use only, and does not have an equivalent CFGA field. Should be set to 0.

P_DSS This is a one byte encoded field, whereas the IOSPSM field is a two byte unencoded field containing the actual number of bytes per sector. The P_DSS field is encoded as follows:

$00	128 bytes
$01	256 bytes
$02	512 bytes
$03	1024 bytes
$04-$FF	Reserved encodings.

P_DBS This is a one byte encoded field, whereas the IOSREC field is a two byte unencoded field containing the actual number of bytes per record (block). The P_DBS field is encoded as follows:

\$ØØ	128 bytes
\$Ø1	256 bytes
\$Ø2	512 bytes
\$Ø3	1Ø24 bytes
\$Ø4-\$FF	Reserved encodings.

P_DSH This is a one byte field, whereas the IOSSHD field is two bytes. This field is equivalent to the *least significant* byte of IOSSHD.

P_DST This is two bytes, whereas the IOSSPT field is a one byte field.

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.
The device configuration will be changed.
Z(ero) = Set to 1 if no errors.

EXAMPLE:

If AØ points to a packet formatted as specified above.

```
        PEA.L     (AØ)           Load command packet
        SYSCALL   .DSKCFIG       reconfigure device
        BNE        ERROR         Branch if error
        "
        "
        "
ERROR   xxxxx     xxx            Handle error
        xxxxx     xxx
```

5.2.9 .DSKFMT Function .DSKFMT

TRAP FUNCTION: .DSKFMT - Disk Format Function

CODE: $0014

DESCRIPTION: This function allows the user to send a format command to the specified device. The parameters required for the command are passed in a command packet which has been built somewhere in memory. The address of the packet is passed as an argument to the function. The format of the packet is as follows:

```
        F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
      +-------------------------------+-------------------------------+
$00   |         Controller LUN        |           Device LUN          |
      +-------------------------------+-------------------------------+
$02   |                          Status Word                          |
      +---------------------------------------------------------------+
$04   |                                                               |
      +---------------         Memory Address         ---------------+
$06   |                                                               |
      +---------------------------------------------------------------+
$08   |                                                               |
      +---------------      Block Number (Disk)       ---------------+
$0A   |                                                               |
      +---------------------------------------------------------------+
$0C   |                               0                               |
      +-------------------------------+-------------------------------+
$0E   |           Flag Byte           |        Address Modifier       |
      +-------------------------------+-------------------------------+
```

Field descriptions:

Controller LUN	Logical Unit Number (LUN) of controller to use.
Device LUN	Logical Unit Number (LUN) of device to use.
Status Word	This status word will reflect the result of the operation. It will be zero if the command completed without errors. Refer to Appendix D for meanings of returned error codes.

Memory Address — Address of buffer in memory. On a disk read data will be written starting at this address. On a disk write data will be read starting at the address.

Block Number — For disk devices, when doing a format track, the track that contains this block number is formatted. This field is ignored for streamer tape devices.

Flag Byte — Contains additional control information. Bit Ø is interpreted as follows for disk devices:

- If Ø, it indicates a "Format Track" operation. The track that contains the specified block is formatted.
- If 1, it indicates a "Format Disk" operation. All the tracks on the disk will be formatted.

For streamer tapes, bit Ø is interpreted as follows:

- If Ø, it selects a "Retension Tape" operation. This will rewind the tape to BOT, advance the tape without interruptions to EOT, and then rewind it back to BOT. Tape retension is recommended by cartridge tape suppliers before writing or reading data when a cartridge has been subjected to a change in environment or a physical shock, has been stored for a prolonged period of time or at extreme temperature, or has been previously used in a start/stop mode.
- If 1, it selects an "Erase Tape" operation. This will completely clear the tape of previous data and at the same time will retension the tape.

Address Modifier — VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If non-zero, the specified value will be 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.
Z(ero) = Set to 1 if no errors.

EXAMPLE:

If AØ points to a packet formatted as specified above.

```
        PEA.L     (AØ)      Load command packet
        SYSCALL   .DSKFMT   format device
        BNE       ERROR     If errors, branch
         *
         *
         *
ERROR   xxxxx      xxx           Handle error
        xxxxx      xxx
```

5.2.10 .DSKCTRL Function .DSKCTRL

TRAP FUNCTION: .DSKCTRL - Disk control function-

CODE: $0015

DESCRIPTION: This function is used to implement any special device control functions that can not be accommodated easily with any of the other disk functions. At the present, the only defined function is SEND packet, which allows the user to send a packet in the specific format of the controller. The required parameters are passed in a command packet which has been built somewhere in memory. The address of the packet is passed as an argument to the function.

The packet is as follows:

```
    F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
+-------------------------------+-------------------------------+
|        Controller LUN         |          Device LUN           |
+-------------------------------+-------------------------------+
|                          Status Word                          |
+---------------------------------------------------------------+
|                                                               |
+---------------          Memory Address         ---------------+
|                                                               |
+---------------------------------------------------------------+
|                                                               |
+---------------                0                ---------------+
|                                                               |
+---------------------------------------------------------------+
|                               0                               |
+-------------------------------+-------------------------------+
|               0               |       Address Modifier        |
+-------------------------------+-------------------------------+
```

Field descriptions:

Controller LUN	Logical Unit Number (LUN) of controller to use.
Device LUN	Logical Unit Number (LUN) of device to use.

Status Word — This status word will reflect the result of the operation. It will be zero if the command completed without errors. Refer to Appendix D for meanings of returned error codes.

Memory Address — Contains a pointer to the controller packet to send. Note that the controller packet to send (as opposed to the command packet) is controller and device dependent. Information about this packet should be found in the user's manual for the controller and device being accessed.

Address Modifier — VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If non-zero, the specified value will be 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.
Additional side effects depend on the packet sent to the controller.
Z(ero) = Set to 1 if no errors.

EXAMPLE:

If A1 points to a packet formatted as specified above.

```
        PEA.L     (A1)      Load command packet
        SYSCALL   .DSKCTRL  invoke control function
        BNE       ERROR     If errors, branch
        "
        "
        "
ERROR   xxxxx     xxx            Handle error
        xxxxx     xxx
```

5.2.11 .OUTCHR Function .OUTCHR

TRAP FUNCTION: .OUTCHR - Output character routine-

CODE: $ØØ2Ø

DESCRIPTION: This function will output 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    DØ,-(SP)     Send character in DØ
SYSCALL   .OUTCHR      To default output port
```

5.2.12 .OUTSTR, .OUTLN Function

.OUTSTR
.OUTLN

TRAP FUNCTION: .OUTSTR - Output string to default output port-

.OUTLN - Output string along with CR/LF-

CODE: **$ØØ21**

$ØØ22

DESCRIPTION: **.OUTSTR** will output a string of characters to the default output port. **.OUTLN** will output 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 AØ = start of string
   A1 = end of string+1

        MOVEM.L   AØ/A1,-(SP)  Load pointers to string
        SYSCALL   .OUTSTR      And print it
```

5.2.13 .WRITE, .WRITELN Function

.WRITE
.WRITELN

TRAP FUNCTION: .WRITE - Output string with no CR of LF-

.WRITELN - Output string with CR and LF-

CODE: $ØØ23

$ØØ24

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 follow:
SP ==> Address of string <long>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack

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.

5.2.14 .PCRLF Function .PCRLF

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

CODE: $ØØ26

DESCRIPTION: **.PCRLF** will send a < CR> < LF> sequence to the default output port.

ENTRY CONDITIONS:

No arguments or stack allocation required.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

None

EXAMPLE:

```
SYSCALL   .PCRLF       Output CRLF
```

5.2.15 .ERASLN Function .ERASLN

TRAP FUNCTION: .ERASLN - Erase line-

CODE: $ØØ27

DESCRIPTION: **.ERASLN** 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
```

5.2.16 .WRITD, .WRITDLN Function

.WRITD
.WRITDLN

TRAP FUNCTION: .WRITD - Output string with data-

.WRITDLN - Output string with data and CRLF-

CODE: $ØØ28

$ØØ25

DESCRIPTION: These trap functions take advantage of the monitor I/O routine which outputs a user string containing embedded variable fields. The user passes the starting address of the string and the address of a data stack containing the data which will be inserted into the string. The output goes to the default output port.

ENTRY CONDITIONS:

Eight bytes of parameter positioned in stack as follow:

```
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>
```

EXIT CONDITIONS DIFFERENT FROM ENTRY:

```
SP ==> Top of stack
```

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)+          De-allocate 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 below).

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, i.e., "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 will be read from the data stack to be displayed.

4. The data stack is not destroyed by this routine. If it is necessary for the space in the data stack to be deallocated, it must be done by the calling routine, as shown in the above example.

5.2.17 .SNDBRK Function .SNDBRK

TRAP FUNCTION: .SNDBRK - Send break-

CODE: $ØØ29

DESCRIPTION: Used to send break to default output port(s).

ENTRY CONDITIONS:

No arguments or stack allocation required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Each serial port specified by current default port list will have sent "break".

EXAMPLE:

```
SYSCALL   .SNDBRK
```

5.2.18 .TM_INI Function .TM_INI

TRAP FUNCTION: .TM_INI - Timer initialization routine-

CODE: $ØØ4Ø

DESCRIPTION: This routine initializes the on-board timers, and also calculates a calibration factor used by the other timer functions. This routine should be used the first time that the timer functions are used.

ENTRY CONDITIONS:

No arguments required.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

TM.CAL1(A5) loaded with calibration factor.
Timers are configured for 24-bit operation.

EXAMPLE:

```
SYSCALL   .TM_INI    Initialize timer
```

5.2.19 .TM_STRØ Function .TM_STRØ

TRAP FUNCTION: .TM_STRØ - Start timer at T=Ø-

CODE: $ØØ41

DESCRIPTION: This routine will first reset the timer to Ø and then it will start it.

ENTRY CONDITIONS:

No arguments required.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Timer is started.

EXAMPLE:

```
SYSCALL   .TM_STRØ
```

5.2.20 .TM_RD Function .TM_RD

TRAP FUNCTION: .TM_RD - Timer read function-

CODE: $0042

DESCRIPTION: This routine is used to read the value of the timer (microseconds).

ENTRY CONDITIONS:

SP ==> Space for result <long>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Time in microseconds <long>

EXAMPLE:

```
SUBQ.L    #4,A7       Allocate space for result
SYSCALL   .TM_RD      Read timer
MOVE.L    (SP)+,D0    Load time in microseconds
```

5.2.21 .DELAY Function .DELAY

TRAP FUNCTION: .DELAY - Timer delay function-

CODE: $0043

DESCRIPTION: **.DELAY** is used to generate accurate timing delays that are independent of the processor frequency and instruction execution rate. This function uses the on-board timer for operation. The user specifies the desired delay count in microseconds. **.DELAY** will return to the caller after the specified delay count is exhausted. The on-board timer has to be initialized once before this function is called by invoking the **.TM_INI** trap function.

ENTRY CONDITIONS:

SP ==> Delay time in microseconds <long>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack

EXAMPLE:

```
SYSCALL   .TM_INI    Initialize timer
PEA.L     &15000000  Load a 15 second delay
SYSCALL   .DELAY
 *
 *
 *
PEA.L     &50000     Load a 50 millisecond delay
SYSCALL   .DELAY
```

5.2.22 .REDIR Function .REDIR

TRAP FUNCTION: .REDIR - Redirect I/O function-

CODE: $0060

DESCRIPTION: This routine is used to select an I/O port and at the same time invoke a particular I/O function. The invoked I/O function will read or write to the selected port.

ENTRY CONDITIONS:

SP ==>	Port	<word>
	I/O function to call	<word>
	Parameters of I/O function	<size specified by function>
	Space for results	<size specified by function>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==>	Result	<size specified by function>

EXAMPLE:

None

NOTES:

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

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

Then the parameters required by **.REDIR** should be pushed and the call made to **.REDIR**.

```
MOVE.W    #.INCHR,-(SP) Load function code
MOVE.W    #1,-(SP)      Load port number
SYSCALL   .REDIR        Redirect I/O function
```

Finally, the results should be popped from the stack:

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

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

5.2.23 .REDIR_I, .REDIR_O Function

.REDIR_I
.REDIR_O

TRAP FUNCTION: .REDIR_I - Redirect input-

.REDIR_O - Redirect output-

CODE: $ØØ61

$ØØ62

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 will remain 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 new default
```

5.2.24 .RETURN Function .RETURN

TRAP FUNCTION: .RETURN - Return to 135Bug-

CODE: $ØØ63

DESCRIPTION: This function is used to return control to 135Bug 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 135Bug.

ENTRY CONDITIONS:

No arguments required.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Control is returned to 135Bug.

EXAMPLE:

```
SYSCALL   .RETURN       Return to 135Bug
```

5.2.25 .BINDEC Function .BINDEC

TRAP FUNCTION: .BINDEC - Used to calculate the BCD equivalent of the binary number specified-

CODE: **$ØØ64**

DESCRIPTION: This function takes a 32-bit unsigned binary number and changes it to an equivalent BCD (Binary Coded Decimal 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:

```
SUB.L     #8,A7         Allocate space for result
MOVE.L    DØ,-(SP)      Load hex number
SYSCALL   .BINDEC       Call .BINDEC
MOVEM.L   (SP)+,D1/D2   Load result
```

5.2.26 .CHANGEV Function .CHANGEV

TRAP FUNCTION: .CHANGEV - Parse value, assign to variable-

CODE: $ØØ67

DESCRIPTION: Attempt to parse value in user specified buffer. If user's buffer is empty, prompt user for new value, otherwise update integer offset into buffer to skip "value". Display new value and assign to variable unless user's input is an empty string.

ENTRY CONDITIONS:

```
SP ==> Address of 32-bit offset into user's buffer
       Address of user's buffer (pointer/count format string)
       Address of 32-bit integer variable to " change"
       Address of string to use in prompting and displaying value
```

EXIT CONDITIONS DIFFERENT FROM ENTRY:

```
SP ==> Top of stack
```

EXAMPLE:

```
PROMPT    DC.B     $E,'COUNT = |1Ø,8|'
GETCOUNT  PEA      PROMPT(PC)   Point to prompt string
          PEA      COUNT        Point to variable to change
          PEA      POINT        Point to offset into buffer
          PEA      BUFFER       Point to buffer
          SYSCALL  .CHANGEV     Make the system call
          RTS                   COUNT changed, return
```

If the above code was called with BUFFER containing "1 3" in pointer/count format and POINT containing 2 (longword), then COUNT would be assigned the value 3, and POINT would contain 4 (pointing to first character past "3"). Note that POINT is the offset from the start address of the buffer (not the address of the first character in the buffer!) to the next character to process. In this case, a value of 2 in POINT indicates that the space between "1" and "3" is the next character to be processed. After calling **.CHANGEV**, the screen would display the following line:

```
COUNT = 3
```

If the above code was called again, nothing could be parsed from BUFFER, so a prompt would be issued. For purpose of example, the string "5" is entered in response to the prompt.

```
COUNT = 3? 5 <CR>
COUNT = 5
```

If in the previous example nothing had been entered at the prompt, COUNT would retain its prior value.

```
COUNT = 3? <CR>
COUNT = 3
```

5.2.27 .STRCMP Function .STRCMP

TRAP FUNCTION: .STRCMP - Compare two strings (pointer/count)-

CODE: $ØØ68

DESCRIPTION: Comparison for equality is made and a boolean flag is returned to the caller. The flag will be $ØØ if the strings are not identical, otherwise it will be $FF.

ENTRY CONDITIONS:

```
SP ==> Address of string 1
       Address of string 2
       Three bytes (unused)
       Byte to receive string comparison result
```

EXIT CONDITIONS DIFFERENT FROM ENTRY:

```
SP ==> Three bytes (unused)
       Byte to receive string comparison result
```

EXAMPLE:

If A1 and A2 contain addresses of the two strings.

```
SUBQ.L    #4,SP       Allocate longword to receive result
PEA       (A1)        Push address of one string
PEA       (A2)        Push address of the other string
SYSCALL   .STRCMP     Compare the strings
MOVE.L    (SP)+,DØ    Pop boolean flag into data register
TST.B     DØ          Check boolean flag
BNE       ARE_SAME    Branch if strings are identical
```

5.2.28 .MULU32 Function .MULU32

TRAP FUNCTION: .MULU32 - Unsigned 32 x 32 bit multiply-

CODE: $ØØ69

DESCRIPTION: Two 32-bit unsigned integers are multiplied and the product is returned on the stack as a 32-bit unsigned integer. No overflow checking is performed.

ENTRY CONDITIONS:

```
SP ==> 32-bit multiplier
       32-bit multiplicand
       32-bit space for result
```

EXIT CONDITIONS DIFFERENT FROM ENTRY:

```
SP ==> 32-bit product (result from multiplication)
```

EXAMPLE:

Multiply DØ by D1, load result into D2.

```
SUBQ.L    #4,SP       Allocate space for result
MOVE.L    DØ,-(SP)    Push multiplicand
MOVE.L    D1,-(SP)    Push multiplier
SYSCALL   .MULU32     Multiply DØ by D1
MOVE.L    (SP)+,D2    Get product
```

5.2.29 .DIVU32 Function .DIVU32

TRAP FUNCTION: .DIVU32 - Unsigned 32 x 32 bit divide-

CODE: $ØØ6A

DESCRIPTION: Unsigned division is performed on two 32-bit integers and the quotient is returned on the stack as a 32-bit unsigned integer. The case of division by zero is handled by returning the maximum unsigned value $FFFFFFFF.

ENTRY CONDITIONS:

```
SP ==> 32-bit divisor              (value to divide by)
       32-bit dividend             (value to divide)
       32-bit space for result
```

EXIT CONDITIONS DIFFERENT FROM ENTRY:

```
SP ==> 32-bit quotient (result from division)
```

EXAMPLE:

Divide DØ by D1, load result into D2.

```
SUBQ.L    #4,SP      Allocate space for result
MOVE.L    DØ,-(SP)   Push dividend
MOVE.L    D1,-(SP)   Push divisor
SYSCALL   .DIVU32    Divide DØ by D1
MOVE.L    (SP)+,D2   Get quotient
```

APPENDIX A - 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:

S-RECORD FIELD DESCRIPTIONS

Field	Printable Characters	Contents
type	2	S-records type -- SØ, S1, etc.
record length	2	The count of the character pairs in the record, excluding type and records 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	Ø-2n	From Ø 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).

S-RECORD FIELD DESCRIPTIONS (cont.)

Field	Printable Characters	Contents
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 records length, address, and the code/data fields.

Each record may be terminated with a CR/LN/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 records 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 the program must be consulted. 1ØxBug supports SØ, S1, S2, S3, S7, S8, and S9 records.

An S-Record format module may contain S-records of the following types:

SØ 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 records. The address field is normally zeros.

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 passed. There is no code/data field.

S8 A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is 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 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 files 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 allow 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
S113003000144ED492
S9030000FC
```

The module consists of one SØ record, four S1 records, and an S9 record.

The SØ record is comprised of the following character pairs:

SØ S-record type SØ, indicating that it is a header record.

Ø6 Hexadecimal Ø6 (decimal 6), indicating that six character pairs (or ASCII bytes) follow.

ØØ
ØØ Four-character 2-byte address field, zeros in this example.

48
44 ASCII H, D and R - "HDR".
52

1B The checksum.

The first S1 record is explained as follows:

S1 S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.

13 Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.

ØØ Four-character 2-byte address field; hexadecimal address ØØØØ,
ØØ 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
37C	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 also each contain $13 (19) character pairs and are ended with checksums 13 and 52 respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The S9 record is explained as follows:

S9 S-record type S9, indicating that it is a termination record.

03 Hexadecimal 03, indicating that three character pairs (3 bytes) follow.

00
00 The address field, zeros.

FC The checksum of the S9 record.

Each printable character in an S-record is enclosed 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			
S		1		1		3	
5	3	3	1	3	1	3	3
Ø1Ø1	ØØ11	ØØ11	ØØØ1	ØØ11	ØØØ1	ØØ11	ØØ11

Address							
Ø		Ø		Ø		Ø	
3	Ø	3	Ø	3	Ø	3	Ø
ØØ11	ØØØØ	ØØ11	ØØØØ	ØØ11	ØØØØ	ØØ11	ØØØØ

Code/Data								
2		8		5		F		
3	2	3	8	3	5	4	6	....
ØØ11	ØØ1Ø	ØØ11	1ØØØ	ØØ11	Ø1Ø1	Ø1ØØ	Ø11Ø	....

Checksum			
2		A	
3	2	4	1
ØØ11	ØØ1Ø	Ø1ØØ	ØØØ1

APPENDIX B - INFORMATION USED BY BO/BH COMMANDS

VOLUME ID BLOCK (VID) -Always at Block Ø-

Label	Offset	Length (bytes)	Contents
VIDOSS	$14(2Ø)	4	Starting block number of operating system.
VIDOSL	$18(24)	2	Operating system length in blocks.
VIDOSA	$1E(3Ø)	4	Starting memory location to load operating system.
VIDCAS	$9Ø(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 (CFGA)

Label	Offset	Length (bytes)	Contents
IOSATM	$Ø4(4)	2	Attributes mask.
IOSPRM	$Ø6(6)	2	Parameters mask.
IOSATW	$Ø8(8)	2	Attributes word.
IOSREC	$ØA(1Ø)	2	Record (block) size in bytes.
IOSSPT	$18(24)	1	Sectors/Track.
IOSHDS	$19(25)	1	Number of heads on drive.
IOSTRK	$1A(26)	2	Number of cylinders.
IOSILV	$1C(28)	1	Interleave factor on media.
IOSSOF	$1D(29)	1	Spiral offset.
IOSPSM	$1E(3Ø)	2	Physical sector size of media in bytes.
IOSSHD	$2Ø(32)	2	Starting head number.
IOSPCOM	$24(36)	2	Precompensation cylinder.
IOSSR	$27(39)	1	Stepping rate code.
IOSRWCC	$28(4Ø)	2	Reduced write current cylinder number.
IOSECC	$2A(42)	2	ECC data burst length.
IOSEATM	$2C(44)	2	Extended attributes mask.
IOSEPRM	$2E(46)	2	Extended parameters mask.
IOSEATW	$3Ø(48)	2	Extended attributes word.
IOSGPB1	$32(5Ø)	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.
IOSRUNIT	$37(55)	1	Reserved Area Units.
IOSRSVC1	$38(56)	2	Reserved count 1.
IOSRSVC2	$3A(58)	2	Reserved count 2.

IOSATM and IOSEATM

A "1" in a particular bit position indicates that the corresponding attribute from the attributes (or extended attributes) word should be used to update the configuration. A "Ø" in a bit position indicates that the current attribute should be retained.

IOSATM ATTRIBUTE MASK BIT DEFINITIONS

Label	Bit Position	Description
IOADDEN	Ø	Data density.
IOATDEN	1	Tranck density.
IOADSIDE	2	Single/double sided.
IOAFRMT	3	Floppy disk format.
IOARDISC	4	Disk type.
IOADDEND	5	Drive data density.
IOATDEND	6	Drive track density.
IOARIBS	7	Embedded servo drive seek.
IOADPCOM	8	Post-read/pre-write precompensation.
IOASIZE	9	Floppy disk size.
IOATKZD	13	Track zero data density.

At the present all IOSEATM bits are undefined and should be set to Ø.

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 "Ø" 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	Ø	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	1Ø	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	Ø	Gap byte 1.
IOAGPB2	1	Gap byte 2.
IOAGPB3	2	Gap byte 3.
IOAGPB4	3	Gap byte 4.
IOASSC	4	Spare sectors count.
IOARUNIT	5	Reserved area units.
IOARVC1	6	Reserved count 1.
IOARVC2	7	Reserved count 2.

IOSATW and IOSEATW

Contains various flags that specify characteristics of the media and drive.

IOSATW BIT DEFINITIONS

Bit Number	Description
Bit Ø	Data density: Ø = Single density (FM encoding) 1 = Double density (MFM encoding)
Bit 1	Track density: Ø = Single density (48 TPI) 1 = Double density (96 TPI)
Bit 2	Number of sides: Ø = Single sided floppy 1 = Double sided floppy
Bit 3	Floppy disk format: Ø = Motorola format 1 to N on side Ø N+1 to 2N on side 1 1 = Standard IBM format 1 to N on both sides
Bit 4	Disk type: Ø = Floppy disk 1 = Hard disk
Bit 5	Drive data density: Ø = Single density (FM encoding) 1 = Double density (MFM encoding)
Bit 6	Drive track density: Ø = Single density 1 = Double density
Bit 7	Embedded servo drive: Ø = Do not seek on head switch 1 = Seek on head switch
Bit 8	Post-read/pre-write precompensation: Ø = Pre-write 1 = Post-read
Bit 9	Floppy disk size: Ø = 5-1/4" floppy 1 = 8" floppy
Bit 13	Track zero density: Ø = Single density (FM encoding) 1 = Same as remaining tracks
Unused bits	All unused bits must be set to Ø.

At the present all IOSEATW bits are undefined and should be set to Ø.

PARAMETER FIELD DEFINITIONS

Parameter	Description
Record (Block) size	Number of bytes per record (block). Must be an integer multiple of the physical sector size.
Sector/track	Number of sectors per track in bytes.
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 Ø. 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 will begin.

PARAMETER FIELD DEFINITIONS (cont.)

Parameter	Description

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 a follows:

Step Rate Code	Winchester Hard Disks	5-1/4" Floppy	8" 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

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.

PARAMETER FIELD DEFINITIONS (cont.)

Parameter	Description
Spare sectors count	This field contains the number of sectors per track allocated as spare sectors. These sectors will only be used as replacements for bad sectors on the disk.
Reserved Area Units	This field specifies the units used for the next two fields (IOSRSVC1 and IOSRSVC2). If zero the units are in *tracks*, if 1 the units are in *cylinders*.
Reserved Count 1	This field specifies the number of tracks (IOSRUNIT=Ø), or the number of cylinders (IOSRUNIT=1) reserved for the alternate mapping area on the disk.
Reserved Count 2	This field specifies the number of tracks (IOSRUNIT=Ø), or the number of cylinders (IOSRUNIT=1) reserved for use by the controller.

APPENDIX C - DISK CONTROLLER DATA

Disk Controller Modules Supported

The following VMEbus disk/tape controller modules are supported by 135Bug:

CONTROLLER TYPE	CLUN #1 / ADDR #1	CLUN #2 / ADDR #2
MVME319 - SCSI/Floppy/Tape Controller	$ØØ	$Ø7
	$FFFFØØØØ	$FFFFØ2ØØ
MVME32Ø - Winchester/Floppy Controller	$ØØ	$Ø6
	$FFFFBØØØ	$FFFFACØØ
MVME321 - Winchester/Floppy Controller	$ØØ	$Ø1
	$FFFFØ5ØØ	$FFFFØ6ØØ
MVME323 - ESDI Controller	$Ø8	$Ø9
	$FFFFAØØØ	$FFFFA2ØØ
MVME327 - SCSI Controller	$ØØ-Ø7	$ØØ-Ø7
	$FFFFØ6ØØ	$FFFFØ7ØØ
MVME35Ø - Streamer Tape Controller	$Ø4	$Ø5
	$FFFF5ØØØ	$FFFF51ØØ
MVME36Ø - SMD Controller	$Ø2	$Ø3
	$FFFFØCØØ	$FFFFØEØØ

Disk Controller Default Configurations

```
Controller LUN Ø
Controller Type    : MVME319
Controller Address: $FFFFØØØØ
Number of Devices : 8
Devices            : DLUN Ø = 4Ø Megabyte Winchester hard drive (see note)
                   : DLUN 1 = 4Ø Megabyte Winchester hard drive (see note)
                   : DLUN 2 = 4Ø Megabyte Winchester hard drive (see note)
                   : DLUN 3 = 4Ø Megabyte Winchester hard drive (see note)
                   : DLUN 4 = 8" DS/DD Motorola format floppy drive
                   : DLUN 5 = 8" DS/DD Motorola format floppy drive
                   : DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive
```

```
Controller LUN 7
Controller Type    : MVME319
Controller Address: $FFFFØ2ØØ
Number of Devices : 8
Devices            : DLUN Ø = 4Ø Megabyte Winchester hard drive (see note)
                   : DLUN 1 = 4Ø Megabyte Winchester hard drive (see note)
                   : DLUN 2 = 4Ø Megabyte Winchester hard drive (see note)
                   : DLUN 3 = 4Ø Megabyte Winchester hard drive (see note)
                   : DLUN 4 = 8" DS/DD Motorola format floppy drive
                   : DLUN 5 = 8" DS/DD Motorola format floppy drive
                   : DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive
```

NOTE: Devices Ø through 3 are accessed via the SCSI interface on the MVME319. An ADAPTEC ACB-4ØØØ Winchester Disk Controller module is required to interface between the SCSI and the disk drive. Refer to the MVME319 User's Manual for further information.

```
Controller LUN 0
Controller Type    : MVME320
Controller Address: $FFFFB000
Number of Devices : 4
Devices            : DLUN 0 = 40 Megabyte Winchester hard disk
                   : DLUN 1 = 40 Megabyte Winchester hard disk
                   : DLUN 2 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 3 = 5-1/4" DS/DD 96 TPI floppy drive

Controller LUN 6
Controller Type    : MVME320
Controller Address: $FFFFAC00
Number of Devices : 4
Devices            : DLUN 0 = 40 Megabyte Winchester hard disk
                   : DLUN 1 = 40 Megabyte Winchester hard disk
                   : DLUN 2 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 3 = 5-1/4" DS/DD 96 TPI floppy drive

Controller LUN 0
Controller Type    : MVME321
Controller Address: $FFFF0500
Number of Devices : 8
Devices            : DLUN 0 = 40 Megabyte Winchester hard disk
                   : DLUN 1 = 40 Megabyte Winchester hard disk
                   : DLUN 2 = 40 Megabyte Winchester hard disk
                   : DLUN 3 = 40 Megabyte Winchester hard disk
                   : DLUN 4 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 5 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive
```

```
Controller LUN 1
Controller Type    : MVME321
Controller Address: $FFFFØ6ØØ
Number of Devices : 8
Devices            : DLUN Ø = 4Ø Megabyte Winchester hard disk
                   : DLUN 1 = 4Ø Megabyte Winchester hard disk
                   : DLUN 2 = 4Ø Megabyte Winchester hard disk
                   : DLUN 3 = 4Ø Megabyte Winchester hard disk
                   : DLUN 4 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 5 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive


Controller LUN 8
Controller Type    : MVME323
Controller Address: $FFFFAØØØ
Number of Devices : 4
Devices            : DLUN Ø = CDC WREN III 182 Megabyte ESDI hard disk
                                                   (512 byte sectors)
                   : DLUN 1 = CDC WREN III 182 Megabyte ESDI hard disk
                                                   (512 byte sectors)
                   : DLUN 2 = CDC WREN III 182 Megabyte ESDI hard disk
                                                   (512 byte sectors)
                   : DLUN 3 = CDC WREN III 182 Megabyte ESDI hard disk
                                                   (512 byte sectors)


Controller LUN 9
Controller Type    : MVME323
Controller Address: $FFFFA2ØØ
Number of Devices : 4
Devices            : DLUN Ø = CDC WREN III 182 Megabyte ESDI hard disk
                                                   (512 byte sectors)
                   : DLUN 1 = CDC WREN III 182 Megabyte ESDI hard disk
                                                   (512 byte sectors)
                   : DLUN 2 = CDC WREN III 182 Megabyte ESDI hard disk
                                                   (512 byte sectors)
                   : DLUN 3 = CDC WREN III 182 Megabyte ESDI hard disk
                                                   (512 byte sectors)
```

```
Controller LUN Ø
Controller Type    : MVME327
Controller Address: $FFFFØ6ØØ
Number of Devices : 1
Devices           : DLUN Ø = CDC WREN III 155 Megabyte SCSI hard disk
                                                  (512 byte sectors)

Controller LUN 1
Controller Type    : MVME327
Controller Address: $FFFFØ6ØØ
Number of Devices : 1
Devices           : DLUN Ø = MICROPOLIS 15Ø Megabyte SCSI hard disk
                                                (512 byte sectors)

Controller LUN 2
Controller Type    : MVME327
Controller Address: $FFFFØ6ØØ
Number of Devices : 1
Devices           : DLUN Ø = CDC WREN IV 3ØØ Megabyte SCSI hard disk
                                                  (512 byte sectors)

Controller LUN 3
Controller Type    : MVME327
Controller Address: $FFFFØ6ØØ
Number of Devices : 1
Devices           : DLUN Ø = SEAGATE 8Ø Megabyte SCSI hard disk
                                            (512 byte sectors)

Controller LUN 4
Controller Type    : MVME327
Controller Address: $FFFFØ6ØØ
Number of Devices : 1
Devices           : DLUN Ø = ARCHIVE VIPER Streaming Tape Drive
```

```
Controller LUN 5
Controller Type   : MVME327
Controller Address: $FFFFØ6ØØ
Number of Devices : 1
Devices           : DLUN Ø = ARCHIVE VIPER Streaming Tape Drive


Controller LUN 7
Controller Type   : MVME327
Controller Address: $FFFFØ6ØØ
Number of Devices : 1
Devices           : DLUN Ø = 5-1/4" DS/DD 96 TPI floppy drive
                  : DLUN 1 = 5-1/4" DS/DD 96 TPI floppy drive


Controller LUN Ø
Controller Type   : MVME327
Controller Address: $FFFFØ7ØØ
Number of Devices : 1
Devices           : DLUN Ø = CDC WREN III 155 Megabyte SCSI hard disk
                                                 (256 byte sectors)


Controller LUN 1
Controller Type   : MVME327
Controller Address: $FFFFØ7ØØ
Number of Devices : 1
Devices           : DLUN Ø = MICROPOLIS 15Ø Megabyte SCSI hard disk
                                               (256 byte sectors)


Controller LUN 2
Controller Type   : MVME327
Controller Address: $FFFFØ7ØØ
Number of Devices : 1
Devices           : DLUN Ø = CDC WREN IV 3ØØ Megabyte SCSI hard disk
                                                 (256 byte sectors)
```

```
Controller LUN 3
Controller Type    : MVME327
Controller Address: $FFFFØ7ØØ
Number of Devices : 1
Devices            : DLUN Ø = SEAGATE 8Ø Megabyte SCSI hard disk
                                              (256 byte sectors)


Controller LUN 4
Controller Type    : MVME327
Controller Address: $FFFFØ7ØØ
Number of Devices : 1
Devices            : DLUN Ø = ARCHIVE VIPER Streaming Tape Drive


Controller LUN 5
Controller Type    : MVME327
Controller Address: $FFFFØ7ØØ
Number of Devices : 1
Devices            : DLUN Ø = ARCHIVE VIPER Streaming Tape Drive


Controller LUN 7
Controller Type    : MVME327
Controller Address: $FFFFØ7ØØ
Number of Devices : 1
Devices            : DLUN Ø = 5-1/4" DS/DD 96 TPI floppy drive
                   : DLUN 1 = 5-1/4" DS/DD 96 TPI floppy drive


Controller LUN 4
Controller Type    : MVME35Ø
Controller Address: $FFFF5ØØØ
Number of Devices : 1
Devices            : DLUN Ø = QIC-Ø2 Streaming Tape Drive
```

```
Controller LUN 5
Controller Type    : MVME350
Controller Address: $FFFF5100
Number of Devices : 1
Devices            : DLUN 0 = QIC-02 Streaming Tape Drive


Controller LUN 2
Controller Type    : MVME360
Controller Address: $FFFF0C00
Number of Devices : 4
Devices            : DLUN 0 = 2333K Fuji SMD drive (512-byte sectors)
                   : DLUN 1 = null device
                   : DLUN 2 = 2322K Fuji SMD drive (512-byte sectors)
                   : DLUN 3 = null device


Controller LUN 3
Controller Type    : MVME360
Controller Address: $FFFF0E00
Number of Devices : 4
Devices            : DLUN 0 = 2322K Fuji SMD drive (256-byte sectors)
                   : DLUN 1 = null device
                   : DLUN 2 = 80 Megabyte Fixed CMD drive
                   : DLUN 3 = 16 Megabyte Removable CMD drive
```

APPENDIX D - DISK COMMUNICATION STATUS CODES

The status word returned by the disk TRAP #15 routines flags an error condition if it is *non-zero*. 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

Code	Definition
$00	No error detected.
$01	Invalid Controller Type.
$02	Invalid Controller LUN.
$03	Invalid Device LUN.
$04	Controller Initialization Failed.
$05	Command aborted via break.
$06	Invalid Command Packet.
$07	Invalid address for transfer.

MVME319 CONTROLLER DEPENDENT STATUS CODES

Code	Definition
$ØØ	Correct execution without error.
$Ø1	Data CRC/ECC error.
$Ø2	Disk write protected.
$Ø3	Drive not ready.
$Ø4	Deleted data mark read.
$Ø5	Invalid drive number.
$Ø6	Invalid disk address.
$Ø7	Restore error.
$Ø8	Record not found.
$Ø9	Sector ID CRC/ECC error.
$ØA	VMEbus DMA error.
$ØF	Controller error.
$1Ø	Drive error.
$11	Seek error.
$12	I/O DMA error.

MVME32Ø CONTROLLER DEPENDENT STATUS CODES

Code	Definition
$ØØ	Correct execution without error.
$Ø1	Nonrecoverable error which cannot be completed (auto retries were attempted).
$Ø2	Drive not ready.
$Ø3	Reserved.
$Ø4	Sector address out of range.
$Ø5	Throughput error (floppy data overrun).
$Ø6	Command rejected (illegal command).
$Ø7	Busy (controller busy).
$Ø8	Drive not available (head out of range).
$Ø9	DMA operation cannot be completed (VMEbus error).
$ØA	Command abort (reset busy).
$ØB-$FF	Not used.

MVME321 CONTROLLER DEPENDENT STATUS CODES

Code	Definition
	*** General Error Codes ***
$ØØ	Correct execution without error.
$17	Timeout.
$18	Bad drive.
$1A	Bad Command.
$1E	Fatal Error.
	*** Hard Disk Error Codes ***
$Ø1	Write protected disk.
$Ø2	Sector not found.
$Ø3	Drive not ready.
$Ø4	Drive fault or timeout on recalibrate.
$Ø5	CRC or ECC error in data field.
$Ø6	UPD7261 FIFO overrun/underrun.
$Ø7	End of cylinder.
$Ø8	Illegal drive specified.
$Ø9	Illegal cylinder specified.
$ØA	Format operation failed.
$ØB	Bad disk descriptor.
$ØC	Alternate track error.
$ØD	Seek error.
$ØE	UPD7261 busy.
$ØF	Data does not verify.
$1Ø	CRC error in ID field.
$11	Reset request (missing address mark).
$12	Correctable ECC error.
$13	Abnormal command completion.
$2Ø	Missing Data Mark.
	*** Floppy Disk Error Codes ***
$Ø1	End-of-transfer size mismatch.
$Ø2	Bad tpi combination specified.
$Ø3	Drive motor not coming on.
$Ø4	Disk door open.
$Ø5	Command not completing.
$Ø6	Bad restore operation.

MVME321 CONTROLLER DEPENDENT STATUS CODES (cont.)

Code	Definition
$Ø7	Illegal side reference on device.
$Ø8	Illegal track reference on device.
$Ø9	Illegal sector reference on device.
$ØA	Illegal step rate specified.
$ØB	Bad density specified.
$ØC	Write protected disk.
$ØD	Format error.
$ØE	Can not find side, track, or sector.
$ØF	CRC error in ID field(s).
$1Ø	CRC error in data field.
$11	DMA underrun.
$2Ø	Bad disk size in descriptor.

MVME323 CONTROLLER DEPENDENT STATUS CODES

Code	Definition
$ØØ	Correct execution without error.
$1Ø	Disk not ready.
$12	Seek error.
$13	ECC code error-data field.
$14	Invalid command code.
$15	Illegal fetch and execute command.
$16	Invalid sector command.
$17	Illegal memory types.
$18	Bus time out.
$19	Header checksum error.
$1A	Disk write protected.
$1B	Unit not selected.
$1C	Seek error timeout.
$1D	Fault timeout.
$1E	Drive faulted.
$1F	Ready timeout.
$2Ø	End of media.
$21	Translation fault.

MVME323 CONTROLLER DEPENDENT STATUS CODES (cont.)

Code	Definition
$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 write format.
$29	Sector not found.
$2A	ID field error - wrong head.
$2B	Invalid sync in data field.
$2C	No valid header found.
$2D	Seek timeout error.
$2E	Busy timeout.
$2F	Not on cylinder.
$3Ø	RTZ timeout.
$31	Invalid sync in header.
$32-3E	Not used.
$3F	No heads specified.
$4Ø	Unit not initialized.
$41	Not used.
$42	Gap specification error.
$43-4A	Not used.
$4B	Seek error.
$4C-4F	Not used.
$5Ø	Sectors per track specification 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.
$6Ø	IOPB failed.
$61	DMA failed.
$62	Illegal VME address.

MVME323 CONTROLLER DEPENDENT STATUS CODES (cont.)

Code	Definition
$63-69	Not used.
$6A	Unrecognized header field.
$6B	Mapped header error.
$6C-6E	Not used.
$6F	No spare sector enabled.
$70-76	Not used.
$77	Command aborted.
$78	AC-fail detected.
$79-EF	Not used.
$F0-FE	Fatal Error.
$FF	Command not implemented.

MVME327 CONTROLLER DEPENDENT STATUS CODES

Code	Definition
	*** Command Parameter Errors ***
$01	Bad descriptor.
$02	Bad command.
$03	Unimplemented command.
$04	Bad drive.
$05	Bad logical disk address.
$06	Bad scatter-gather table.
$07	Unimplemented device.
$08	Unit not initialized.
	*** Media Errors ***
$10	No ID found on track.
$11	Seek error.
$12	Relocated track error.
$13	Record not found, bad ID.
$14	Data sync fault.
$15	Non-correctable data error.
$16	Record not found.
$17	Media error.

MVME327 CONTROLLER DEPENDENT STATUS CODES (cont.)

Code	Definition
	*** Drive Errors ***
$2Ø	Drive fault.
$21	Write protected disk.
$22	Motor not on.
$23	Door open.
$24	Drive not ready.
$25	Drive busy.
	*** VME DMA Errors ***
$3Ø	VMEbus error.
$31	Bad address alignment.
$32	Bus timeout.
$33	Invalid DMA transfer count.
	*** Disk Format Errors ***
$4Ø	Not enough alternates.
$41	Format failed.
$42	Verify error.
$43	Bad format parameters.
$44	Cannot fix bad spot.
$45	Too many defects.

MVME35Ø CONTROLLER DEPENDENT STATUS CODES

Code	Definition
$ØØ	Correct execution without error.
$Ø1	Block in error not located.
$Ø2	Unrecoverable data error.
$Ø3	End of media.
$Ø4	Write protected.
$Ø5	Drive offline.
$Ø6	Cartridge not in place.
$ØD	No data detected.
$ØE	Illegal command.
$12	Tape reset did not occur.

MVME350 CONTROLLER DEPENDENT STATUS CODES (cont.)

Code	Definition
$17	Timeout.
$18	Bad drive.
$1A	Bad Command.
$1E	Fatal Error.

MVME360 CONTROLLER DEPENDENT STATUS CODES

Code	Definition
$00	Correct execution without error.
$10	Disk not ready.
$12	Seek error.
$13	ECC code error-data field.
$14	Invalid command code.
$15	Illegal fetch and execute command.
$16	Invalid sector command.
$17	Illegal memory types.
$18	Bus time out.
$19	Header checksum error.
$1A	Disk write protected.
$1B	Unit not selected.
$1C	Seek error timeout.
$1D	Fault timeout.
$1E	Drive faulted.
$1F	Ready timeout.
$20	End of media.
$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 write format.
$29	Sector not found.

MVME36Ø CONTROLLER DEPENDENT STATUS CODES (cont.)

Code	Definition
$2A	ID field error - wrong head.
$2B	Invalid sync in data field.
$2C	No valid header found.
$2D	Seek timeout error.
$2E	Busy timeout.
$2F	Not on cylinder.
$3Ø	RTZ timeout.
$31	Invalid sync in header.
$32-3F	Not used.
$4Ø	Unit not initialized.
$41	Not used.
$42	Gap specification error.
$43-4A	Not used.
$4B	Seek error.
$4C-4F	Not used.
$5Ø	Sectors per track specification 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.
$6Ø	IOPB failed.
$61	DMA failed.
$62	Illegal VME address.
$63-69	Not used.
$6A	Unrecognized header field.
$6B	Mapped header error.
$6E	Not used.
$6F	No spare sector enabled.
$7Ø-76	Not used.
$77	Command aborted.
$78	AC-fail detected.
$79-EF	Not used.

MVME360 CONTROLLER DEPENDENT STATUS CODES (cont.)

Code	Definition
$F0-FE	Fatal Error.
$FF	Command not implemented.

APPENDIX E - VME135 STATUS REGISTER (STAT1)

STAT1 is a software-accessible board status register on the VME135 module. It is implemented in hardware as an ten-position DIP switch. The reference designator of this DIP switch is S4. The contents of this register may be obtained, with the exception of Bits 8 and 9, by reading a byte at $FFFBØØØD. STAT1 is a read-only register and reflects the settings of the user configuration switch. This status register is examined by 135Bug to determine the user's preferences concerning the 135Bug operating environment. Certain control registers on the VME135 are then set up by 135Bug in accordance with the user's selections.

STAT1 appears to software as shown below.

Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit Ø
SCON	PBDIS	ENVØ	ENV1	D32	A32´	VSBSC	VSBEN	MPSUP	BOOT
S4-1	S4-2	S4-3	S4-4	S4-5	S4-6	S4-7	S4-8	S4-9	S4-1Ø

If a particular switch is open (**OFF**), the line is pulled up and the bit will be read as a logical 1. If the switch is closed (**ON**), the bit will be read as a logical Ø.

The board status information in STAT1 is a follows:

SCON < **VMEbus System Controller**>

This switch is used to configure the VME135 as the VMEbus system controller in a multi-processor environment. Only one board on a VMEbus can be configured as the system controller. The front panel LED labeled "SCON" is illuminated when the module is configured as the system controller. Only one SCON should be on in systems that use more than one VME135 on a single VMEbus backplane.

SCON = Ø (ON): This VME135 is the system controller.
SCON = 1 (OFF): This VME135 is not the system controller.

PBDIS **< Pushbutton Enable/Disable Select>**

This switch is used to select the ABORT/RESET pushbutton enable/disable. When disabled, pushing the ABORT or RESET pushbuttons, will have no effect on hardware or the software currently executing.

PBDIS = Ø (ON): RESET and ABORT pushbuttons enabled.
PBDIS = 1 (OFF): RESET and ABORT pushbuttons disabled.

ENVØ-ENV1 **< Operating Environment Select Bits>**

Interpreted by 135Bug, these bits (switches S4-3 and S4-4, respectively) determine certain defaults which are set up at power-up/reset and dictate the 135Bug operating environment. These defaults include the location of 135Bug's VBR and stack space. For more information refer to section 1.5, "ENVØ,ENV1 Switches" and to section 1.7, "Memory Requirements".

ENVØ	ENV1	Description
Ø	Ø	135Bug operates locally at BASE Ø
Ø	1	135Bug operates locally at high memory BASE $FFXØØØØØ
1	Ø	135Bug operates over VMEbus BASE Ø, with OFFSET calculated by (board n - 1) * $4ØØØ. (n = 1, 2, 3, etc.)
1	1	135Bug operates in first VMEbus non-mapped (DRAM) memory space with OFFSET calculated by ID byte * $4ØØØ.

D32 **< Data Bus Width Select>**

This bit (switch S4-5) provides a software selectable 32- and 16-bit VMEbus data width. This bit should be used with care because when D32 = Ø, all memory references to the VMEbus can be 32 bits. When D32 = 1, all memory references to VMEbus are forced to be 16 bits.

D32 = Ø (ON): Selects 32-bit data.
D32 = 1 (OFF): Selects 16-bit data.

A32 **< Address Bus Width Select>**

This bit (switch S4-6) provides a software selectable 32- and 24-bit address option for VMEbus references. The appropriate address modifiers are generated for 32- or 24-bit address VMEbus accesses. A32 = Ø indicates 32-bit address option; A32 = 24 indicates a 24-bit address space.

A32 = Ø (ON): Selects 32-bit addressing.
A32 = 1 (OFF): Selects 24-bit addressing.

VSBSC **< VSB System Controller>**

This bit (switch S4-7) is used to configure the VME135 as the VSB system controller in a multi-processor environment. Only one board on a VSB can be configured as the system controller.

VSBSC = Ø (ON) : If VSB enabled, this VME135 is the VSB bus system controller.
VSBSC = 1 (OFF): If VSB disabled, this VME135 is not the VSB bus system controller.

VSBEN **< VSB Enable>**

This bit (switch S4-8) is used to select the VSB bus mode. When VSBEN = 1, all VSB activity is suspended. Setting VSBEN = Ø, enables the VSB bus.

VSBEN = Ø (ON) : Enables VSB.
VSBEN = 1 (OFF): Disables VSB. (all off-board accesses are done over VMEbus).

MPSUP **< Multi-Processor Support>**

This bit (switch S4-9) is used to select the MP-CSR bit psuedo interrupt handling option.

MPSUP = Ø (ON) : Disables polling of MP bits LMØ, SIGLP, SIGHP, and BRIRQ.
MPSUP = 1 (OFF): Enables polling of MP bits LMØ, SIGLP, SIGHP, and BRIRQ.

BOOT **< Bootstrap Mode>**

The BOOT bit (switch S4-1Ø) selects the mechanism to be used for operating the system bootstrap.

```
BOOT = Ø (ON) : Select manual boot (using BO/BH commands).
BOOT = 1 (OFF): Enable autoboot operation (BOOT from ROM, DISK or
                TAPE).
```

APPENDIX F - MAPPING SWITCH (S3)

Switch S3 is the slave resource mapping switch. It is an eight-position piano type DIP switch that maps the memory and MPCSR on the VMEbus.

```
-------------------------------------------------------------------
  (Address): A  A  A  A    A  A  A  A    A  A  A  A    A  A  A  A
  (Bit #):   1  1  1  1    1  1  Ø  Ø    Ø  Ø  Ø  Ø    Ø  Ø  Ø  Ø
             5  4  3  2    1  Ø  9  8    7  6  5  4    3  2  1  Ø
-------------------------------------------------------------------
MPCSR =     |x  x  x| Ø    Ø  Ø |x  x    x  x  x| Ø    Ø  Ø  Ø  Ø
Base        +-------+           +---------------+
Addr            |                          |
                |            ------------
                |            |
       +-----------+-----------------+
      /            |                  \
     +-----------------------------------+
     | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |     S3 Switch Positions
     +-----------------------------------+
                   \                  /
                    +-----------------+
   DRAM                      |
   Base               +------------+
   Addr =   ØØØØ  Ø Ø Ø|x  x x x x | ØØØØ  ØØØØ  ØØØØ  ØØØØ  ØØØØ
   ------------------------------------------------------------------
            AAAA  A A A A  A A A A   AAAA  AAAA  AAAA  AAAA  AAAA
            3322  2 2 2 2  2 2 2 2   1111  1111  11ØØ  ØØØØ  ØØØØ
            1Ø98  7 6 5 4  3 2 1 Ø   9876  5432  1Ø98  7654  321Ø
   ------------------------------------------------------------------
```

The following illustrations provide various examples of MPCSR and DRAM base addressing.

Example 1:

```
+---------------------------+           Switch S3
|↑  1  2  3  4  5  6  7  8  |         Mapping Switch
|O[x][x][ ][ ][ ][ ][ ][ ]  |     (Factory Configuration)
|N[ ][ ][x][x][x][x][x][x]  |
+---------------------------+     (Note: ON is Ø, OFF is 1)
    O  O  O  O  O  O  O  O
    F  F  N  N  N  N  N  N
    F  F

    1  1  Ø  Ø  Ø  Ø  Ø  Ø
```

MPCSR
Base Addr = 1 1 Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø ØØØØ = $CØØØ

DRAM
Base Addr = ØØØØ Ø Ø Ø Ø Ø Ø Ø Ø ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ = $ØØØØ ØØØØ

Example 2:

```
+---------------------------+           Switch S3
|↑  1  2  3  4  5  6  7  8  |         Mapping Switch
|O[x][x][ ][ ][ ][ ][x][ ]  |
|N[ ][ ][x][x][x][x][ ][x]  |
+---------------------------+     (Note: ON is Ø, OFF is 1)
    O  O  O  O  O  O  O  O
    F  F  N  N  N  N  F  N
    F  F              F

    1  1  Ø  Ø  Ø  Ø  1  Ø
```

MPCSR
Base Addr = 1 1 Ø Ø Ø Ø Ø Ø Ø 1 Ø Ø ØØØØ = $CØ4Ø

DRAM
Base Addr = ØØØØ Ø Ø Ø Ø Ø Ø 1 Ø ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ = $ØØ2Ø ØØØØ

APPENDIX G - VME135 CONFIDENCE TEST STATUS CODES

This appendix contains information about a software-accessible Confidence Test Status byte which is available on a Power up/Reset sequence. The status code may be obtained by reading the MP COMM byte in the local MP-CSR at $FFFBØØ79, or the MP-CSR's address over the VMEbus at $FFFFXXX9 (refer to Appendix F, Mapping Switch S3).

During normal 135Bug operation, a message will be displayed on a Confidence Test failure, indicating the failure code. If the Confidence Test completes successfully, no message is displayed, and the status code will be set to $Ø.

When the Confidence Test status check is not done in the normal 135Bug prompt mode, wait until the BSY bit in the MP-CSR is cleared, to assure the test has completed and the status has been updated.

The Confidence Test Code assignments follow:

CONFIDENCE TEST CODE ASSIGNMENTS

Test	Code	Description
PASS	Ø	Confidence Test Passed
CPU_A	$A	MPU Register Test Failure
CPU_B	$B	MPU Instruction Test Failure
CPU_C	$C	VME135 EPROM Test Failure
CPU_D	$D	VME135 Local Ram Test Failure
CPU_E	$E	MPU Addressing Mode Test Failure
CPU_F	$F	VME135 Status and Control Register Test Failure
CPU_G	$1Ø	MPU Exception Test Failure
CPU_I	$12	VME135 MP-CSR Test Failure
SIO_Ø	$AØ	VME135 DUART Register Test Failure
SIO_1	$A1	VME135 DUART Register Test Failure
SIO_2	$A2	VME135 DUART Register Test Failure
SIO_3	$A3	VME135 DUART Register Test Failure
SIO_4	$A4	VME135 DUART Port Register Test Failure
SIO_5	$A5	VME135 DUART Port Register Test Failure
SIO_F	$AF	VME135 DUART Port Register Test Failure

CONFIDENCE TEST CODE ASSIGNMENTS (cont.)

Test	Code	Description
SIOTX_Ø	$BØ	VME135 DUART Transmitter Test Failure
SIOTX_1	$B1	VME135 DUART Transmitter Test Failure
SIOTX_3	$B3	VME135 DUART Transmitter Test Failure
SIOTX_5	$B5	VME135 DUART Transmitter Test Failure
SIOTX_7	$B7	VME135 DUART Transmitter Test Failure
SIOTX_F	$BF	VME135 DUART Transmitter Test Failure
SIORX_Ø	$CØ	VME135 DUART Receiver Test Failure
SIORX_2	$C2	VME135 DUART Receiver Test Failure
SIORX_3	$C3	VME135 DUART Receiver Test Failure
SIORX_4	$C4	VME135 DUART Receiver Test Failure
SIORX_F	$CF	VME135 DUART Receiver Test Failure
SIOTIM_Ø	$DØ	VME135 DUART Timer Test Failure
SIOTIM_1	$D1	VME135 DUART Timer Test Failure
SIOTIM_2	$D2	VME135 DUART Timer Test Failure
SIOTIM_3	$D3	VME135 DUART Timer Test Failure
SIOTIM_4	$D4	VME135 DUART Timer Test Failure
SIOTIM_5	$D5	VME135 DUART Timer Test Failure
SIOTIM_6	$D6	VME135 DUART Timer Test Failure
SIOTIM_F	$DF	VME135 DUART Timer Test Failure

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 INC.

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