

ARTISAN®
TECHNOLOGY GROUP

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

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

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

Field descriptions:

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

Device LUN – LUN of device to use.

Status Word – This status word reflects the result of the operation. It is zero if the command completed without errors. Refer to Appendix F for meanings of returned error codes.

Memory Address – Address of buffer in memory. On a disk read, data is written starting at this address. On a disk write, data is read starting at this address.

Block Number – For disk devices, this is the block number where the transfer starts. On a disk read, data is read starting at this block. On a disk write, data is written starting at this block.

File Number – For streaming tape devices, this is the file number where the transfer starts. This field is used if the IFN bit in the Flag Byte is cleared (refer to the Flag Byte description).

Number of Blocks – This field indicates the number of blocks to read from the disk (.DSKRD) or to write to the disk (.DSKWR). For streaming tape devices, the actual number of blocks transferred is returned in this field.

Flag Byte – The flag byte is used to specify variations of the same command, and to receive special status information. Bits 0 through 3 are used as command bits, and bits 4 through 7 are used as status bits. For disk devices, this field must be set to zero. For streaming tape devices, the following bits are defined:

Bit 7 Filemark flag. If 1, a filemark was detected at the end of the last operation.

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

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

Address Modifier – VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If nonzero, the specified value is used.

ENTRY CONDITIONS:

SP ==> Address (longword) Address of command packet

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Status word of command packet is updated.
Data is written into memory as a result of .DSKRD function.
Data is written to disk as a result of .DSKWR function.
Z(ero) = Set to 1 if no errors.

EXAMPLE: If A0, A1 point to packets formatted as specified above;

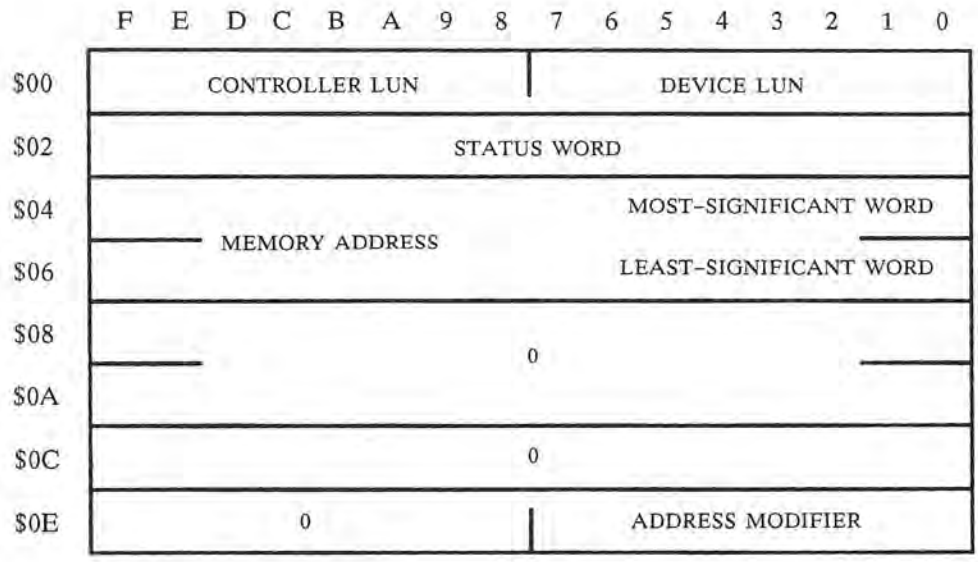
```
        PEA.L   (A0)
        SYSCALL .DSKRD          Read from disk
        BNE     ERROR           Branch if error
        PEA.L   (A1)
        SYSCALL .DSKWR          Write to disk
        BNE     ERROR           Branch if error
        .
        .
        .
ERROR   xxxxx   xxx             Handle error
        xxxxx   xxx
```

.DSKCFIG Function

TRAP FUNCTION: .DSKCFIG – Disk configure function

CODE: $0012

DESCRIPTION: This function allows the user to change the configuration of the specified device. It effectively performs an "IOT under program control". All the required parameters are passed in a command packet which has been built somewhere in memory. The address of the packet is passed as an argument to the function. The packet format is as follows:



Field descriptions:

Controller LUN – LUN of controller to use.

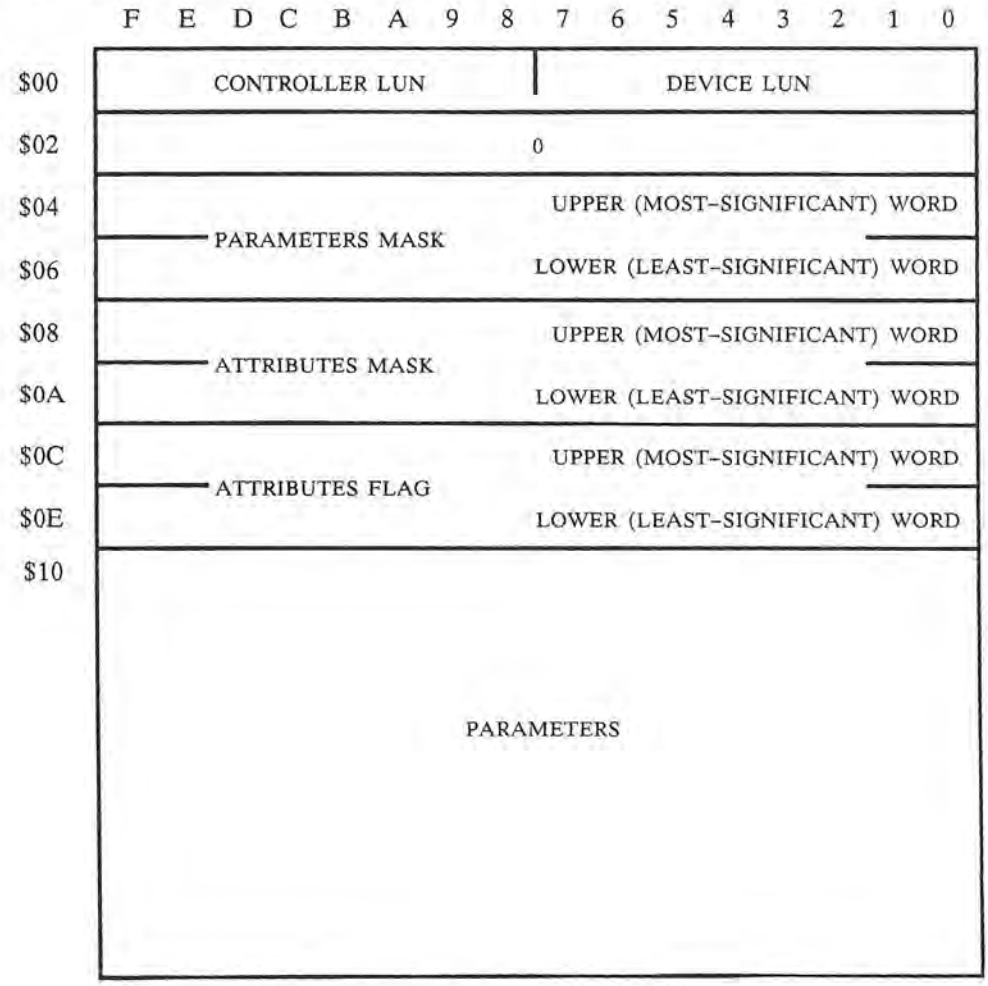
Device LUN – LUN of device to use.

Status Word – This status word reflects the result of the operation. It is zero if the command completed without errors. Refer to Appendix F for meanings of returned error codes.

Memory Address – Contains a pointer to a Device Descriptor Packet that contains the configuration information to be changed.

Address Modifier – VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If nonzero, the specified value is used.

The Device Descriptor Packet is as follows:



.DSKCFIG

Most of the fields in the device descriptor packet are equivalent to the fields defined in the CFGA configuration area block, as described in Appendix D. In the field descriptions following, reference is made to the equivalent field in the CFGA whenever possible. For additional information on these fields, refer to Appendix D.

Controller LUN – Same as in command packet.

Device LUN – Same as in command packet.

Parameters Mask – Equivalent to the IOSPRM and IOSEPRM fields, with the lower word equivalent to IOSPRM, and the upper word equivalent to IOSEPRM.

Attributes Mask – Equivalent to the IOSATM and IOSEATM fields, with the lower word equivalent to IOSATM, and the upper word equivalent to IOSEATM.

Attributes Flags – Equivalent to the IOSATW and IOSEATW fields, with the lower word equivalent to IOSATW, and the upper word equivalent to IOSEATW.

Parameters – The parameters used for device reconfiguration are specified in this area. Most parameters have an *exact* CFGA equivalent. The following list shows the field name, offset from start of packet, length, equivalent CFGA field, and short description of each field. Those parameters that do not have an *exact* equivalent are indicated with "*", and are explained after the list.

FIELD NAME	OFFSET (BYTES)	LENGTH (BYTES)	CFGA EQUIV.	DESCRIPTION
P_DDS*	$10	1	–	Device descriptor size
P_DSR	$11	1	IOSSR	Step rate
P_DSS*	$12	1	IOSPSM	Sector size (encoded)
P_DBS*	$13	1	IOSREC	Block size (encoded)
P_DST*	$14	2	IOSSPT	Sectors/track
P_DIF	$16	1	IOSILV	Interleave factor
P_DSO	$17	1	IOSSOF	Spiral offset
P_DSH*	$18	1	IOSSHD	Starting head
P_DNH	$19	1	IOSHDS	Number of heads
P_DNCYL	$1A	2	IOSTRK	Number of cylinders
P_DPCYL	$1C	2	IOSPCOM	Precompensation cylinder
P_DRWCYL	$1E	2	IOSRWCC	Reduced write current cylinder
P_DECCB	$20	2	IOSECC	ECC data burst length
P_DGAP1	$22	1	IOSGPB1	Gap 1 size
P_DGAP2	$23	1	IOSGPB2	Gap 2 size
P_DGAP3	$24	1	IOSGPB3	Gap 3 size
P_DGAP4	$25	1	IOSGPB4	Gap 4 size
P_DSSC	$26	1	IOSSSC	Spare sectors count
P_DRUNIT	$27	1	IOSRUNIT	Reserved area units
P_DRCALT	$28	2	IOSRSVC1	Reserved count for alternates
P_DRCCTR	$2A	2	IOSRSVC2	Reserved count for controller

List Notes:

P_DDS This field is for internal use only, and does not have an equivalent CFGA field. It should be set to 0.

P_DSS This is a one byte encoded field, whereas the IOSPSM field is a two byte unencoded field containing the actual number of bytes per sector. The P_DSS field is encoded as follows:

$00	128 bytes
$01	256 bytes
$02	512 bytes
$03	1024 bytes
$04–$FF	Reserved encodings

P_DBS This is a one byte encoded field, whereas the IOSREC field is a two byte unencoded field containing the actual number of bytes per record (block). The P_DBS field is encoded as follows:

$00	128 bytes
$01	256 bytes
$02	512 bytes
$03	1024 bytes
$04–$FF	Reserved encodings

P_DST This is a two byte field, whereas the IOSSPT field is one byte.

P_DSH This is a one byte field, whereas the IOSSHD field is two bytes. This field is equivalent to the lower byte of IOSSHD.

ENTRY CONDITIONS:

SP ==> Address (longword) Address of command packet

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Status word of command packet is updated.
The device configuration is changed.
Z(ero) = Set to 1 if no errors.

EXAMPLE: If A0 points to a packet formatted as specified above;

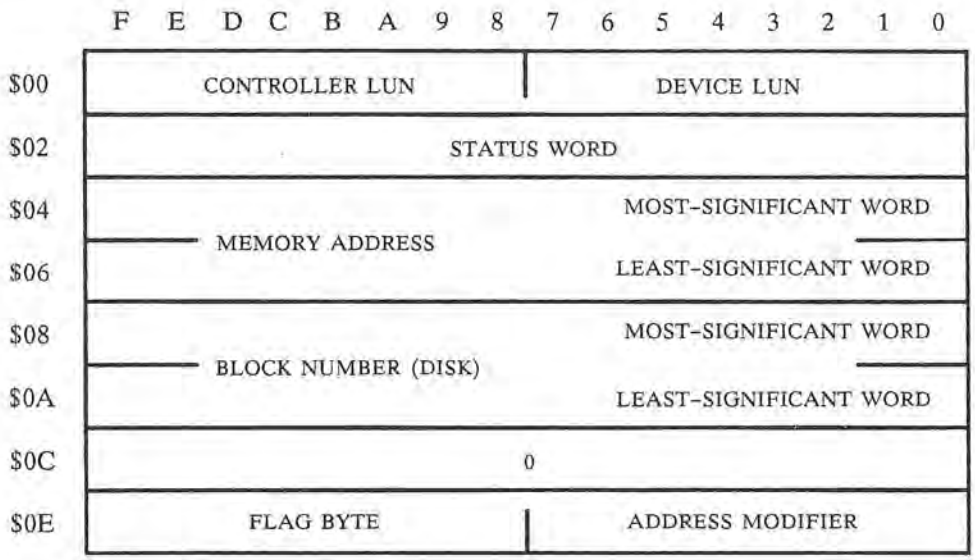
```
        PEA.L    (A0)          Load command packet
        SYSCALL  .DSKCFIG      Reconfigure device
        BNE      ERROR         Branch if error
        .
        .
        .
ERROR   xxxxx    xxx           Handle error
        xxxxx    xxx
```

.DSKFMT Function

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:



Field descriptions:

Controller LUN – LUN of controller to use.

Device LUN – LUN of device to use.

Status Word – This status word reflects the result of the operation. It is zero if the command completed without errors. Refer to Appendix F for meanings of returned error codes.

Memory Address – Address of buffer in memory. On a disk read, data is written starting at this address. On a disk write, data is read starting at this address.

Block Number – For disk devices, when doing a format track, the track that contains this block number is formatted. This field is ignored for streaming tape devices.

Flag Byte – Contains additional information. Bit 0 is interpreted as follows for disk devices:

If 0, it indicates a "Format Track" operation. The track that contains the specified block is formatted.

If 1, it indicates a "Format Disk" operation. All the tracks on the disk are formatted.

For streaming tapes, bit 0 is interpreted as follows:

If 0, it selects a "Retension Tape" operation. This rewinds the tape to BOT, advances the tape without interruptions to EOT, and then rewinds it back to BOT. Tape retension is recommended by cartridge tape suppliers before writing or reading data when a cartridge has been subjected to a change in environment or a physical shock, has been stored for a prolonged period of time or at extreme temperature, or has been previously used in a start/stop mode.

If 1, it selects an "Erase Tape" operation. This completely clears the tape of previous data and at the same time retensions the tape.

Address Modifier – VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If nonzero, the specified value is used.

ENTRY CONDITIONS:

SP ==> Address (longword) Address of command packet

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Status word of command packet is updated.
Z(ero) = Set to 1 if no errors.

EXAMPLE: If A0 points to a packet formatted as specified above;


```
        PEA.L   (A0)        Load command packet
        SYSCALL .DSKFMT     format device
        BNE     ERROR       If errors, branch
        .
        .
        .
ERROR   xxxxx   xxx         Handle error
        xxxxx   xxx
```

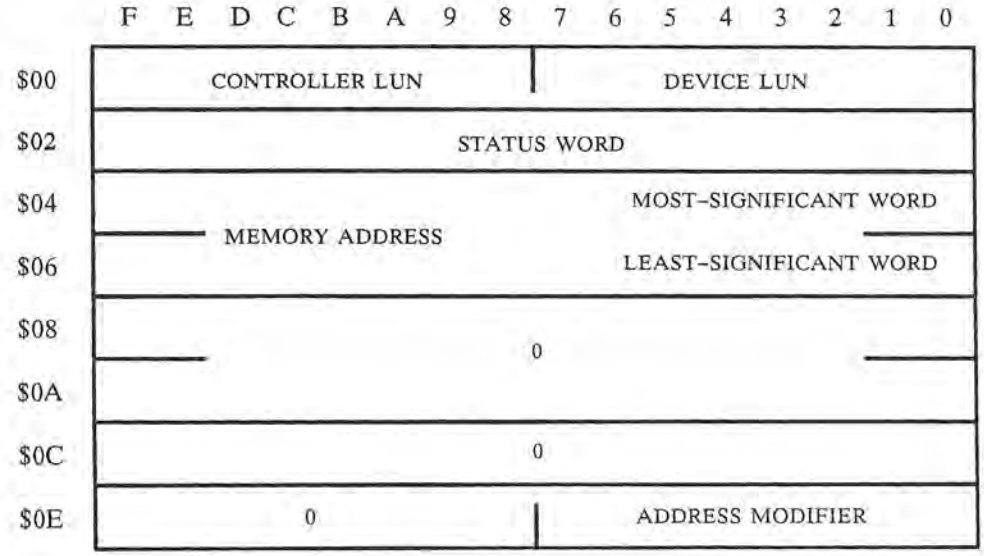
.DSKCTRL Function

TRAP FUNCTION: .DSKCTRL – Disk control function

CODE: $0015

DESCRIPTION: This function is used to implement any special device control functions that cannot be accommodated easily with any of the other disk functions. At the present, the only defined function is SEND packet, which allows the user to send a packet in the specified format of the controller. The required parameters are passed in a command packet which has been built somewhere in memory. The address of the packet is passed as an argument to the function.

The packet is as follows:



Field descriptions:

Controller LUN - LUN of controller to use.

Device LUN - LUN of device to use.

Status Word - This status word reflects the result of the operation. It is zero if the command completed without errors. Refer to Appendix F for meanings of returned error codes.

Memory Address - Contains a pointer to the controller packet to send. Note that the controller packet to send (as opposed to the command packet) is controller and device dependent. Information about this packet should be found in the user's manual for the controller and device being accessed.

Address Modifier - VMEbus address modifier to use while transferring data. If zero, a default value is selected by the bug. If nonzero, the specified value is used.

ENTRY CONDITIONS:

SP ==> Address (longword) Address of command packet

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Status word of command packet is updated.
Additional side effects depend on the packet sent to the controller.
Z(ero) = Set to 1 if no errors.

EXAMPLE: If A1 points to a packet formatted as specified above;

```
        PEA.L    (A1)          Load command packet
        SYSCALL  .DSKCTRL      invoke control function
        BNE      ERROR         If errors, branch
        .
        .
        .
ERROR   xxxxx    xxx           Handle error
        xxxxx    xxx
```

.OUTCHR Function

TRAP FUNCTION: .OUTCHR - Output character routine

CODE: $0020

DESCRIPTION: This function outputs a character to the default output port.

ENTRY CONDITIONS:

SP ==> Character (byte)
Word fill (byte) (Placed automatically by MPU)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Character is sent to the default I/O port.

EXAMPLE:

```
MOVE.B   D0,-(SP)      Send character in D0
SYSCALL  .OUTCHR       To default output port.
```

.OUTSTR, .OUTLN Functions

.OUTSTR
.OUTLN

TRAP FUNCTIONS:. OUTSTR - Output string to default output port
.OUTLN - Output string along with (CR)(LF)

CODES: $0021
$0022

DESCRIPTION: .OUTSTR outputs a string of characters to the default output port. .OUTLN outputs a string of characters followed by a (CR)(LF) sequence.

5

ENTRY CONDITIONS:

SP ==> Address of first character (longword)
+4 Address of last character+1 (longword)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack

EXAMPLE: If A0 = start of string
A1 = end of string+1

```
MOVEM.L  A0/A1,-(SP)      Load pointers to string
SYSCALL  .OUTSTR          And print it.
```

.WRITE, .WRITELN Functions

.WRITE
.WRITELN

TRAP FUNCTIONS: .WRITE – Output string with no CR or LF
.WRITELN – Output string with CR and LF

CODES: $0023
$0024

DESCRIPTION: These output functions are designed to output strings formatted with a count byte followed by the characters of the string. The user passes the starting address of the string. The output goes to the default output port.

ENTRY CONDITIONS:

Four bytes of parameter positioned in stack as follows:

SP ==> Address of string <long>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Parameter stack space will have been deallocated.

EXAMPLE: For example, the following section of code ...

```
MESSAGE1    DC.B    9, 'MOTOROLA '
MESSAGE2    DC.B    9, 'QUALITY !'
            .
            .
            .
            PEA.L   MESSAGE1(PC)    Push address of string.
            SYSCALL .WRITE          Use TRAP #15 macro.
            PEA.L   MESSAGE2(PC)    Push address of other string.
            SYSCALL .WRITE          Invoke function again.
```

... would print out the following message:

```
MOTOROLA QUALITY!
```

Using function .WRITELN, however, instead of function .WRITE would output the following message:

```
MOTOROLA
QUALITY!
```

NOTE: The string must be formatted such that the first byte (the byte pointed to by the passed address) contains the count (in bytes) of the string. There is no special character at the end of the string as a delimiter.

.PCRLF Function

TRAP FUNCTION: .PCRLF - Print (CR)(LF) sequence

CODE: $0026

DESCRIPTION: Sends a (CR)(LF) sequence to the default output port.

ENTRY CONDITIONS:

No arguments or stack allocation required.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

None

EXAMPLE:

```
SYSCALL   .PCRLF          Output CRLF
```

.ERASLN Function

.ERASLN

TRAP FUNCTION: .ERASLN – Erase Line

CODE: $0027

DESCRIPTION: Used to erase the line at the present cursor position. If the terminal type flag is set for hardcopy mode, a (CR)(LF) is issued instead.

ENTRY CONDITIONS:

No arguments required.

5

EXIT CONDITIONS DIFFERENT FROM ENTRY:

The cursor is positioned at the beginning of a blank line.

EXAMPLE:

```
SYSCALL   .ERASLN
```

.WRITD, .WRITDLN Functions

.WRITD
.WRITDLN

TRAP FUNCTIONS: .WRITD - Output string with data
.WRITDLN - Output string with data and (CR)(LF)

CODES: $0028
$0025

DESCRIPTION: These trap functions take advantage of the monitor I/O routine which outputs a user string containing embedded variable fields. The user passes the starting address of the string and the address of a data stack containing the data which is inserted into the string. The output goes to the default output port.

ENTRY CONDITIONS:

Eight bytes of parameter positioned in stack as follows:

SP ==> Address of string (longword)
Data list pointer (longword)

A separate data stack or data list arranged as follows:

Data list pointer =>	Data for 1st variable in string	(longword)
	Data for next variable	(longword)
	Data for next variable	(longword)
	etc.	

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Parameter stack space will have been deallocated.

EXAMPLE: For example, the following section of code ...

```
ERRMESSG  DC.B      $14,'ERROR CODE = |10,8Z|'
```

```
          .
          .
          .
          MOVE.L    #3,-(A5)        Push error code on data stack.
          PEA.L     (A5)            Push data stack location.
          PEA.L     ERRMESSG(PC)    Push address of string.
          SYSCALL   .WRITDLN        Invoke function.
          TST.L     (A5)+           Deallocate data from data stack.
```

... would print out the following message:

```
ERROR CODE = 3
```

NOTES:
1. The string must be formatted such that the first byte (the byte pointed to by the passed address) contains the count (in bytes) of the string (including the data field specifiers, described in NOTE 2. following).
2. Any data fields within the string must be represented as follows: "|*radix fieldwidth*[**Z**]|" where *radix* is the base that the data is to be displayed in (in hexadecimal, for example, "A" is base 10, "10" is base 16, etc.) and *fieldwidth* is the number of characters this data is to occupy in the output. The data is right justified, and left-most characters are removed to make the data fit. The "Z" is included if it is desired to suppress leading zeros in output.
3. All data is to be placed in the stack as longwords. Each time a data field is encountered in the user string, a longword is read from the data stack to be displayed.
4. The data stack is not destroyed by this routine. If it is necessary that the space in the data stack be deallocated, then this must be done by the calling routine, as shown in the preceding example.

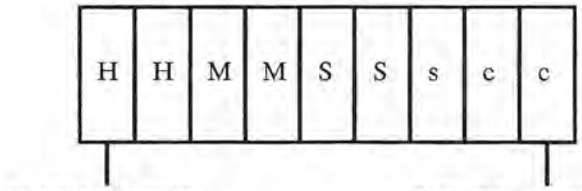
.RTC_TM Function

TRAP FUNCTION: .RTC_TM – Time initialization for RTC

CODE: $0050

DESCRIPTION: Initializes the MK48T02 Real-Time Clock with the time that is located in a user-specified buffer.

The data input format can be either ASCII or unpacked BCD. The order of the data in the buffer is:



ENTRY CONDITIONS:

SP ==> Time initialization buffer (address)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Parameter is deallocated from stack.

EXAMPLE: Time is to be initialized to 2:05:32 PM with a calibration factor of -15 (s=sign, cc=value).

Data in BUFFER is `3134 3035 3332 2D 3135` or
`x1x4 x0x5 x3x2 2D x1x5.` (x = don't care)

```
PEA.L     BUFFER(PC)      Put buffer address on stack
SYSCALL   .RTC_TM         Initialize time and start clock
```

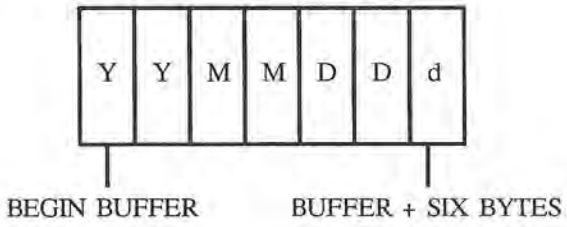
.RTC_DT Function

TRAP FUNCTION: .RTC_DT - Date initialization for RTC

CODE: $0051

DESCRIPTION: Initializes the MK48T02 Real-Time Clock with the date that is located in a user-specified buffer.

The data input format can be either ASCII or unpacked BCD. The order of the data in the buffer is:



ENTRY CONDITIONS:

SP ==> Date initialization buffer (address)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Parameter is deallocated from stack.

EXAMPLE: Date is to be initialized to Monday, Nov. 18, 1988 (d = day of week).

Data in BUFFER is `3838 3131 3138 32` or
`x8x8 x1x1 x1x8 x2.` (x = don't care)

```
PEA.L     BUFFER(PC)      Put buffer address on stack
SYSCALL   .RTC_DT         Initialize date and start clock
```

.SNDBRK Function

TRAP FUNCTION: .SNDBRK – Send break

CODE: $0029

DESCRIPTION: Used to send a break to the default output port(s).

ENTRY CONDITIONS:

No arguments or stack allocation required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Each serial port specified by current default port list has sent "break".

EXAMPLE:

```
SYSCALL  .SNDBRK
```

.DELAY Function

TRAP FUNCTION: .DELAY – Timer delay function

CODE: $0043

DESCRIPTION: This function is used to generate accurate timing delays that are independent of the processor frequency and instruction execution rate. This function uses the onboard timer for operation. The user specifies the desired delay count in milliseconds. .DELAY returns to the caller after the specified delay count is exhausted.

ENTRY CONDITIONS:

SP ==> Delay time in milliseconds (longword)

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack <long>

EXAMPLE:

```
PEA.L     &15000          Load a 15 second delay
SYSCALL   .DELAY
  *
  *
  *
PEA.L     &50             Load a 50 millisecond delay
SYSCALL   .DELAY
```

.RTC_DSP Function

TRAP FUNCTION: .RTC_DSP – Display time from RTC

CODE: $0052

DESCRIPTION: Displays the day of the week, date, and time in the following format:

(Day of week) MM/DD/YY hh:mm:ss

ENTRY CONDITIONS:

No arguments or stack allocation required

EXIT CONDITIONS DIFFERENT FROM ENTRY:

The cursor is left at the end of the string.

EXAMPLE: SYSCALL .RTC_DSP Displays the day, date, and time on the console.

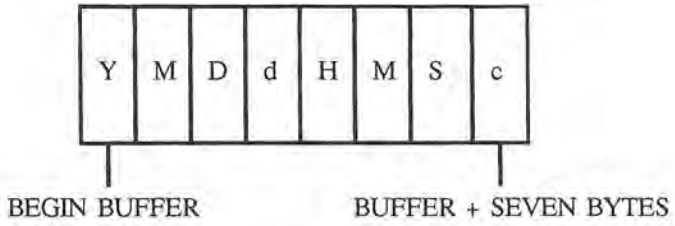
.RTC_RD Function

TRAP FUNCTION: .RTC_RD – Read the RTC registers

CODE: $0053

DESCRIPTION: Used to read the Real-Time Clock registers. The data returned is in BCD. The last byte of the returned data is the calibration value (c); bit #5 is a sign bit (1 indicates positive, 0 indicates negative).

The order of the data in the buffer is:



ENTRY CONDITIONS:

SP ==> Buffer address where RTC data is to be returned <long>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
Buffer now contains date and time in BCD format.

EXAMPLE: A date and time of Saturday, May 11, 1988 2:05:32 PM are be returned in the buffer (d = day of week, c = calibration value).

Data in buffer = `88 05 11 07 14 05 32 xx` (xx = unknown)

```
PEA.L    BUFFER(PC)     Put buffer address on stack
SYSCALL  .RTC_RD        Read timer
```

.REDIR Function

TRAP FUNCTION: .REDIR - Redirect I/O function

CODE: $0060

DESCRIPTION: Used to select an I/O port and at the same time invoke a particular I/O function. The invoked I/O function reads or writes to the selected port.

ENTRY CONDITIONS:

SP ==>	Port	<word>
	I/O function to call	<word>
	Parameters of I/O function	<size specified by function>
	Space for results	<size specified by function>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==>	Result	<size specified by function>

EXAMPLE: None

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

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

Then the parameters required by .REDIR should be pushed and the call made to .REDIR:

```
MOVE.W      #.INCHR,-(SP)  Load function code
MOVE.W      #1,-(SP)       Load port number
SYSCALL     .REDIR         Redirect I/O function
```

Finally, the results should be popped from the stack:

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

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

.REDIR_I, .REDIR_O Functions

.REDIR_I
.REDIR_O

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

CODES: $0061
$0062

DESCRIPTION: The .REDIR_I and .REDIR_O functions are used to change the default port number of the input and output ports, respectively. This is a permanent change, that is, it remains in effect until a new .REDIR command is issued.

ENTRY CONDITIONS:

SP ==> Port Number <word>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack
.SIO_IN - Loaded with a new mask if .REDIR_I called
.SIO_OUT - Loaded with a new mask if .REDIR_O called

EXAMPLE:

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

.RETURN Function

TRAP FUNCTION: .RETURN - Return to 143Bug

CODE: $0063

DESCRIPTION: Used to return control to 143Bug 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 143Bug.

ENTRY CONDITIONS:

No arguments required.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

Control is returned to 143Bug.

EXAMPLE:

```
SYSCALL  .RETURN     Return to 143Bug.
```

NOTE: .RETURN must be used only by code that was started using 143Bug.

.BINDEC Function

TRAP FUNCTION: .BINDEC – Calculate BCD equivalent of the binary number specified

CODE: $0064

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

ENTRY CONDITIONS:

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

EXIT CONDITIONS DIFFERENT FROM ENTRY:

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

EXAMPLE:

```
SUBQ.L   #8,A7          Allocate space for result
MOVE.L   D0,-(SP)       Load hex number
SYSCALL  .BINDEC        Call .BINDEC
MOVE.L   (SP)+,D1/D2    Load result
```

.CHANGEV Function

TRAP FUNCTION: .CHANGEV – Parse value, assign to variable

CODE: $0067

DESCRIPTION: Attempt to parse value in user specified buffer. If user's buffer is empty, prompt user for new value, otherwise update integer offset into buffer to skip "value". Display new value and assign to variable unless user's input is an empty string.

ENTRY CONDITIONS:

SP ==> Address of 32-bit offset into user's buffer
Address of user's buffer (pointer/count format string)
Address of 32-bit integer variable to "change"
Address of string to use in prompting and displaying value

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Top of stack

EXAMPLE:

```
PROMPT    DC.B     $14,'COUNT = |10,8|'
GETCOUNT  PEA.L    PROMPT(PC)      Point to prompt string
          PEA.L    COUNT           Point to variable to change
          PEA.L    POINT           Point to offset into buffer
          PEA.L    BUFFER          Point to buffer
          SYSCALL  .CHANGEV        Make the system call
          RTS                      COUNT changed, return
```

NOTE: If the above code was called with BUFFER containing "1 3" in pointer/count format and POINT containing 2 (longword), then COUNT would be assigned the value 3, and POINT would contain 4 (pointing to first character past "3"). Note that POINT is the offset from the start address of the buffer (not the address of the first character in the buffer) to the next character to process. In this case, a value of 2 in POINT indicates that the space between "1" and "3" is the next character to be processed. After calling .CHANGEV, the screen displays the following line:

```
COUNT = 3
```

If the above code was called again, nothing could be parsed from BUFFER, so a prompt would be issued. For purpose of example, the string "5" is entered in response to the prompt.

```
COUNT = 3? 5
COUNT = 5
```

If in the previous example nothing had been entered at the prompt, COUNT would retain its prior value.

```
COUNT = 3? (CR)
COUNT = 3
```

.STRCMP Function

TRAP FUNCTION: .STRCMP – Compare two strings (pointer/count)

CODE: $0068

DESCRIPTION: Comparison for equality is made and boolean flag is returned to caller. The flag is $00 if the strings are not identical; otherwise it is $FF.

ENTRY CONDITIONS:

SP ==> Address of string 1
Address of string 2
Three bytes (unused)
Byte to receive string comparison result

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Three bytes (unused)
Byte to receive string comparison result

EXAMPLE: If A1 and A2 contain addresses of the two strings;

```
SUBQ.L    #4,SP        Allocate longword to receive result.
PEA.L     (A1)         Push address of one string.
PEA.L     (A2)         Push address of the other string.
SYSCALL   .STRCMP      Compare the strings.
MOVE.L    (SP)+,D0     Pop boolean flag into data register.
TST.B     D0           Check boolean flag.
BNE       ARE_SAME     Branch if strings are identical.
```

.MULU32 Function

.MULU32

TRAP FUNCTION: .MULU32 - Unsigned 32-bit x 32-bit multiply

CODE: $0069

DESCRIPTION: Two 32-bit unsigned integers are multiplied and the product is returned on the stack as a 32-bit unsigned integer. No overflow checking is performed.

ENTRY CONDITIONS:

SP ==> 32-bit multiplier
32-bit multiplicand
32-bit space for result

EXIT CONDITION DIFFERENT FROM ENTRY:

SP ==> 32-bit product (result from multiplication)

EXAMPLE: Multiply D0 by D1; load result into D2.

```
SUBQ.L   #4,SP          Allocate space for result.
MOVE.L   D0,-(SP)       Push multiplicand.
MOVE.L   D1,-(SP)       Push multiplier.
SYSCALL  .MULU32        Multiply D0 by D1.
MOVE.L   (SP)+,D2       Get product.
```

.DIVU32 Function

TRAP FUNCTION: .DIVU32 - Unsigned 32-bit x 32-bit divide

CODE: $006A

DESCRIPTION: Unsigned division is performed on two 32-bit integers and the quotient is returned on the stack as a 32-bit unsigned integer. The case of division by zero is handled by returning the maximum unsigned value $FFFFFFFF.

ENTRY CONDITIONS:

SP ==>	32-bit divisor	(value to divide by)
	32-bit dividend	(value to divide)
	32-bit space for result	

EXIT CONDITION DIFFERENT FROM ENTRY:

SP ==> 32-bit quotient (result from division)

EXAMPLE: Divide D0 by D1; load result into D2.

```
SUBQ.L   #4,SP          Allocate space for result.
MOVE.L   D0,-(SP)       Push dividend.
MOVE.L   D1,-(SP)       Push divisor.
SYSCALL  .DIVU32        Divide D0 by D1.
MOVE.L   (SP)+,D2       Get quotient.
```

.CHK_SUM Function

TRAP FUNCTION: .CHK_SUM – Generate checksum for address range

CODE: $006B

DESCRIPTION: This function generates a checksum for an address range that is passed in as arguments.

ENTRY CONDITIONS:

SP ==>	Beginning address	<long>
	Ending address + 1	<long>
	Space for checksum	<word>

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==>	Checksum	<word>

EXAMPLE:

```
CLR.W     -(SP)       Make room for the checksum.
PEA.L     A1          Push pointer to ending address + 1.
PEA.L     A0          Push pointer to starting address.
SYSCALL   .CHK_SUM    Invoke TRAP #15 call.
MOVE.W    (SP)+,D0    Load D0.W with checksum (EEOO)
                                MSB=even  LSB=odd
```

NOTES:
1. If a Bus Error results from this routine, then the bug bus error exception handler is invoked and the calling routine is also aborted.
2. The calling routine must insure that the beginning and ending addresses are on word boundaries or the integrity of the checksum cannot be guaranteed.

.BRD_ID Function

TRAP FUNCTION: .BRD_ID – Return pointer to board ID packet

CODE: $0070

DESCRIPTION: This routine returns a pointer on the stack to the 'board identification' packet. The packet is built at initialization time and contains information about the board and the peripherals it supports.

The format of the board identification packet is shown below:

$00	EYE CATCHER			
$04	REV	MONTH	DAY	YEAR
$08	PACKET SIZE		RESERVED	
$0C	BOARD NUMBER		BOARD SUFFIX	
$10	OPTIONS (COPROCESSOR, ETC.)		FAMILY	CPU
$14	CONTROLLER LUN		DEVICE LUN	
$18	DEVICE TYPE		DEVICE NUMBER	
$1C	BUG/SSID MODEM STRUCTURE POINTER			

Eye Catcher	Longword containing ASCII string "|ID|".
Rev	Byte contains Bug revision (in BCD).
Month, Day, Year	3 bytes contain date (in BCD) Bug was frozen.
Packet Size	Word contains the size of the packet.
Reserved	Reserved for future use.
Board Number	Word contains the board number (in BCD).
Board Suffix	Word contains the ASCII board suffix (XT, A, 20).

.BRD_ID

Options:

bits 0-3 — Four bits contain CPU type:

CPU = 1 ; MC68010 present
CPU = 2 ; MC68020 present
CPU = 3 ; MC68030 present
CPU = 4 ; MC68040 present

bits 4-6 — Three bits contain the Family type:

Fam = 0 ; 68xxx family
Fam = 1 ; 88xxx family

bits 7-31 — The remaining bits define various coprocessors presently defined:

Bit 7 set = FPC present
Bit 8 set = PMMU present
Bit 9 set = MMB present

SSID Pointer — Longword contains a pointer to the modem structure for Bug and SSID.

ENTRY CONDITIONS:

SP ==> Result <long> — Allocate space for ID packet address.

EXIT CONDITIONS DIFFERENT FROM ENTRY:

SP ==> Address <long> — Starting address of ID packet.

EXAMPLE:

```
CLR.L     -(SP)      Make room for returned pointer.
SYSCALL   .BRD_ID    Board ID trap call.
MOVE.L    (SP)+,A0   Get pointer off stack.
```

CHAPTER 6
143Bug DIAGNOSTIC FIRMWARE GUIDE

Scope

This diagnostic guide contains information about the operation and use of the MVME143 Diagnostic Firmware Package, hereafter referred to as "the diagnostics". The *System Start-up* and *Diagnostic Monitor* paragraphs give the user guidance in setting up the system and invoking the various utilities and tests. The *Utilities* paragraph describes utilities available to the user. The *MPU Tests for the MC68030* through *Z8530 Functionality Test - Command SCC* are guides to using each test.

Overview of Diagnostic Firmware

The MVME143 diagnostic firmware package consists of two 128K x 8 EPROMs which are installed on the MVME143. These two EPROMs (which also contain 143Bug) contain a complete diagnostic monitor along with a battery of utilities and tests for exercise, test, and debug of hardware in the MVME143 environment.

The diagnostics are menu-driven for ease of use. The Help (HE) command displays a menu of all available diagnostic functions (i.e., the tests and utilities). Several tests have a subtest menu which may be called using the HE command. In addition, some utilities have subfunctions, and as such have subfunction menus.

System Start-Up

For 143Bug diagnostics to operate properly, follow this setup procedure.

> **CAUTION**
>
> **Connecting modules while power is applied may result in damage to components on the module.**

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

b. Refer to the *MVME143 MPU VMEmodule User's Manual* and configure header J1 to enable or disable the system controller functions of the MVME143 as required for the user's particular application.

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

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

e. Connect the terminal which is to be used as the 143Bug system console to connector J9 (DB25 connector) (port 1) on the front panel. Set up the terminal as follows:

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

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

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

g. Power up the system. 143Bug executes some self-checks and displays the debugger prompt ("143-Bug>") (refer to the *Monitor Start-up* paragraph in this chapter).

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

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

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

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

The firmware monitor comes up with the FAIL LED on.

Diagnostic Monitor

The tests described herein are called via a common diagnostic monitor, hereafter called monitor. This monitor is command-line driven and provides input/output facilities, command parsing, error reporting, interrupt handling, and a multi-level directory.

Monitor Start-Up

When the monitor is first brought up, following power-up or switch RESET, the following is displayed on the diagnostic video display terminal (port 1 terminal):

```
Copyright Motorola Inc. 1988, All Rights Reserved

VME143 Monitor/Debugger Release 1.0 - 4/08/88

COLD Start
143-Bug>
```

If after a delay, the 143Bug begins to display test result messages on the bottom line of the screen in rapid succession, the MVME143 is in the Bug "system" mode. If this is not the desired mode of operation, then press the ABORT switch. When the menu id displayed, enter a 3 to go to the system debugger. The environment may be changed by using the Set Environment (ENV) command. Refer to Appendix A for details of Bug operation in the system mode.

At the prompt, enter SD to switch to the diagnostics directory. The Switch Directories (SD) command is described elsewhere in this chapter. The prompt should now read "143-Diag>".

Command Entry and Directories

Entry of commands is made when the prompt "143-Diag>" appears. The name (mnemonic) for the command is entered before pressing the carriage return (CR). Multiple commands may be entered. If a command expects parameters and another command is to follow it, separate the two with an exclamation point (!). For instance, to invoke the command MT B after the command MT A, the command line would read MT A ! MT B. Spaces are not required but are shown here for legibility. Several commands may be combined on one line.

Several commands consist of a command name that is listed in a main (root) directory and a subcommand that is listed in the directory for that particular command. In the main directory are commands like MPU and CA30. These commands are used to refer to a set of lower level commands.

To call up a particular MPU test, enter (on the same line) MPU A. This command causes the monitor to find the MPU subdirectory, and then to execute the command "A" from that subdirectory.

Examples:

Single-Level Commands	HE		Help
	DE		Display Error Counters
Two-Level Commands	MPU		MPU Tests for the MC68030
		A	Register Test
	CA30		MC68030 Onchip Cache Tests
		G	Unlike Instruction Function Codes

Help - Command HE

Online documentation has been provided in the form of a Help command (syntax: **HE** [*command name*]). This command displays a menu of the top level directory if no parameters are entered, or a menu of each subdirectory if the name of that subdirectory is entered. (The top level directory lists (Dir) after the name of each command that has a subdirectory.) For example, to bring up a menu of all the memory tests, enter HE MT. When a menu is too long to fit on the screen, it pauses until the operator presses the carriage return, (CR), again.

Self Test - Prefix/Command ST

The monitor provides an automated test mechanism called self test. Entering **ST** + *command* causes the monitor to run only the tests included in that command. Entering **ST** - *command* runs all the tests included in an internal self-test directory except the command listed. ST without any parameters runs the entire directory, which contains most of the MVME143 diagnostics.

Each test for that particular command is listed in the paragraph pertaining to the command.

When in "system" mode, and Bug has been invoked, the suite of extended confidence tests that are run at system mode start up can be executed from Bug. This is done with the SST command. This is useful for debugging board failures that may require toggling between the test suite and Bug. Upon completion of running the test suite, the Bug prompt is displayed, ready for other commands. For details on extended confidence test operation, refer to Appendix A, *Bug System Mode Operation*.

Switch Directories - Command SD

To leave the diagnostic directory (and disable the diagnostic tests), enter SD. At this point, only the commands for 143Bug function. When in the 143Bug directory, the prompt reads 143-Bug>. To return to the diagnostic directory, the command SD is entered again. When in the diagnostic directory, the prompt reads 143-Diag>. The purpose of this feature is to allow the user to access 143Bug without the diagnostics being visible.

Loop-On-Error Mode - Prefix LE

Occasionally, when an oscilloscope or logic analyzer is in use, it becomes desirable to endlessly repeat a test at the point where an error is detected. LE accomplishes that for most of the tests. To invoke LE, enter it before the test that is to run in loop-on-error mode.

Stop-On-Error Mode - Prefix SE

It is sometimes desirable to stop a test or series of tests at the point where an error is detected. SE accomplishes that for most of the tests. To invoke SE, enter it before the test or series of tests that is to run in stop-on-error mode.

Loop-Continue Mode - Prefix LC

To endlessly repeat a test or series of tests, the prefix LC is entered. This loop includes everything on the command line. To break the loop, press the BREAK key on the diagnostic video display terminal. Certain tests disable the BREAK key interrupt; so, pressing the ABORT or RESET switches on the MVME143 front panel may become necessary.

Non-Verbose Mode - Prefix NV

Upon detecting an error, the tests included for the MVME143 display a substantial amount of data. To avoid the necessity of watching the scrolling display, a mode is provided that suppresses all messages except PASSED or FAILED. This mode is called non-verbose and is invoked prior to calling a command by entering NV. NV ST MT would cause the monitor to run the MT self-test, but show only the names of the subtests and the results (pass/fail).

Display Error Counters – Command DE

Each test or command in the diagnostic monitor has an individual error counter. As errors are encountered in a particular test, that error counter is incremented. If one were to run a self-test or just a series of tests, the results could be broken down as to which tests passed by examining the error counters. DE displays the results of a particular test if the name of that test follows DE. Only nonzero values are displayed.

Clear (Zero) Error Counters – Command ZE

The error counters originally come up with the value of zero, but it is occasionally desirable to reset them to zero at a later time. This command resets all of the error counters to zero. The error counters can be individually reset by entering the specific test name following the command. Example: ZE MPU A clears the error counter associated with MPU A.

Display Pass Count – Command DP

A count of the number of passes in loop-continue mode is kept by the monitor. This count is displayed with other information at the conclusion of each pass. To display this information without using LC, enter DP.

Zero Pass Count – Command ZP

Invoking this command resets the pass counter DP to zero. This is frequently desirable before typing in a command that invokes the loop-continue mode. Entering this command on the same line as LC results in the pass counter being reset every pass.

Utilities

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

Write Loop – Command WL.size

The WL.size command invokes a streamlined write of specified size to the specified memory location. This command is intended as a technician aid for debug once specific fault areas are identified. The write loop is very short in execution so that measuring devices such as oscilloscopes may be utilized in tracking failures. Pressing the BREAK key does not stop the command, but pressing the ABORT switch or RESET switch does.

Command size must be specified as B for byte, W for word, or L for longword.

The command requires two parameters: target address and data to be written. The address and data are both hexadecimal values and must be preceded by a $ if the first digit is other than 0–9 (i.e., $FF would be entered as $FF). To write $00 out to address $00010000, enter WL.B $00010000 00. Omission of either or both parameters causes prompting for the missing values.

Read Loop – Command RL.size

The RL.size command invokes a streamlined read of specified size from the specified memory location. This command is intended as a technician aid for debug once specific fault areas are identified. The read loop is very short in execution so that measuring devices such as oscilloscopes may be utilized in tracking failures. Pressing the BREAK key does not stop the command, but pressing the ABORT switch or RESET switch does.

Command size must be specified as B for byte, W for word, or L for longword.

The command requires one parameter: target address. The address is a hexadecimal value. To read from address $00010000, enter RL.B $00010000. Omission of the parameter causes prompting for the missing value.

Write/Read Loop – Command WR.size

The WR.size command invokes a streamlined write and read of specified size to the specified memory location. This command is intended as a technician aid for debug once specific fault areas are identified. The write/read loop is very short in execution so that measuring devices such as oscilloscopes may be utilized in tracking failures. Pressing the BREAK key does not stop the command, but pressing the ABORT switch or RESET switch does.

Command size must be specified as B for byte, W for word, or L for longword.

The command requires two parameters: target address and data to be written. The address and data are both hexadecimal values and must be preceded by a $ if the first digit is other than 0–9 (i.e., $FF would be entered as $FF. To write $00 out to address $00010000, enter WR.B $00010000 00). Omission of either or both parameters causes prompting for the missing values.

MPU Tests For The MC68030 – Command MPU

The following paragraphs describe the MPU tests for the MC68030.

General Description

This paragraph details the diagnostics provided to test the MC68030 MPU, as listed in Table 6–1.

TABLE 6–1. MC68030 MPU Diagnostic Tests

MONITOR COMMAND	TITLE
MPU A	Register Test
MPU B	Instruction Test
MPU C	Address Mode Test
MPU D	Exception Processing Test

The normal procedure for fixing an MC68030 MPU error is to replace the MPU.

Hardware Configuration

The following hardware is required to perform these tests:

MVME143 – Module being tested
VME chassis
Video display terminal

MPU A – Register Test

MPU A

The following paragraphs describe the MPU A register test.

Description

This command does a thorough test of all the registers in the MC68030 chip, including checking for bits stuck high or low.

Command Input

```
143-Diag>MPU A
```

Response/Messages

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

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


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

```
A     MPU Register test.....................Running ---------->..... FAILED
(error message)
```

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

```
Failed D0-D7 register check
Failed SR register check
Failed USP/VBR/CAAR register check
Failed CACR register check
Failed A0-A4 register check
Failed A5-A7 register check
```

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

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

MPU B – Instruction Test

The MPU B instruction test is described in the following paragraphs.

Description

This command tests various data movement, integer arithmetic, logical, shift and rotate, and bit manipulation instructions of the MC68030 chip.

Command Input

```
143-Diag>MPU B
```

Response/Messages

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

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

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

```
B     MPU Instruction Test.....................Running ---------->..... FAILED
(error message)
```

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

```
Failed AND/OR/NOT/EOR instruction check
Failed DBF instruction check
Failed ADD or SUB instruction  check
Failed MULU or DIVU instruction check
Failed BSET or BCLR instruction check
Failed LSR instruction check
Failed LSL instruction check
Failed BFSET or BFCLR instruction check
Failed BFCHG or BFINS instruction check
Failed BFEXTU instruction check
```

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

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

MPU C – Address Mode Test

The MPU C address mode test is described in the following paragraphs.

Description

This command tests the various addressing modes of the MC68030 chip. These include absolute address, address indirect, address indirect with postincrement, and address indirect with index modes.

Command Input

```
143-Diag>MPU C
```

Response/Messages

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

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

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

```
C     MPU Address Mode test.....................Running ---------->..... FAILED
(error message)
```

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

```
Failed Absolute Addressing check
Failed Indirect Addressing check
Failed Post increment check
Failed Pre decrement check
Failed Indirect Addressing with Index check
Unexpected Bus Error at $XXXXXXXX
```

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

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

MPU D - Exception Processing Test

The MPU D exception processing test is described in the following paragraphs.

Description

This command tests many of the exception processing routines of the MC68030, but not the interrupt auto vectors or any of the floating point coprocessor vectors.

Command Input

```
143-Diag>MPU D
```

Response/Messages

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

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

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

```
D     MPU Exception Processing Test.....Running ---------->..... FAILED
Test Failed Vector # XXX
```

Here # XXX is the hexadecimal exception vector offset, as explained in the *MC68030 32-Bit Microprocessor User's Manual*, MC68030UM.

However, if the failure involves taking an exception different from that being tested, the display is:

```
D     MPU Exception Processing Test.....Running ---------->..... FAILED
Unexpected exception taken to Vector # XXX
```

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

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

MC68030 Onchip Cache Tests – Command CA30

The MC68030 onchip cache tests are described in the following paragraphs.

General Description

This paragraph details the diagnostics provided to test the MC68030 cache, as listed in Table 6-2.

TABLE 6-2. MC68030 Cache Diagnostic Tests

MONITOR COMMAND	TITLE
CA30 A	Basic data caching test
CA30 B	D cache tag RAM test
CA30 C	D cache data RAM test
CA30 D	D cache valid flags test
CA30 E	D cache burst fill test
CA30 F	Basic instruction caching test
CA30 G	Unlike instruction function codes test
CA30 H	I cache disable test
CA30 I	I cache clear test

The normal procedure for fixing an MC68030 cache error is to replace the MPU.

Hardware Configuration

The following hardware is required to perform these tests:

MVME143 – Module being tested
VME chassis
Video display terminal.

CA30 A – Basic Data Caching Test

The CA30 A basic data caching test is described in the following paragraphs.

Description

This test checks out the basic caching function by deliberately causing stale data, then reading the corresponding locations. Failure is declared if the cache misses or if any value is read that is not what is expected to be in the cache.

The test is meant only to provide a gross functional check of the cache. Its purpose is to verify that each entry in the cache latches and holds data independently of the other entries. It does not check for bad bits in the tag RAM, valid flags, or data RAM other than what is required to cache a simple pattern.

Command Input

```
143-Diag>CA30 A
```

Response/Messages

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

```
A     Basic data caching .....................Running ---------->
```

If there are any cache misses, then the test fails and the display appears as follows.

```
A     Basic data caching .....................Running ---------->
CACHE MISSED! ..... FAILED
```

If there are no cache misses, then the test passes.

```
A     Basic data caching .....................Running ----------> PASSED
```

CA30 B – Data Cache Tag RAM Test

The CA30 B data cache tag RAM test is described in the following paragraphs.

Description

This test verifies the data cache tag RAM by caching accesses to locations that cause a variety of values to be written into the tag RAM. The test addresses and function codes are translated by the onchip MMU to select locations physically in the onboard RAM. The criterion for passing the test is hitting in the cache on a read from a test location following a cacheable write access to that same location. The data in the cache has been made stale between the cacheable write and the following read to provide the distinction between hits and misses. The failure of the tag RAM to properly latch a tag should cause the cache to miss when it should hit.

The MMU is used to allow a wide variety of values to be used for the tags without having read/write memory at each location corresponding to those tags. This relies on the cache being logical as opposed to physical.

To allow the test to be run either out of ROM or RAM, the onboard resources and the bottom 16Mb of the addressing space are mapped transparently for supervisor mode accesses. This allows debugging of the test with full access to the monitor/debugger facilities, too. This requires that the test addresses that fall in these two ranges not bear supervisor function codes.

The translation of the test addresses and function codes is implemented by setting the CRP descriptor type to "early termination" and loading an offset into the CRP table address field. This offset is the distance from the test address to a location in the test area and is added to every logical address that does not qualify for transparent translation (refer to following paragraph).

The 16Mb transparently mapped areas are described by the Transparent Translation (TT) registers. TT0 matches addresses $00XXXXXX while TT1 matches addresses $FFXXXXXX. Both are qualified such that the function code must specify supervisor mode. The purpose in this is to force test addresses that fall in either range to be offset instead of transparently mapped. The test addresses have been selected to avoid matching both address and function code with either TT register.

CA30 B

Command Input

```
143-Diag>CA30 B
```

Response/Messages

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

```
B     D cache tag RAM .......................Running ---------->
```

If there are any cache misses, then the test fails and the display appears as follows.

```
B     D cache tag RAM .......................Running ---------->
CACHE MISSED!     FAILED
```

If there are no cache misses, then the test passes.

```
B     D cache tag RAM .......................Running ----------> PASSED
```

CA30 C - Data Cache Data RAM Test

CA30 C

The CA30 C data cache data RAM test is described in the following paragraphs.

Description

This is essentially a memory test. For each entry in the cache, several values are cached, made stale, then read back. Failure is declared if the value written (and supposedly cached) differs from that read later from the same location. No distinction is made between cache misses and genuine bad data; the only concern here is the latching of the data.

Command Input

```
143-Diag>CA30 C
```


Response/Messages

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

```
C     D cache data RAM test ..................Running ---------->
```

If there are any cache misses, then the test fails and the display appears as follows.

```
C     D cache data RAM test ..................Running ---------->
CACHE MISSED!     FAILED
```

If there are no cache misses, then the test passes.

```
C     D cache data RAM test ..................Running ----------> PASSED
```

CA30 D – Data Cache Valid Flags Test

CA30 D

The CA30 D data cache valid flags test is described in the following paragraphs.

Description

This test verifies that each valid flag is set when its entry is valid and cleared either when the entire cache is flushed or an individual line is cleared. This test does check for side effects on other valid flags.

This test is actually two inline subtests: V FLAG CLEAR and V FLAG SET. The former checks that each valid flag can be individually cleared while the latter checks for the opposite.

Command Input

```
143-Diag>CA30 D
```

Response/Messages

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

```
D     D cache valid flags test ..............Running ---------->
```

If there are any cache hits when clearing valid flags, then the test fails and the display appears as follows:

```
D     D cache valid flags test ..............Running ---------->
VALID BIT CLEAR SUBTEST - EXPECTED MISS ..... FAILED
```

If there are any cache misses when setting valid flags, then the test fails and the display appears as follows:

```
D     D cache valid flags test ..............Running ---------->
VALID BIT SET   SUBTEST - EXPECTED HIT ..... FAILED
```

If all flags are valid, then the test passes.

```
D     D cache valid flags test ..............Running ----------> PASSED
```

CA30 E – Data Cache Burst Fill Test

The CA30 E data cache burst fill test is described in the following paragraphs.

Description

This test checks the burst fill mechanism. The criterion for passing this test is hitting in the cache for all entries in a line after one entry in that line has been cached. Each line is tested. The entry read to cause the filling is varied from line to line.

Command Input

```
143-Diag>CA30 E
```

Response/Messages

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

```
E     D cache burst fill test ..................Running ---------->
```

If there are any cache misses, then the test fails and the display appears as follows.

```
E     D cache burst fill test ..................Running ---------->
CACHE MISSED!     FAILED
```

If there are no cache misses, then the test passes.

```
E     D cache burst fill test ..................Running ----------> PASSED
```

CA30 F – Basic Instruction Caching Test

CA30 F

The CA30 F basic instruction caching test is described in the following paragraphs.

Description

This command tests the basic caching function of the MC68030 microprocessor. The test caches a program segment that resides in RAM, freezes the cache, changes the program segment in RAM, then reruns the program segment. If the cache is functioning correctly, the cached instructions are executed. Failure is detected if the MC68030 executes the instructions that reside in RAM; any cache misses cause an error.

The process is first attempted in supervisor mode for both the initial pass through the program segment and the second pass. It is then repeated, using user mode for the initial pass and the second pass. A bit is included in each cache entry for distinguishing between supervisor and user mode. If this bit is stuck or inaccessible, the cache misses during one of these two tests.

Command Input

```
143-Diag>CA30 F
```

Response/Messages

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

```
F     Basic instr. caching ..................Running ---------->
```

If there are any cache misses during the second pass through the program segment, then the test fails and the display appears as follows.

```
F     Basic instr. caching ..................Running ---------->..... FAILED
2 CACHE MISSES!
CACHED IN SUPY MODE, RERAN IN SUPY MODE
```

If there are no cache misses during the second pass, then the test passes.

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

CA30 G – Unlike Instruction Function Codes Test

CA30 G

The CA30 G unlike instruction function codes test is described in the following paragraphs.

Description

This command tests the ability of the onchip cache to recognize instruction function codes. Bit 2 of the function code is included in the tag for each entry. This provides a distinction between supervisor and user modes for the cached instructions. To test this mechanism, a program segment that resides in RAM is cached in supervisor mode. The cache is frozen, then the program segment in RAM is changed. When the program segment is executed a second time in user mode, there should be no cache hits due to the different function codes. Failure is detected if the MC68030 executes the cached instructions.

After the program segment has been cached in supervisor mode and rerun in user mode, the process is repeated, caching in user mode and rerunning in supervisor mode. Again, the cache should miss during the second pass through the program segment.

Command Input

```
143-Diag>CA30 G
```

Response/Messages

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

```
G     Unlike instr. fn. codes ..................Running ---------->
```

If there are any cache hits during the second pass through the program segment, then the test fails and the display appears as follows.

```
G     Unlike instr. fn. codes ..................Running ---------->..... FAILED
5 CACHE HITS!
CACHED IN SUPY MODE, RERAN IN USER MODE
```

If there are no cache hits during the second pass, then the test passes.

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

CA30 H - Disable Test

The CA30 H disable test is described in the following paragraphs.

Description

In the MC68030 Cache Control Register (CACR) a control bit is provided to enable the cache. When this bit is clear, the cache should never hit, regardless of whether the address and function codes match a tag. To test this mechanism, a program segment is cached from RAM. The cache is frozen to preserve its contents, then the enable bit is cleared. The program segment in RAM is then changed and rerun. There should be no cache hits with the enable bit clear. Failure is declared if the cache does hit.

Command Input

```
143-Diag>CA30 H
```

Response/Messages

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

```
H     I cache disable test .....Running ---------->
```

If there are any cache hits during the second pass through the program segment, then the test fails and the display appears as follows.

```
H     I cache disable test .....Running ---------->..... FAILED
1 CACHE HIT!
CACHED IN SUPY MODE, RERAN IN SUPY MODE
```

If there are no cache hits during the second pass, then the test passes.

```
H     I cache disable test .....Running ----------> PASSED
```

CA30 I - Clear Test

The CA30 I clear test is described in the following paragraphs.

Description

A control bit is included in the MC68030 CACR to clear the cache. Writing a one to this bit invalidates every entry in the onchip cache. To test this function, a program segment in RAM is cached and then frozen there to preserve it long enough to activate the cache clear control bit. The program segment in RAM is then modified and rerun with the cache enabled. If the cache hits, the clear is incomplete and failure is declared.

Command Input

```
143-Diag>CA30 I
```


Response/Messages

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

```
I     I cache clear test .....................Running ---------->
```

If there are any cache hits during the second pass through the program segment, then the test fails and the display appears as follows.

```
I     I cache clear test .....................Running ---------->..... FAILED
58 CACHE HITS!
CACHED IN SUPY MODE, RERAN IN SUPY MODE
```

If there are no cache hits during the second pass, then the test passes.

```
I     I cache clear test .....................Running ----------> PASSED
```

Memory Tests - Command MT

The memory tests are described in the following paragraphs.

General Description

This set of tests accesses random access memory (read/write) that may or may not reside on the MVME143 module. Default is the onboard RAM. To test offboard RAM, change Start and Stop Addresses per MT B and MT C as described in the following paragraphs. Memory tests are listed in Table 6-3.

NOTE

If one or more memory tests are attempted at an address where there is no memory, a bus error message appears, giving the details of the problem.

TABLE 6-3. Memory Diagnostic Tests

MONITOR COMMAND	TITLE
MT FP	MEM Bd: Fast Pattern Test
MT FA	MEM Bd: Fast Addr. Test
MT FV	MEM Bd: Fast VMEbus W/R Test
MT A	Set Function Code
MT B	Set Start Address
MT C	Set Stop Address
MT D	Set Bus Data Width
MT E	March Address Test
MT F	Walk a Bit Test
MT G	Refresh Test
MT H	Random Byte Test
MT I	Program Test
MT J	TAS Test
MT K	Brief Parity Test
MT L	Extended Parity Test
MT M	Nibble Mode Test

Hardware Configuration

The following hardware is required to perform these tests.

MVME143 – Module being tested
VME chassis
Video display terminal
Optional offboard memory.

MT A – Set Function Code

MT A

The set function code command (MT A) is described in the following paragraphs.

Description

This command allows the user to select the function code used in most of the memory tests. The exceptions to this are Program Test and TAS Test.

Command Input

```
143-Diag>MT A [new value]
```

Response/Messages

If the user supplied the optional new value, then the display appears as follows:

```
143-Diag>MT A [new value]
Function Code=<new value>
143-Diag>
```

If a new value was not specified by the user, then the old value is displayed and the user is allowed to enter a new value.

NOTE

The default is Function Code=5, which is for onboard RAM.

```
143-Diag>MT A
Function Code=<current value> ?[new value]
Function Code=<new value>
143-Diag>
```

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

```
143-Diag>MT A
Function Code=<current value> ?(CR)
Function Code=<current value>
143-Diag>
```

MT B – Set Start Address

MT B

The set start address command (MT B) is described in the following paragraphs.

Description

This command allows the user to select the start address used by all of the memory tests. For the MVME143, it is suggested that address $00004000 be used. Other addresses may be used, but extreme caution should be used when attempting to test memory below this address.

Command Input

```
143-Diag>MT B [new value]
```

Response/Messages

If the user supplied the optional new value, then the display appears as follows:

```
143-Diag>MT B [new value]
Start Addr.=<new value>
143-Diag>
```

If a new value was not specified by the user, then the old value is displayed and the user is allowed to enter a new value.

NOTE

The default is Start Addr.=00004000, which is for onboard RAM.

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

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

```
143-Diag>MT B
Start Addr.=<current value> ?(CR)
Start Addr.=<current value>
143-Diag>
```

NOTE

If a new value is specified, it is truncated to a longword boundary and, if greater than the value of the stop address, replaces the stop address. The start address is never allowed to be higher in memory than the stop address. These changes occur before another command is processed by the monitor.

MT C – Set Stop Address

The set stop address command (MT C) is described in the following paragraphs.

Description

This command allows the user to select the stop address used by all of the memory tests.

Command Input

```
143-Diag>MT C [new value]
```

Response/Messages

If the user supplied the optional new value, then the display appears as follows:

```
143-Diag>MT C [new value]
Stop Addr.=<new value>
143-Diag>
```


If a new value was not specified by the user, then the old value is displayed and the user is allowed to enter a new value.

NOTE

The default is Stop Addr.=DRAMsize–4, which is the end of onboard RAM.

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

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

```
143-Diag>MT C
Start Addr.=<current value> ?(CR)
Start Addr.=<current value>
143-Diag>
```

NOTE

If a new value is specified, it is truncated to a longword boundary and, if less than the value of the start address, is replaced by the start address. The stop address is never allowed to be lower in memory than the start address. These changes occur before another command is processed by the monitor.

MT D – Set Bus Data Width

The set bus data width command (MT D) is described in the following paragraphs.

Description

This command is used to select either 16-bit or 32-bit bus data accesses during the MVME143Bug MT memory tests. The width is selected by entering zero for 16 bits or one for 32 bits.

Command Input

```
143-Diag>MT D [new value: 0 for 16, 1 for 32]
```

Response/Messages

If the user supplied the optional new value, then the display appears as follows:


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

If a new value was not specified by the user, then the old value is displayed and the user is allowed to enter a new value.

NOTE: The default value is Bus Width (32=1/16=0) =0.

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

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

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

MT E – March Address Test

MT E

The march address test command (MT E) is described in the following paragraphs.

Description

This command performs a march address test from Start Address to Stop Address.

The march address test has been implemented in the following manner:

Step 1. All memory locations from Start Address up to Stop Address are cleared to 0.

Step 2. Beginning at Stop Address and proceeding downward to Start Address, each memory location is checked for bits that did not clear and then the contents are changed to all F's (all the bits are set). This process reveals address lines that are stuck high.

Step 3. Beginning at Start Address and proceeding upward to Stop Address, each memory location is checked for bits that did not set and then the memory location is again cleared to 0. This process reveals address lines that are stuck low.

Command Input

```
143-Diag>MT E
```

Response/Messages

After the command is entered, the display should appear as follows:

```
E     MT March Addr. Test.....................Running ---------->
```

If an error is encountered, then the memory location and other related information are displayed.

```
E     MT March Addr. Test.....................Running ---------->..... FAILED
```

(error-related information)

If no errors are encountered, then the display appears as follows:

```
E     MT March Addr. Test.....................Running ----------> PASSED
```

MT F - Walk a Bit Test

The walk a bit test command (MT F) is described in the following paragraphs.

Description

This command performs a walking bit test from start address to stop address.

The walking bit test has been implemented in the following manner:

Step 1. For each memory location, do the following:

Write out a 32-bit value with only the lower bit set.

Read it back and verify that the value written equals the one read. Report any errors.

Shift the 32-bit value to move the bit up one position.

Repeat the procedure (write, read, and verify) for all 32-bit positions.

Command Input

```
143-Diag>MT F
```

Response/Messages

After the command is entered, the display should appear as follows:

```
F     MT Walk a bit Test .....................Running ---------->
```

If an error is encountered, then the memory location and other related information are displayed.

```
F     MT Walk a bit Test .....................Running ---------->..... FAILED
```

(error-related information)

If no errors are encountered, then the display appears as follows:

```
F     MT Walk a bit Test .....................Running ----------> PASSED
```

MT G – Refresh Test

The refresh test command (MT G) is described in the following paragraphs.

Description

This command performs a refresh test from Start Address to Stop Address.

The refresh test has been implemented in the following manner:

Step 1. For each memory location:

Write out value $FC84B730.

Verify that the location contains $FC84B730.

Proceed to next memory location.

Step 2. Delay for 500 milliseconds (1/2 second).

Step 3. For each memory location:

Verify that the location contains $FC84B730.

Write out the complement of $FC84B730 ($037B48CF).

Verify that the location contains $037B48CF.

Proceed to next memory location.

Step 4. Delay for 500 milliseconds.

Step 5. For each memory location:

Verify that the location contains $037B48CF.

Write out value $FC84B730.

Verify that the location contains $FC84B730.

Proceed to next memory location.

MT G

Command Input

```
143-Diag>MT G
```

Response/Messages

After the command is entered, the display should appear as follows:

```
G     MT Refresh Test.....................Running ---------->
```

If an error is encountered, then the memory location and other related information are displayed.

```
G     MT Refresh Test.....................Running ---------->..... FAILED
```

(error-related information)

If no errors are encountered, then the display appears as follows:

```
G     MT Refresh Test.....................Running ----------> PASSED
```

MT H – Random Byte Test

MT H

The random byte test command (MT H) is described in the following paragraphs.

Description

This command performs a random byte test from Start Address to Stop Address.

The random byte test has been implemented in the following manner:

Step 1. A register is loaded with the value $ECA86420.

Step 2. For each memory location:

Copy the contents of the register to the memory location, one byte at a time.

Add $02468ACE to the contents of the register.

Proceed to next memory location.

Step 3. Reload $ECA86420 into the register.

Step 4. For each memory location:

Compare the contents of the memory to the register to verify that the contents are good, one byte at a time.

Add $02468ACE to the contents of the register.

Proceed to next memory location.

Command Input

```
143-Diag>MT H
```

Response/Messages

After the command is entered, the display should appear as follows:

```
H     MT Random Byte Test.....................Running ---------->
```

If an error occurs, then the memory location and other related information are displayed.

```
H     MT Random Byte Test.....................Running ---------->..... FAILED
```

(error-related information)

If no errors occur, then the display appears as follows:

```
H     MT Random Byte Test.....................Running ----------> PASSED
```

MT I – Program Test

MT I

The program test command (MT I) is described in the following paragraphs.

Description

This command moves a program segment into RAM and executes it. The implementation of this is as follows:

Step 1. The program is moved into the RAM, repeating it as many times as necessary to fill the available RAM (i.e., from Start Address to Stop Address–8). Only complete segments of the program are moved. The space remaining from the last program segment copied into the RAM to Stop Address–8 is filled with NOP instructions. Attempting to run this test without sufficient memory (around 400 bytes) for at least one complete program segment to be copied causes an error message to be printed out: INSUFFICIENT MEMORY.

Step 2. The last location, Stop Address, receives an RTS instruction.

Step 3. Finally, the test performs a JSR to location Start Address.

Step 4. The program itself performs a wide variety of operations, with the results frequently checked and a count of the errors maintained. Errant locations are reported in the same fashion as any memory test failure.

Command Input

```
143-Diag>MT I
```

Response/Messages

After the command is entered, the display should appear as follows:

```
I     MT Program Test.....................Running ---------->
```

If the operator has not allowed enough memory for at least one program segment to be copied into the target RAM, then the following error message is printed. To avoid this, make sure that the Stop Address is at least 388 bytes ($00000184) greater than the Start Address.

```
I     MT Program Test.....................Running ---------->
      Insufficient Memory
      PASSED
```

If the program (in RAM) detects any errors, then the location of the error and other information is displayed.

```
I     MT Program Test.....................Running ---------->..... FAILED
(error-related information)
```

If no errors occur, then the display appears as follows:

```
I     MT Program Test.....................Running ----------> PASSED
```

MT J – TAS Test

MT J

The TAS test command (MT J) is described in the following paragraphs.

Description

This command performs a Test and Set (TAS) test from Start Address to Stop Address.

The test is implemented as follows:

Step 1. For each memory location:

Clear the memory location to 0.

Test And Set the location (should set upper bit only).

Verify that the location now contains $80.

Proceed to next location (next byte).

Command Input

```
143-Diag>MT J
```

Response/Messages

After the command is entered, the display should appear as follows:

```
J     MT TAS Test.............................Running ---------->
```

If an error occurs, then the memory location and other related information are displayed.

```
J     MT TAS Test.............................Running ---------->..... FAILED
```

(error-related information)

If no errors occur, then the display appears as follows:

```
J     MT TAS Test.............................Running ----------> PASSED
```

MT K – Brief Parity Test

The brief parity test command (MT K) is described in the following paragraphs.

Description

Test parity checking ability on longwords only from Start Address to Stop Address.

The brief parity test is implemented in the following manner:

Step 1. For each longword memory location:

Copy the contents of the memory location to a register, without parity enabled.

Enable parity checking and incorrect party generation.

Copy the contents of the register to the memory location, generating incorrect parity.

Disable incorrect parity generation.

Copy the contents of the memory location to a second register, generating a parity error.

Copy the contents of the register to the memory location, generating correct parity.

Copy the contents of the memory location to a second register, no parity error should occur.

Proceed to next memory location.

Step 2. Disable parity checking.

Command Input

```
143-Diag>MT K
```

Response/Messages

After the command is entered, the display should appear as follows:

```
K     MT Brief parity test ..................Running ---------->
```

If an error occurs, then the memory location and other related information are displayed.

```
K     MT Brief parity test ..................Running ---------->..... FAILED
```

(error-related information)

If no errors occur, then the display appears as follows:

```
K     MT Brief parity test ..................Running ----------> PASSED
```

MT L – Extended Parity Test

The extended parity test command (MT L) is described in the following paragraphs.

Description

Test parity checking ability on byte boundaries from Start Address to Stop Address.

The extended parity test is implemented in the following manner:

Step 1. For each byte memory location:

Copy the contents of the memory location to a register, without parity enabled.

Enable parity checking and incorrect party generation.

Copy the contents of the register to the memory location, generating incorrect parity.

Disable incorrect parity generation.

Copy the contents of the memory location to a second register, generating a parity error.

Copy the contents of the register to the memory location, generating correct parity.

Copy the contents of the memory location to a second register, no parity error should occur.

Proceed to next memory location.

Step 2. Disable parity checking.

Command Input

```
143-Diag>MT L
```

Response/Messages

After the command is entered, the display should appear as follows:

```
L     MT Extended parity test .................Running ---------->
```

If an error occurs, then the memory location and other related information are displayed.

```
L     MT Extended parity test .................Running ---------->..... FAILED
```

(error-related information)

If no errors occur, then the display appears as follows:

```
L     MT Extended parity test .................Running ----------> PASSED
```

MT M – Nibble Mode Test

The nibble mode test command (MT M) is described in the following paragraphs.

Description

This command tests the nibble mode of memory devices by performing RAM program execution test with instruction cache and burst mode enabled.

The test is implemented as follows:

Step 1. The program is moved into the RAM, repeating it as many times as necessary to fill the available RAM (i.e., from Start Address to Stop Address–8). Only complete segments of the program are moved. The space remaining from the last program segment copied into the RAM to Stop Address–8 is filled with NOP instructions. Attempting to run this test without sufficient memory (around 400 bytes) for at least one complete program segment to be copied causes an error message to be printed out: INSUFFICIENT MEMORY.

Step 2. The last location, Stop Address, receives an RTS instruction.

Step 3. Finally, the test performs a JSR to location Start Address.

Step 4. The program itself performs a wide variety of operations, with the results frequently checked and a count of the errors maintained. Errant locations are reported in the same fashion as any memory test failure (refer to the *Description of Memory Error Display Format* paragraph that follows).

Command Input

```
143-Diag>MT M
```

Response/Messages

After the command is entered, the display should appear as follows:

```
L     MT Nibble mode test.....................Running ---------->
```

If an error occurs, then the memory location and other related information are displayed.

```
L     MT Nibble mode test.....................Running ---------->..... FAILED
```

(error-related information)

If no errors occur, then the display appears as follows:

```
M     MT Nibble mode test.....................Running ----------> PASSED
```

Description of Memory Error Display Format

This paragraph is included to describe the format used to display errors during memory test E through M.

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

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

b. Upon 20 memory errors, the printing of error messages ceases for the remainder of the test.

The following is an example of the display format:

```
FC  TEST ADDR  10987654321098765432109876543210  EXPECTED  READ
 5  00010000   ----------------------X--------  00000100  00000000
 5  00010004   --------------------X-------X----  FFFFEFFF  FFFFFFEF
```

Each line displayed consists of five items: function code, test address, graphic bit report, expected data, and read data. The test address, expected data, and read data are displayed in hexadecimal. The graphic bit report shows a letter X at each errant bit position and a dash (-) at each good bit position.

The heading used for the graphic bit report is intended to make the bit position easy to determine. Each numeral in the heading is the one's digit of the bit position. For example, the leftmost bad bit at test address $10004 has the numeral 2 over it. Because this is the second 2 from the right, the bit position is read 12 in decimal (base 10).

Memory Management Unit Tests – Command MMU

MMU

The memory management unit tests are described in the following paragraphs.

General Description

This paragraph details the diagnostics provided to test the memory management hardware in the system. The tests are listed in Table 6–4.

TABLE 6–4. Memory Management Unit Diagnostic Tests

MONITOR COMMAND	TITLE
MMU A	RP Register
MMU B	TC Register
MMU C	Super_Prog Space
MMU D	Super_Data Space
MMU E	Write/Mapped–Read Pages
MMU F	Read Mapped ROM
MMU G	Fully Filled ATC
MMU H	User_Data Space
MMU I	User_Prog Space
MMU J	Indirect Page
MMU K	Page–Desc Used–Bit
MMU L	Page–Desc Modify–Bit
MMU M	Segment–Desc Used–Bit
MMU P	Invalid Page
MMU Q	Invalid Segment
MMU R	Write–Protect Page
MMU S	Write–Protect Segment
MMU V	Upper–Limit Violation
MMU W	Lower–Limit Violation
MMU X	Prefetch On Invalid–Page Boundary
MMU Y	Modify–Bit and Index
MMU Z 0, 1, or 2	Sixteen–Bit Bus
MMU 0	Read/Modify/Write Cycle

Hardware Configuration

The following hardware is required to perform these tests.

MVME143 - module being tested
VME chassis
Video display terminal

MMU A – RP Register

The Root Pointer (RP) register command (MMU A) is described in the following paragraphs.

Description

This command tests the RP register by doing a walking bit through it.

Command Input

```
143-Diag>MMU A
```

Response/Messages

After entering this command, the display should read as follows:

```
A     RP Register ..............................Running ---------->
```


If the RP fails to latch correctly, then the test fails and the following error message is printed out:

```
A     RP Register ..............................Running ---------->
      Expect=00000010      Read=FFFFFFFF
 .... FAILED
```

If the walk-a-bit test is successful, then the RP register test passes.

```
A     RP Register ..............................Running ----------> PASSED
```

MMU B – TC Register

MMU B

The Translation Control (TC) register test command (MMU B) is described in the following paragraphs.

Description

This command tests the TC register by attempting to clear and then set the Initial Shift (IS) bit.

Command Input

```
143-Diag>MMU B
```

Response/Messages

After entering this command, the display should read as follows:

```
B     TC Register ..............................Running ---------->
```

If the bit cannot be cleared and set, then the test fails.

```
B     TC Register ..............................Running ---------->
      Expect=00008010      Read=00000000
 .... FAILED
```

If the bit gets cleared and set, then the test passes.

```
B     TC Register ..............................Running ----------> PASSED
```

MMU C – Super_Prog Space

The super_prog space test command (MMU C) is described in the following paragraphs.

Description

This command enables the MMU and lets it do a table walk in supervisor program space. The test is implemented in the following manner:

Step 1. Put the function code in the MC68030 DFC register.

Step 2. Load the address of function code table into the root pointer register.

Step 3. Set the E bit in the translation control (TC) register.

Step 4. Let the MMU do a table walk for the next instruction, a NOP.

Step 5. Clear the E bit in the TC register.

Command Input

```
143-Diag>MMU C
```

Response/Messages

After entering this command, the display should read as follows:

```
C     Super_Prog Space .....................Running ---------->
```

If a bus error occurs, then the test fails.

```
C     Super_Prog Space .....................Running ---------->
(bus error:  CPU registers dumped to screen here)
 .... FAILED
```

If the table walk does not cause a bus error, then the test passes.

```
C     Super_Prog Space .....................Running ----------> PASSED
```

MMU D – Super_Data Space

MMU D

The super_data space test command (MMU D) is described in the following paragraphs.

Description

This command enables the MMU and lets it do a table walk twice in supervisor program space, then once in supervisor data space. The two walks in supervisor program space are necessary to fetch the instruction that accesses the supervisor data space and prefetch the next one. The test is implemented in the following manner:

Step 1. Put the function code in the MC68030 DFC register.

Step 2. Load the address of function code table into the RP register.

Step 3. Set the E bit in the TC register.

Step 4. Let the MMU do a table walk for the next instruction, which causes the access to supervisor data space via a read (TST.B). Prefetching causes access to the next location, in supervisor program space.

Step 5. Clear the E bit in the TC register.

Command Input

```
143-Diag>MMU D
```

Response/Messages

After entering this command, the display should read as follows:

```
D     Super_Data Space .....................Running ---------->
```

If a bus error occurs, then the test fails.

```
D     Super_Data Space .....................Running ---------->
(bus error:  CPU registers dumped to screen here)
 .... FAILED
```

If the table walk does not cause a bus error(s), then the test passes.

```
D     Super_Data Space .....................Running ----------> PASSED
```

MMU E - Write/Mapped-Read Pages

The write/mapped-read pages test command (MMU E) is described in the following paragraphs.

Description

This command is a test that writes data with the MMU disabled, then verifies that the data can be read with the MMU enabled. The test is implemented in the following manner:

Step 1. Fill two pages with a pattern.

Step 2. Enable the MMU.

Step 3. Check RAM where the two pages were written. Report all discrepancies.

Command Input

```
143-Diag>MMU E
```

Response/Messages

After entering this command, the display should read as follows:

```
E     Write/Mapped-Read Pages ................Running ---------->
```

If a bus error occurs or data read does not equal that expected, then the test fails. A bus error generates a dump of the MC68030 MPU registers' contents. Data corruption generates a message that indicates where the error occurred, then a map of the table walk is displayed.

```
E     Write/Mapped-Read Pages ................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the reading of the two pages does not generate a bus error, and the patterns read match the expected, then the test passes.

```
E     Write/Mapped-Read Pages ................Running ----------> PASSED
```

MMU F - Read Mapped ROM

The read mapped ROM test command (MMU F) is described in the following paragraphs.

Description

This command tests some of the upper MMU address lines by attempting to access the ROM. Both supervisor program and supervisor data function codes are used to test two separate paths through the translation table. The test is implemented in the following manner:

Step 1. Set up a pointer to the ROM that is PC relative. All PC relative accesses use supervisor program space and are software transparent.

Step 2. Set up a pointer to the ROM that uses virtual addressing. Accesses using this pointer are to supervisor data space.

Step 3. Enable the MMU.

Step 4. For each location in the ROM, read the ROM via both pointers. The data read should be identical.

NOTE

The table walking for the supervisor data space takes a much different path than that for supervisor program space.

If the data does not match, then the test fails. Display the physical address, the expected, and read data.

Step 5. Disable the MMU.

Command Input

```
143-Diag>MMU F
```

Response/Messages

After entering this command, the display should read as follows:

```
F     Read Mapped ROM .....................Running ---------->
```

If the data read via the two pointers ever differs, then the test fails.

```
F     Read Mapped ROM .....................Running ---------->
      Addr=00100000      Expect=00000001       Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the test is able to read every ROM location via both paths, then it passes.

```
F     Read Mapped ROM .....................Running ----------> PASSED
```

MMU G – Fully Filled ATC

MMU G

The fully filled Address Translation Cache (ATC) test command (MMU G) is described in the following paragraphs.

Description

This command tests the ATC by verifying that all entries in the translation cache can hold a page descriptor.

For the MMU, this is done by filling the ATC with locked descriptors, and then verifying that each descriptor is resident in the cache. This is implemented as follows:

Step 1. The lock bit is set in the first 63 page descriptors.

Step 2. The first word in each of those pages is read, creating an entry for each page in the ATC.

Step 3. The lock warning (LW) bit in the PCSR register is checked, and if it is not set, an error is flagged.

Step 4. The MMU PTEST instruction is used to verify that the page descriptors for each of the 63 pages reside in the ATC.

Command Input

```
143-Diag>MMU G
```

Response/Messages

After entering this command, the display should read as follows:

```
G     Fully Filled ATC .....................Running ---------->
```

If a word in the list does not match the corresponding word at the beginning of a page, then the test fails.

```
G     Fully Filled ATC .....................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If every word in the list matches the first word of each page, then the test passes.

```
G     Fully Filled ATC .....................Running ----------> PASSED
```

MMU H – User_Data Space

The user_data space test command (MMU H) is described in the following paragraphs.

Description

This command tests the function code signal lines connecting into the MMU by accessing user data space. This causes the MMU to read the function code and do a table walk as a part of its translation. The test is implemented in the following manner:

Step 1. Write a pattern out to an area that is mapped to user data space for diagnostic purposes.

Step 2. Enable the MMU.

Step 3. Read the area where the pattern was written to, using the function code for user data space. The test fails if the pattern does not match that written out.

Command Input

```
143-Diag>MMU H
```

Response/Messages

After entering this command, the display should read as follows:

```
H     User_Data Space .....................Running ---------->
```

If the pattern written out does not match that read, the test fails.

```
H     User_Data Space .....................Running ---------->
      Addr=00000000      Expect=00000001       Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the pattern written out matches the one read, the test passes.

```
H     User_Data Space .....................Running ----------> PASSED
```

MMU I - User_Prog Space

MMU I

The user_program space test command (MMU I) is described in the following paragraphs.

Description

This command tests the function code signal lines connecting into the MMU by accessing user program space. This causes the MMU to read the function code and do a table walk as a part of its translation. The test is implemented in the following manner:

Step 1. Write a pattern out to an area that is mapped to user program space for diagnostic purposes.

Step 2. Enable the MMU.

Step 3. Read the area where the pattern was written to, using the function code for user program space. The test fails if the pattern does not match that written out.

Command Input

```
143-Diag>MMU I
```

Response/Messages

After entering this command, the display should read as follows:

```
I     User_Prog Space .....................Running ---------->
```

If the pattern written out does not match that read, the test fails.

```
I     User_Prog Space .....................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the pattern written out matches the one read, the test passes.

```
I     User_Prog Space .....................Running ----------> PASSED
```

MMU J – Indirect Page

MMU J

The indirect page test command (MMU J) is described in the following paragraphs.

Description

This command tests the ability of the MMU to handle an indirect descriptor. The test is implemented in the following manner:

Step 1. Modify the descriptor for the first RAM page to point to the descriptor for the next RAM page.

Step 2. Write a known value into the first location of the second RAM page and the complement of that value into the first location of the first RAM page.

Step 3. Enable the MMU.

Step 4. Read the first location of the first virtual RAM page. This should address the first location in the second physical RAM page due to the indirect.

Step 5. If the value read is not the value written out to the second RAM page in Step 2, then the test fails.

Step 6. Disable the MMU.

Command Input

```
143-Diag>MMU J
```

Response/Messages

After entering this command, the display should read as follows:

```
J     Indirect Page ..........................Running ---------->
```

If the value that was supposedly read from the first virtual page in Step 4 does not match the value written in Step 2 to the second physical page, then the test fails.

```
J     Indirect Page ..........................Running ---------->
      Addr=00000000      Expect=00000001       Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the value matches, then the indirect mechanism has functioned correctly and the test passes.

```
J     Indirect Page .........................Running ----------> PASSED
```

MMU K - Page-Desc Used-Bit

MMU K

The page-descriptor used-bit test command (MMU K) is described in the following paragraphs.

Description

This command tests the ability of the MMU to set the Used-bit in a page descriptor when the page gets accessed. The test is implemented in the following manner:

Step 1. Clear the Used-bit in a page descriptor.

Step 2. Enable the MMU.

Step 3. Read from the page.

Step 4. Examine the page descriptor. If the Used-bit is not set, then the test fails.

Command Input

```
143-Diag>MMU K
```

Response/Messages

After entering this command, the display should read as follows:

```
K     Page-Desc Used-Bit .....................Running ---------->
```

If the Used-bit does not get set by the access in Step 3, then the test fails.

```
K     Page-Desc Used-Bit .....................Running ---------->
      Addr=00000000      Expect=00000001       Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the Used-bit does get set, then the test passes.

```
K     Page-Desc Used-Bit .....................Running ----------> PASSED
```

MMU L - Page-Desc Modify-Bit

The page-descriptor modify-bit test command (MMU L) is described in the following paragraphs.

Description

This command tests the ability of the MMU to set the Modify-bit in a page descriptor when the page is written. The test is implemented in the following manner:

Step 1. Clear the Modify-bit in a page descriptor.

Step 2. Enable the MMU.

Step 3. Write from the page.

Step 4. Examine the page descriptor. If the Modify-bit is not set, then the test fails.

Command Input

```
143-Diag>MMU L
```

Response/Messages

After entering this command, the display should read as follows:

```
L     Page-Desc Modify-Bit ..................Running ---------->
```

If the Modify-bit does not get set by the access in Step 3, then the test fails.

```
L     Page-Desc Modify-Bit ..................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the Modify-bit does get set, then the test passes.

```
L     Page-Desc Modify-Bit ..................Running ----------> PASSED
```

MMU M – Segment-Desc Used-Bit

The segment-descriptor used-bit test command (MMU M) is described in the following paragraphs.

Description

This command tests the ability of the MMU to set the Used-bit in a segment descriptor when the corresponding segment is accessed. The test is implemented in the following manner:

Step 1. Clear the Used-bit in a segment descriptor.

Step 2. Enable the MMU.

Step 3. Read from an address mapped to that segment.

Step 4. Check the Used-bit in the segment description. If it has not been set, then the test fails.

Command Input

```
143-Diag>MMU M
```

Response/Messages

After entering this command, the display should read as follows:

```
M     Segment-Desc Used-Bit ..................Running ---------->
```

If the Used-bit does not get set by the access in Step 3, then the test fails.

```
M     Segment-Desc Used-Bit ..................Running ---------->
      Addr=00000000      Expect=00000001       Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the Used-bit does get set, then the test passes.

```
M     Segment-Desc Used-Bit ..................Running ----------> PASSED
```

MMU P - Invalid Page

The invalid page test command (MMU P) is described in the following paragraphs.

Description

This command tests the ability of the MMU to detect an invalid page and generate bus error when access is attempted to that page. The invalid page is intentionally declared that way for test purposes. The test is implemented in the following manner:

Step 1. Modify the descriptor for a RAM page to make it invalid.

Step 2. Enable the MMU.

Step 3. Attempt to read from the page. This should generate a bus error.

Step 4. If no bus error occurred, then the test fails.

Command Input

```
143-Diag>MMU P
```

Response/Messages

After entering this command, the display should read as follows:

```
P     Invalid Page ..........................Running ---------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
P     Invalid Page ..........................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
P     Invalid Page ..........................Running ----------> PASSED
```

MMU Q – Invalid Segment

MMU Q

The invalid segment test command (MMU Q) is described in the following paragraphs.

Description

This command tests the ability of the MMU to detect an invalid segment and generate bus error when access is attempted to that segment. The invalid segment is intentionally declared that way for test purposes. The test is implemented in the following manner:

Step 1. Modify the descriptor for a RAM segment to make it invalid.

Step 2. Enable the MMU.

Step 3. Attempt to read from the page in the segment. This should generate a bus error.

Step 4. If no bus error occurred, then the test fails.

Command Input

```
143-Diag>MMU Q
```

Response/Messages

After entering this command, the display should read as follows:

```
Q     Invalid Segment .......................Running ---------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
Q     Invalid Segment .......................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
Q     Invalid Segment .......................Running ----------> PASSED
```

MMU R – Write-Protect Page

The write-protect page test command (MMU R) is described in the following paragraphs.

Description

This command tests the page write-protect mechanism in the MMU. If the MMU is functioning correctly, then attempting a write to a protected page causes a bus error. The test is implemented in the following manner:

Step 1. Set the WP bit in the descriptor for the first RAM page.

Step 2. Enable the MMU.

Step 3. Attempt to write to the protected page.

Step 4. If a bus error does not occur, then the test fails.

Command Input

```
143-Diag>MMU R
```

Response/Messages

After entering this command, the display should read as follows:

```
R     Write-Protect Page .....................Running ---------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
R     Write-Protect Page .....................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
R     Write-Protect Page .....................Running ----------> PASSED
```

MMU S - Write-Protect Segment

The write-protect segment test command (MMU S) is described in the following paragraphs.

Description

This command tests the segment write-protect mechanism in the MMU. If the MMU is functioning correctly, then attempting a write to a protected segment will cause a bus error. The test is implemented in the following manner:

Step 1. Set the WP bit in the descriptor for the first RAM page.

Step 2. Enable the MMU.

Step 3. Attempt to write to the page in the protected segment.

Step 4. If a bus error does not occur, then the test fails.

Command Input

```
143-Diag>MMU S
```

Response/Messages

After entering this command, the display should read as follows:

```
S     Write-Protect Segment ..................Running ---------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
S     Write-Protect Segment ..................Running ---------->
      Addr=00000000      Expect=00000001       Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
S     Write-Protect Segment ..................Running ----------> PASSED
```

MMU V – Upper-Limit Violation

The upper-limit violation test command (MMU V) is described in the following paragraphs.

Description

This command tests the capability of the MMU to detect when a logical address exceeds the upper limit of a segment. This condition is called an upper limit violation and should cause a bus error. The test is implemented in the following manner:

Step 1. Modify the descriptor for a segment to lower the upper limit to where it permits access to only the first page.

Step 2. Attempt access to the second page.

Step 3. This should cause a bus error. If no bus error occurs, then the test fails.

Command Input

```
143-Diag>MMU V
```

Response/Messages

After entering this command, the display should read as follows:

```
V     Upper-Limit Violation ..................Running ---------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
V     Upper-Limit Violation ..................Running ---------->
      Addr=00000000      Expect=00000001       Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
V     Upper-Limit Violation ..................Running ----------> PASSED
```

MMU W – Lower-Limit Violation

The lower-limit violation test command (MMU W) is described in the following paragraphs.

Description

This command tests the capability of the MMU to detect when a logical address exceeds the lower limit of a segment. This condition is called a lower limit violation and should cause a bus error. The test is implemented in the following manner:

Step 1. Modify the descriptor for a segment to set the lower limit to where it permits access to only the second page.

Step 2. Attempt access to the first page.

Step 3. This should cause a bus error. If no bus error occurs, then the test fails.

Command Input

```
143-Diag>MMU W
```

Response/Messages

After entering this command, the display should read as follows:

```
W     Lower-Limit Violation ..................Running ---------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
W     Lower-Limit Violation ..................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
W     Lower-Limit Violation ..................Running ----------> PASSED
```

MMU X – Prefetch on Invalid-Page Boundary

The prefetch on invalid-page boundary test command (MMU X) is described in the following paragraphs.

Description

This command tests to see if the MC68030 ignores a bus error that occurs as a result of a prefetch. The MMU signals a bus error if a prefetch operation crosses a page boundary into an invalid page. The MC68030 is to ignore such bus errors. The test is implemented in the following manner:

Step 1. Invalidate the second page mapped to user program space. This page is shared with user data space.

Step 2. Insert a trap instruction at the last location of the previous page (this page is still valid).

Step 3. Point to a special trap handler that checks for the bus error by examining some flags.

Step 4. Enable the MMU.

Step 5. Branch to the address in the first page of the trap instruction, leaving supervisor mode and entering user mode.

Step 6. The MC68030 should fetch the operating word at the end of the valid page, then attempt to prefetch the next word, which crosses the page boundary into the invalid page.

Step 7. If the MC68030 takes a bus error exception, then the test fails. Once bus error exception processing completes, control passes to the special trap handler.

Step 8. Once in the special trap handler, the stack is cleaned up (leaving the MC68030 in supervisor mode), and a test is performed to determine if the MC68030 executed a bus error exception.

Step 9. If the bus error occurred, then the test fails.

MMU X

Command Input

```
143-Diag>MMU X
```

Response/Messages

After entering this command, the display should read as follows:

```
X     Prefetch On Inv-Page ..................Running ---------->
```

If a bus error occurs, then the test fails.

```
X     Prefetch On Inv-Page ..................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the prefetching does not cause the MC68030 to take a bus error exception, then the test passes.

```
X     Prefetch On Inv-Page ..................Running ----------> PASSED
```

MMU Y - Modify-Bit and Index

The modify-bit and index test command (MMU Y) is described in the following paragraphs.

Description

This command tests the capability of the MMU to set the Modify-Bit in a page descriptor of a page which has an index field greater than 0 (not a page-0) when the page is written.

Command Input

```
143-Diag>MMU Y
```

Response/Messages

After entering this command, the display should read as follows:


```
Y     Modify-Bit & Index .....................Running ---------->
```

If the Modify-Bit does not get set by the write, then the test fails.

```
Y     Modify-Bit & Index .....................Running ---------->
      Addr=00F00000      Expect=00000010      Read=00000000
(map of table walk displayed here)
 .... FAILED
```

If the Modify-Bit does get set, the test passed.

```
Y     Modify-Bit & Index .....................Running ----------> PASSED
```

MMU Z - Sixteen-Bit Bus

MMU Z

This command is used to run the following tests with the MMU set for 16-bit bus size. The command must be followed by a numeral indicating which sub-test is to be executed.

MMU Z 0 – User-Program Space

The user–program space test command (MMU Z 0) is described in the following paragraphs.

Description

This command is used in conjunction with MMU Z to test the capability of the MMU to access user program space in 16–bit mode.

Command Input

```
143-Diag>MMU Z 0
```

Response/Messages

After entering this command, the display should read as follows:

```
0     User-Prog Space .....................Running ---------->
```

The conditions determining the success of this test are fully described in the MMU I command, as well as the error messages. If the test fails, the display appears as shown:

```
0     User-Prog Space .....................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the test passes, then the display appears as follows:

```
0     User-Prog Space .....................Running ----------> PASSED
```

MMU Z 1 – Page-Desc Modify-Bit

MMU Z 1

The page-descriptor modify-bit test command (MMU Z 1) is described in the following paragraphs.

Description

This command is used in conjunction with MMU Z to test the ability of the MMU to set the Modify-bit in a page descriptor when the page gets written to. This test operates exactly like the test described under the MMU L command; the only difference is that the MMU is set to 16-bit mode before executing the test described in that paragraph.

Command Input

```
143-Diag>MMU Z 1
```

Response/Messages

After entering this command, the display should read as follows:

```
1     Page-Desc Modify-Bit ..................Running ---------->
```

The conditions determining the success of this test are fully described in the MMU L command, as well as the error messages. If the test fails, the display appears as shown:

```
1     Page-Desc Modify-Bit ..................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the test passes, then the display appears as follows:

```
1     Page-Desc Modify-Bit ..................Running ----------> PASSED
```

MMU Z 2 – Indirect Page

The indirect page test command (MMU Z 2) is described in the following paragraphs.

Description

This command is used in conjunction with MMU Z to handle an indirect descriptor. This test operates exactly like the test described under the MMU J command; the only difference is that the MMU is set to 16-bit mode before executing the test described in that paragraph.

Command Input

```
143-Diag>MMU Z 2
```

Response/Messages

After entering this command, the display should read as follows:

```
2     Indirect Page ..........................Running ---------->
```

The conditions determining the success of this test are fully described in the MMU J command, as well as the error messages. If the test fails, the display appears as shown:

```
2     Indirect Page ..........................Running ---------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here)
 .... FAILED
```

If the test passes, then the display appears as follows:

```
2     Indirect Page ..........................Running ----------> PASSED
```

MMU 0 – Read/Modify/Write Cycle

MMU 0

The read/modify/write cycle test command (MMU 0) is described in the following paragraphs.

Description

This test performs the Test-And-Set (TAS) instruction in three modes to verify that the MMU functions correctly during read/modify/write cycles.

The message Hit Page is displayed. The MMU is turned on and a write access is performed to cache the address translation for that location. The first TAS is then done to verify that a hit page can be accessed. No bus error should result from this. The test fails if either a bus error occurs or the location (in RAM) written to does not contain $80 afterward.

The MMU is shut off and the message Missed Page displayed. The MMU is turned on, flushing the ATC. A TAS is then attempted. Because the ATC was just flushed, the access should cause a table walk and a single bus error. An error is declared if other than one bus error occurred.

If the two previous phases completed successfully, the message Unmodified Page is displayed and the final phase begun. The Modified bit for a particular page is cleared, then a read from that page is performed to cache its address translation. Next, a TAS is attempted to that location. A bus error should occur to allow the MMU time to set the Modified bit. An error is declared if the Modified bit was not set or if other than one bus error occurred.

Command Input

```
143-Diag>MMU 0
```

Response/Messages

After entering this command, the display should read as follows:

```
0     R/M/W Cycles ..........................Running ---------->
                Hit page .........
```

If the access to the hit page causes a bus error, or the location written to does not contain $80, then the test fails and the table walk is displayed.

```
0     R/M/W Cycles ..........................Running ---------->
                Hit page .........
      Addr=XXXXXXXX       Expect=80000000       Read=00000000
(map of table walk displayed here)
 .... FAILED
```

If the access to the missed page does not cause a bus error, then the test fails and the table walk is displayed.

```
0     R/M/W Cycles ..........................Running ---------->
                Hit page .........
                Missed page ......
      Addr=XXXXXXXX       Expect=80000000       Read=00000000
(map of table walk displayed here)
 .... FAILED
```


If the access to the modified page does not cause a bus error, or the Modified bit for the page does not get set, then the test fails and the table walk is displayed.

```
0     R/M/W Cycles ..........................Running ---------->
                Hit page .........
                Missed page ......
                Modified page ....
      Addr=XXXXXXXX       Expect=80000000       Read=00000000
(map of table walk displayed here)
 .... FAILED
```

If all three phases of the test complete successfully, then the test passes and the display appears as follows:

```
0     R/M/W Cycles ..........................Running ---------->
                Hit page .........
                Missed page ......
                Modified page .... PASSED
```

Table Walk Display Format

Many of the MMU tests display the supposed path through the translation table upon encountering an error. This paragraph explains the format used to display that path and the meaning of the values shown. A sample table walk display is illustrated in Figure 6-1. It is described in Table 6-5.

```
RP=00000000   TC=11111111 |
     ----V-- Fc ------      ------ Seg0 -----      ----- Seg1 ------
     22222222 33333333 --> 44444444 55555555 --> 66666666 77777777 ---+
                                                                       V
                                                              Page =  88888888
  (shown only if previous page desc is an indirect)           Page =  99999999
```

FIGURE 6-1. Sample Table Walk Display

TABLE 6-5. Sample Table Walk Display

VALUE	DESCRIPTION
00000000	Root pointer register contents, address of function code table.
11111111	Translation control register contents.
22222222	Function code table entry, status longword.
33333333	Function code table entry, address of segment 0 table.
44444444	Segment 0 table entry, status longword.
55555555	Segment 0 table entry, address of segment 1 table.
66666666	Segment 1 table entry, status longword.
77777777	Segment 1 table entry, address of page descriptor.
88888888	Page descriptor longword. Can be indirect.
99999999	Page descriptor. Shown only if previous one is indirect.

For further information as to the meaning of these values, refer to the *MC68030 Enhanced 32-bit Microprocessor User's Manual.*

Real-Time Clock Test – Command RTC

RTC

The Real-Time Clock (RTC) test command is described in the following paragraphs.

Description

This command tests the MK48T02 RTC. The battery backed-up RAM is tested, the oscillator is stopped and started, and the output is checked for roll-over.

Command Input

```
143-Diag>RTC
```

Response/Messages

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

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


If the non-destructive test of the RAM fails, the following message appears:

```
RTC   Real Time Clock Test.....................Running ---------->..... FAILED
RAM failed at $XXXXXXXX; Wrote $XX; Read $XX
```

> **CAUTION**
>
> Whether the test pass or fail, time displayed after test may not be correct.

If the oscillator does not stop on command, this message is displayed:

```
RTC   Real Time Clock Test.....................Running ---------->..... FAILED
Can't stop RTC oscillator
```

If the oscillator does not restart on command, this message is displayed:

```
RTC   Real Time Clock Test.....................Running ---------->..... FAILED
Can't start RTC oscillator
```

The test next checks time, day of week, and date in the roll-over. If any digit is wrong in roll-over, then the test fails and the appropriate one of the following error message appears as:

```
Time read was xx:xx:xx, should be 00:00:01
Day of week not 1
Date read was xx/xx/xx, should be 01/01/00
```

If a bus error occurs, the error message is:

```
Unexpected Bus Error
```

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

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

Bus Error Test – Command BERR

The Bus Error (BERR) test command is described in the following paragraphs.

Description

This command tests for local bus time-out and global bus time-out bus error conditions, including the following:

no bus error by reading from ROM

local bus time-out by reading from an undefined FC location

local bus time-out by writing to an undefined FC location

Command Input


```
143-Diag>BERR
```

Response/Messages

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

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

If a bus error occurs in the first part of the test, then the test fails and the display appears as follows.

```
BERR  Bus Error Test.........................Running ---------->..... FAILED
Got Bus Error when reading from ROM
```

If no bus error occurs in one of the other parts of the test, then the test fails and the appropriate error message appears as one of the following:

```
No Bus Error when reading from BAD address space
No Bus Error when writing to BAD address space
```

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

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

Floating Point Coprocessor (MC68882) TEST – Command FPC

FPC

The Floating Point Coprocessor (FPC) test command is described in the following paragraphs.

Description

This command tests the functions of the FPC, including all the types of FMOVE, FMOVEM, FSAVE, and FRESTORE instructions; and tests various arithmetic instructions that set and clear the bits of the FPC Status Register (FPSR).

Command Input

```
143-Diag>FPC
```

Response/Messages

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

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

If there is no FPC or if it is inoperable, then the display appears as follows:

```
FPC   Floating Pnt. Coprocessor Test.....Running ---------->..... FAILED
No FPC detected
```

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

```
FPC   Floating Pnt. Coprocessor Test.....Running ---------->..... FAILED
Test failed FPC routine at XXXXXXXX
```

Here XXXXXXXX is the hexadecimal address of the part of the test that failed. The user may look in detail at this location in the 143Bug EPROM to determine exactly what function failed.

If any part of the test is halted by an unplanned interrupt, then the display appears as follows.

```
FPC   Floating Pnt. Coprocessor Test.....Running ---------->..... FAILED
Unexpected interrupt
```

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

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

MC68030 Functionality Test – Command PIT

The MC68030 functionality test command (PIT) is described in the following paragraphs.

Description

This command tests port A, B, and C data registers for data integrity, and tests the parity error interrupt and timer interrupt capability. It then checks to be sure that the parity error interrupt can be disabled through the H34EN bit in the PGCR, and the PARIEN bit in the PBCR.

Command Input

```
143-Diag>PIT
```

Response/Messages

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

```
PIT     PIT Functionality test.............Running ---------->
```

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

```
PIT     PIT Functionality test.............Running ---------->..... FAILED
```

--(error message)

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

```
Unexpected PIT interrupt (interrupt was disabled)
Did not receive interrupt from timer
Unexpected bus error
Did not receive parity IRQ
Unexpected parity IRQ (interrupt was disabled)
Did not receive interrupt from timer
PARERR status bit in wrong state
```

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

```
PIT   PIT Functionality test.............Running ----------> PASSED
```

Z8530 Functionality Test – Command SCC

SCC

The Z8530 functionality test command (SCC) is described in the following paragraphs.

Description

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

Command Input

```
143-Diag>SCC
```

Response/Messages

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

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

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

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

           Interrupt Level = 7,          Baud = 19200

           Tx Err  Stat Chg   Rx Err   Spec Rx   Tx Tout   Rx Tout
Port 1      -        -         -         -         F         F
Port 2      -        -         -         -         F         F
Port 3      -        -         -         -         F         F
Port 4      -        -         -         -         F         F

          Z8530 Functionality Test .........FAILED

     Tx Err   = Transmit error
     Stat chg = External/status change
     Rx Err   = Receive error
     Spec Rx  = Special Receive condition
     Tx Tout  = Transmit Timeout
     Rx Tout  = Receive Timeout
```

The only other possible error message is:

```
Unexpected Bus Error
```

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

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

APPENDIX A
MVME143BUG SYSTEM MODE OPERATION

General Description

To provide compatability with the Motorola VME Delta Series systems, the MVME143Bug has a special mode of operation that allows the following features to be enabled:

a. Extended confidence tests that are run automatically on power up or reset of the MVME143.

b. A menu that allows several system start up features to be selected, such as:

- Continue start up.
- Select alternate boot device.
- Select MVME143Bug debugger.
- Initiate a service call.
- Display test errors found during start up confidence testing.
- Dump contents of system memory to tape.

c. Return to the menu upon system start up errors instead of return to the debugger.

d. Enabling of the Bug autoboot sequence.

The flow of system mode operation is shown in Figure A-1. Upon either power up or system reset, the MVME143 first executes a limited confidence test suite. This is the same test suite that the Bug normally executes on power up when not in the system mode. Upon successful completion of the limited confidence tests, a five second period is allowed to interrupt the autoboot sequence. By typing an "h" the user can cause the module to display the service menu, permitting the selection of an alternate boot device, entry to the debugger, etc., as described above. Upon selection of "continue start up" the module conducts a more extensive confidence test, including a complete parity memory test. This memory test takes a minimum of 90 seconds for a 4Mb onboard memory. Successful completion of the extended confidence test initiates the autoboot sequence, with boot taking place either from the default device (refer to Chapter 3 for information on entering/changing the default boot device) or from the selected boot device if an alternate device has been selected.

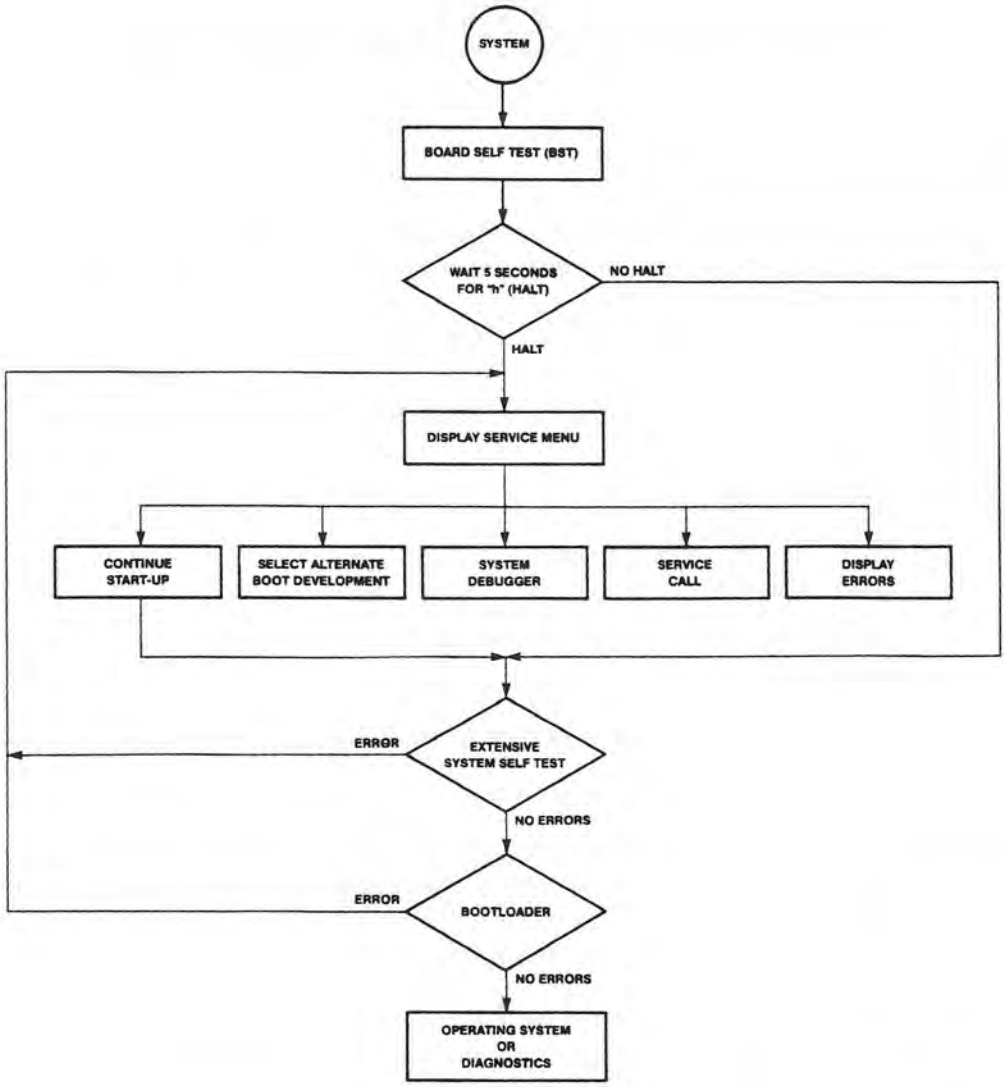


FIGURE A–1. Flow Diagram of 143Bug System Operational Mode

If the limited confidence test fails to complete correctly, it may display an error message. Explanations of these error messages can be found in Appendix B. Error message explanations for the extended confidence test are given in Chapter 6 under the heading for the failed test.

The sequence of extended confidence tests for the MVME143 is as follows:

68030 register tests
68030 instruction tests
68030 address mode tests
68030 exception processing tests
68030 cache tests
fast memory test
fast address test
VMEbus test
MMU tests
onboard tests
    bus error test
    FPC (floating point coprocessor)test
    PI/T test
    RTC test
    serial port test

Menu Details

Following are more detailed descriptions of the menu selections.

Continue System Start Up

The only action required by the user is to enter a 1 followed by a carriage return. The system then continues the start up process by initializing extended confidence testing followed by a system boot.

Select Alternate Boot Device

The user is prompted with "Enter Alternate Boot Device (Controller, Drive, File):". The selection of devices that are supported by the 143Bug are listed in Appendix E. Entry of a selected device followed by a carriage return redisplays the menu for another selection, normally "continue system start up" at this point.

Go To System Debugger

When selected, this entry places the user in 143Bug, diagnostic mode, indicated by the prompt 143-Diag>. If desired, return to the menu can be accomplished by typing "menu" when the Bug prompt appears. When in 143-Diag mode, operation is defined by sections of this manual dealing with the Bug and FAT diagnostics.

Initiate Service Call

The initiate service call function is described in the following paragraphs.

General Flow

Initiated by typing a 4 (CR) in response to the menu prompt, this function is normally used to complete a connection to a service organization which can then use the "dual console" mode of operation to assist a customer with a problem. Interaction with the service call function proceeds as follows:

First, the system asks

Is the modem: 0-UDS, 1-Hayes, 2-Manual, 3-Terminal: Your Selection?

Explanation:

UDS means that the modem is compatible with the UDS modem protocol as used in internal VME Delta Series modems. The model number of this modem is UDS 2122662.

Hayes means that the modem is compatible with a minimal subset of the Hayes modem protocol. This minimum subset is chosen to address the broadest spectrum of Hayes compatible modem products. Note that the modem itself is not tested when Hayes protocol is chosen, while the modem is tested with the UDS protocol choice.

Manual mode connects directly to the modem in an ASCII terminal mode, allowing any nonstandard protocol modem to be used.

Terminal mode is used to connect any ASCII terminal in place of a modem, via a null modem, or equivalent cable. It is useful in certain trouble-shooting applications for providing a slave terminal without the necessity of dialing through a modem.

When a selection of one of the above options is made (option 0 in this case), the system asks:

Do you want to change the baud rate from 1200 (Y/N)?

Note that any question requiring a Y or N answer defaults to the response listed furthest to the right in the line (i.e., a question with Y/N defaults to NO if only a carriage return is entered). If the user answers Y to the baud rate question, the system prompts:

Baud rate [300, 1200, 2400, 4800, 9600] 1200?

The user should enter a selected baud rate, such as 300, and type a return. A return only leaves the baud rate as previously set. The system then asks:

Is the modem already connected to customer service (Y/N)?

When a connection has been made to customer service (or any other remote device), hang up does not automatically occur; it is an operation initiated by the user. If a system reset has occurred, for instance, a hang up does not take place, and connection to CSO is still in effect. In this case, it is not necessary or desirable to attempt to reconnect on a connection that is already in effect. When an answer is entered to the question, the system responds:

Enter System ID Number:

This number is one assigned to the user system by its affiliated Customer Service Organization. The system itself does not care what is entered here, but the Customer Service computer may do a check to assure the validity of this number for login purposes. The system responds with:

Wait for an incoming Call or Dial Out (W/D)?

The user has the option of either waiting for the other computer to dial in to complete the connection, or dialing out itself. If W is selected, then skip the next two steps. If D is selected, the system asks:

Hayes Modem:

(T) = Tone Dialing (Default), (P) = Pulse Dialing
(,) = Pause and Search for a Dial Tone

UDS Modem:

(T) = Tone Dialing (Default), (P) = Pulse Dialing
(=) = Pause and Search for a Dial Tone
(,) = Wait 2 Seconds

Enter CSO phone number:

The user must enter the number, including area code if required, without any separators except for a (,) or (=) if required to allow search for a dial tone (depending on which modem protocol was selected), such as when dialing out of a location having an internal switchboard. Additionally, the number must be prefaced by one of the above dialing mode selections. The dialing selection can also be changed within the number being dialed if necessary if an internal dialing system takes a different dialing mode than the external world switched network. When connection has been made, the system reports:

Service Call in progress – Connected

The remote system can now send one of three unique commands to the local system to request specific actions via the local firmware. These commands are:

Dump Memory Command

The command to dump the private RAM used by the 147 ROM to log errors is *dppl*. The command must be received as shown in lowercase with no carriage return. Also no editing is allowed and each byte must be received within 2 seconds of the last. This command is four bytes long.

The memory dumped is the first block of memory past the exception vectors in the address range $800 to $1FFF. The memory is formatted into S-records as defined by Motorola. The S-record is sent and an ACK character $06 is expected after each record is sent. If any other byte is received, the record is resent. The record is resent 10 times before the command is aborted. The S2 record is used for the 24-bit address of the data sent. When the end of the private memory is reached, the S9 record is sent to terminate the dumping of memory.

The dumping command displays on the console that S-records are being dumped and that dumping is complete. After dumping memory is complete, the code waits for another command.

Refer to Appendix C for details on S-records.

Message Command

The command to send a message from the CSO center to the console of the calling system is <mess>, 4 bytes, followed by a string of data no more than 80 bytes in length terminated with a carriage return. The ROM code moves the string to the console followed by a carriage return and a line feed.

This command can be used to send canned messages to the operator, giving some indication of activity while various processes are taking place at CSO. For example, "Please Stand By". Many of these message commands may be sent while in the command mode.

Request for Concurrent Console Command

The request for concurrent console command is <rcc>, 3 bytes only. This prompts the operator about the request. If the operator enters "y", a single character "y" is sent to CSO followed by the console menu as displayed on the operators console. If the operator enters "n", then the single character "f" is sent to CSO and the call is terminated.

When concurrent mode is entered all input from either port, console, or remote, is taken simultaneously. All output is sent to both ports concurrently. Either the console or the remote console may terminate the concurrent mode at any time by typing a control-a. The phone line is hung up by the 147 ROM code and a message is displayed indicating the end of the concurrent mode.

The most likely command sequence at this point is a message command to indicate connection to the remote system, followed by a request for concurrent mode operation. When these are received, the user system asks:

Concurrent mode (Y/N)?

If the user wishes to enter concurrent mode, Y must be selected. The system then presents the information:

Control A to exit concurrent mode

The menu is redisplayed and concurrent mode is in effect. Any normal system operation can now be initiated at either the local or remote connected terminal, including system reboot.

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

Note that a seventh entry has been added to the menu. This Conversation Mode entry allows either party to initiate a direct conversation mode between the two terminals, the remote system terminal, and the local terminal. This seventh entry is only displayed when the system is in concurrent mode, although it actually can be selected and used at any time; only the prompt line is not displayed in normal operation. Conversation mode can be exited by typing a control A, in which case concurrent mode is terminated as well and the modem is hung up. To terminate conversation mode, but remain in concurrent mode, type the following command:

(CR),(CR)

The system then redisplays the selection menu for further operator action.

When the menu is displayed, and concurrent mode is in effect, there is another path available to terminate the concurrent connection. If the user selects menu entry 4 (Initiate Service Call) while a call is underway, the system asks:

Do you wish to disconnect the remote link (Y/N)?

If the user answers N, the system gives the option of returning to (or entering) the conversation mode:

Do you wish the conversation mode (Y/N)?

A Y response results in return to conversation mode, while an N redisplays the menu.

The system responds with the following series of messages if the disconnect option is chosen:

Wait for concurrent mode to terminate
Hanging up the phone
Concurrent mode is terminated

The last message is followed by the display of the menu WITHOUT the seventh selection available. Normal system operation is now possible.

Manual Mode Connection

As described briefly earlier, a manual modem connect mode is available to allow use of modems that do not adhere to either of the standard protocols supported, but have a defined ASCII command set. If the manual mode is selected, a few differences must be taken into account. A new mode, called "Transparent Mode" is entered when manual modem control is attempted. This means that the user terminal is in effect connected directly to the modem for control purposes. When in transparent mode, the user must take responsibility for modem control, and informing the system of when connection has taken place, etc. If "manual mode" selection is made from the "is the modem --" prompt, the following dialog takes place.

All prompts and expected responses through the "Enter System ID Number:" takes place as above. However, in manual mode, after the ID number has been entered, the system prompts:

Manually call CSO and when you are connected, exit the transparent mode
Escape character: $01=^A

The user should type a control A when connection is made, or if for any reason a connection cannot be made. Because the system has no knowledge of the status of the system when transparent mode is exited, it asks:

Did you make the connection (Y/N)?

If the user answers Y to the question, the system then continues with a normal dialog with the remote system, which would be for the remote system to send the "banner" message followed by a request for concurrent mode operation. It N is the response, the system asks:

Terminate CSO conversation (Y/N)?

A positive response to this question causes the system to reenter transparent mode and prompt:

```
Manually hang up the modem and when you are done, exit the transparent
mode
Escape character: $01 = ^A
```

The system is now in normal operation, and the menu is redisplayed.

Note that in manual mode of operation, transparent mode refers to the connection between the user terminal and the modem for manual modem control, and concurrent mode refers to the concurrent operation of a modem connected terminal and the system console.

Terminal Mode Operation

Operation with the terminal mode selected from the prompt string "Is the Modem --" is in most ways identical to other connection modes, except that after the prompt to allow change of baud rate, the system automatically enters concurrent mode. Additionally, exiting concurrent mode does not give prompts and messages referring to the hang up sequence. All other system operation is the the same as other modes of connection.

Display System Test Errors

This menu selection displays any errors accumulated by the extended confidence test suite when last run. This can be a useful field service tool.

Dump Memory To Tape

This selection creates an image of the system area of memory on a streaming tape if the prerequisite controller is attached. The option works only with QIC-02 devices as used with the Motorola MVME350 controller. Latest versions of SYSTEM V/68 operating system "crash" utilities do not utilize the results of this tape image.

This completes the description of system mode operation of the MVME143Bug.

APPENDIX B
DEBUGGING PACKAGE MESSAGES

B

The following tables list the debugging package error messages.

DEBUGGER ERROR MESSAGES	MEANING
Error Status: XXXX	Disk communication error status word when IOP command, or .DSKRD or .DSKWR TRAP #15 functions, are unsuccessful. Refer to Appendix F for details.
*** Illegal argument ***	Improper argument in known command.
Invalid command	Unknown command.
Invalid LUN	Controller and device selected during IOP or IOT command do not correspond to a valid controller and device.
*** Invalid Range ***	Range entered wrong in BF, BI, BM, BS, or DU commands.
Long Bus Error	Message displayed when using an unassigned or reserved function code or mnemonic.
part of S-record data	Printed out if non-hex character is encountered in data field in LO or VE commands.
RAM FAIL AT $XXXXXXXX	Parity is not correct at address $XXXXXXXX during a BI command.
STATUS No error since start of program	
Upload of S-Records complete.	Message from VERSAdos UPLOADS utility after successful DU command.
The following record(s) did not verify	
SNXXYYYYAAAA.....ZZ.....CS	Failure during the LO or VE commands. ZZ is the non-matching byte and CS is the non-matching checksum.

B

DEBUGGER ERROR MESSAGES	MEANING
unassembled line ----------^ *** Unknown Field ***	Message and pointer ("^") to field of suspected error when using ;DI option in MM command.
Verify passes	Successful VE command.

DIAGNOSTIC ERROR MESSAGES	MEANING
Addr=XXXXXXXX Expect=YYYYYYYY Read=ZZZZZZZZ	Error message in all MMU tests except A, B, and 0. XXXXXXXX, YYYYYYYY, and ZZZZZZZZ are hex numbers.
Battery low (data may be corrupted)	Power-up test error message.
N CACHE (HITS!/MISSES!) CACHED IN XXXX MODE, RERAN IN XXXX MODE ..... FAILED	MC68030 Cache Tests error message,where N is a number and XXXX is SUPY or USER.
CPU Addressing Modes test failed	Power-up test error message.
CPU Instruction test failed	Power-up test error message.
CPU Register test failed	Power-up test error message.
Date read was xx/xx/xx, should be 01/01/00	RTC Test error message.
Day of week not 1	RTC Test error message.
Exception Processing test failed	Power-up test error message.
Expect=XXXXXXXX Read=YYYYYYYY	Error message in MMU A or B tests. XXXXXXXX and YYYYYYYY are hex numbers.

DIAGNOSTIC ERROR MESSAGES	MEANING
FAILED	Error message in non-verbose (NV) mode.
Failed *name* addressing check	MPU Address Mode Test error message. *name* is the particular addressing mode(s) whose test(s) failed.
Failed *name* instruction check	MPU Instruction Test error message. *name* is the particular instruction(s) whose test(s) failed.
Failed *name* register check	MPU Register Test error message. *name* is the particular register(s) whose test(s) failed.
FC TEST ADDR 10987654321098765432109876543210 EXPECTED READ N NNNNNNNN ----------X-X------ NNNNNNNN NNNNNNNN	Error message display format for Memory Tests E - J, where the N's are numbers.
Got Bus Error when reading from ROM	Bus Error Test error message.
Hit page ..... Missed page ..... Modified page	Error messages in MMU 0 test.
Insufficient Memory PASSED	Memory Test I Program Test error message when the range of memory selected is less than 388 bytes and the program segment cannot be copied into RAM.
No Bus Error when (writing to/reading from) BAD address space	Bus Error Test error message.
No FPC detected	FPC Test error message when there is no FPC on the module.
MMU does not respond	MMU Test error message when the MMU fails/ does not respond.
Non-volatile RAM access error	Power-up test error message.

B

DIAGNOSTIC ERROR MESSAGES	MEANING
PASSED	Successful test message in non-verbose (NV) mode.
RAM test failed	Power-up test error message.
ROM test failed	Power-up test error message.
Test failed FPC routine at $XXXXXXXX	FPC Test error message. $XXXXXXXX is address of part of test that failed.
Test Failed Vector # XXX	MPU Exception Processing Test error message. # XXX is the exception vector offset.
Time read was *xx:xx:xx*, should be 00:00:01	RTC Test error message.
Unexpected Bus Error	MPU Address Mode, MFP Functionality, RTC, or Z8530 Functionality Test error message.
Unexpected exception taken to Vector # XXX	MPU Exception Processing Test error message. # XXX is the exception vector offset.
Unexpected interrupt	FPC Test error message.

OTHER MESSAGES	MEANING
143-Bug>	Debugger prompt.
143-Diag>	Diagnostic prompt.
At Breakpoint	Indicates program has stopped at breakpoint.
Auto Boot from controller X, device Y, STRING	Message when Autoboot is enabled by AB command. X and Y are hex numbers; STRING is an ASCII string.
Autoboot in progress... To Abort hit (BREAK)	If Autoboot is enabled, this message is displayed at Power-Up informing user that Autoboot has begun.
!!Break!!	BREAK key on console has stopped operation.
(Clock is in Battery Save Mode)	Message output when PS command halts the RTC oscillator.
COLD Start	Vectors have been initialized.
Data = $XX	XX is truncated data cut to fit data field size during BF or BV commands.
Effective address: XXXXXXXX	Exact location of data during BF, BI, BM, BS, BV, DU, and EEP commands; or where program was executed during GD, GN, GO, and GT commands.
Effective count : &XXX	Actual number of data patterns acted on during BF, BI, BS, BV, or EEP commands; or the number of bytes moved during DU command.
Escape character: $HH=AA	Exit code from transparent mode, in hex (HH) and ASCII (AA) during TM command.

B

OTHER MESSAGES	MEANING
Initial data = $XX, increment = $YY	XX is starting data and YY is truncated increment cut to fit data field size during BF or BV commands.
-last match extends over range boundary-	String found in BS command ends outside specified range.
Logical unit $XX unassigned	Message that may be output during PA or PF commands. $XX is a hex number indicating the port involved.
No Auto Boot from controller X, device Y, STRING	Message when Autoboot is disabled by NORB command. X and Y are hex numbers; STRING is an ASCII string.
No printer attached	Message that may be output during NOPA command.
-not found-	String not found in BS command.
OK to proceed (y/n)?	"Interlock" prompt before configuring port in PF command.
Press "RETURN" to continue	Message output during BS or HE command when more than 24 lines of output are available.
ROM boot disabled	Message output when NORB command disables ROMboot function.

OTHER MESSAGES	MEANING
UPLOAD "S" RECORDS Version x.y Copyrighted by MOTOROLA, INC.	Message from VERSAdos UPLOADS utility during DU command.
volume=xxxx catlg=xxxx file=FILE1 ext=MX	
WARM Start	Vectors have not been initialized.
Weekday xx/xx/xx xx:xx:xx	Day, date, and 24-hour time presentation during SET and TIME commands.

APPENDIX C
S-RECORD OUTPUT FORMAT

C

The S-record format for output modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can thus be visually monitored and the S-records can be more easily edited.

S-RECORD CONTENT

When viewed by the user, S-records are essentially character strings made of several fields which identify the record type, record length, memory address, code/data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The five fields which comprise an S-record are shown below:

type	record length	address	code/data	checksum

where the fields are composed as follows:

FIELD	PRINTABLE CHARACTERS	CONTENTS
type	2	S-record type -- S0, S1, etc.
record length	2	The count of the character pairs in the record, excluding the type and record length.
address	4, 6, or 8	The 2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
code/data	0-2n	From 0 to n bytes of executable code, memory-loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record).

checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

C

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing system.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

S-RECORD TYPES

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other record transportation control programs, as well as cross assemblers, linkers, and other file-creating or debugging programs, utilize only those S-records which serve the purpose of the program. For specific information on which S-records are supported by a particular program, the user's manual for that program must be consulted. 143Bug supports S0, S1, S2, S3, S7, S8, and S9 records.

An S-record-format module may contain S-records of the following types:

S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. Under VERSAdos, the resident linker's IDENT command can be used to designate module name, version number, revision number, and description information which will make up the header record. The address field is normally zeroes.

S1 A record containing code/data and the 2-byte address at which the code/data is to reside.

S2 A record containing code/data and the 3-byte address at which the code/data is to reside.

S3 A record containing code/data and the 4-byte address at which the code/data is to reside.

S5 A record containing the number of S1, S2, and S3 records transmitted in a particular block. This count appears in the address field. There is no code/data field.

S7 A termination record for a block of S3 records. The address field may optionally contain the 4-byte address of the instruction to which control is to be passed. There is no code/data field.

S8 A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.

S9 A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. Under VERSAdos, the resident linker's ENTRY command can be used to specify this address. If not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. S7 and S8 records are usually used only when control is to be passed to a 3- or 4-byte address. Normally, only one header record is used, although it is possible for multiple header records to occur.

CREATION OF S-RECORDS

S-record-format programs may be produced by several dump utilities, debuggers, VERSAdos' resident linkage editor, or several cross assemblers or cross linkers. On VERSAdos, the Build Load Module (MBLM) utility allows an executable load module to be built from S-records, and has a counterpart utility in BUILDS, which allows an S-record file to be created from a load module.

Several programs are available for downloading a file in S-record format from a host system to an 8-bit microprocessor-based or a 16-bit microprocessor- based system.

EXAMPLE

Shown below is a typical S-record-format module, as printed or displayed:

```
S006000048445521B
S1130000285F245F2212226A000424290008237C2A
S113001000020008000826290018538123410018l3
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The module consists of one S0 record, four S1 records, and an S9 record.

The S0 record is comprised of the following character pairs:

S0 S-record type S0, indicating that it is a header record.

06 Hexadecimal 06 (decimal 6), indicating that six character pairs (or ASCII bytes) follow.

00 Four-character 2-byte address field, zeroes in this example.
00

48
44 ASCII H, D, and R - "HDR".
52

1B The checksum.

The first S1 record is explained as follows:

S1 S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.

13 Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.

00 Four-character 2-byte address field; hexadecimal address 0000, where
00 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.L4	(A2),A1
24290008	MOVE.L	FUNCTION(A1),D2
237C	MOVE.L#	FORCEFUNC,FUNCTION(A1)

. (The balance of this code is continued in the
. code/data fields of the remaining S1 records,
. and stored in memory location 0010, etc.)

2A The checksum of the first S1 record.

The second and third S1 records each also contain $13 (19) character pairs and are ended with checksums 13 and 52, respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The S9 record is explained as follows:

S9 S-record type S9, indicating that it is a termination record.

03 Hexadecimal 03, indicating that three character pairs (3 bytes) follow.

00 The address field, zeroes.
00

FC The checksum of the S9 record.

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted. For example, the first S1 record above is sent as:

TYPE				LENGTH				ADDRESS								CODE/DATA										CHECKSUM			
S		1		1		3		0		0		0		0		2		8		5		F	...			2		A	
53		31		31		33		30		30		30		30		32		38		35		46	...			32		41	
0101	0011	0011	0001	0011	0001	0011	0011	0011	0000	0011	0000	0011	0000	0011	0000	0011	0010	0011	1000	0011	0101	0100	0110	...		0011	0010	0100	0001

APPENDIX D
INFORMATION USED BY BO AND BH COMMANDS

VOLUME ID BLOCK #0 (VID)

LABEL	OFFSET$(&)	LENGTH (BYTES)	CONTENTS
VIDOSS	$14 (20)	4	Starting block number of operating system.
VIDOSL	$18 (24)	2	Operating system length in blocks.
VIDOSA	$1E (30)	4	Starting memory location to load operating system.
VIDCAS	$90 (144)	4	Media configuration area starting block.
VIDCAL	$94 (148)	1	Media configuration area length in blocks.
VIDMOT	$F8 (248)	8	Contains the string "MOTOROLA" or "EXORMACS".

CONFIGURATION AREA BLOCK #1 (CFGA)

LABEL	OFFSET$(&)	LENGTH (BYTES)	CONTENTS
IOSPRM	$06 (6)	2	Parameters mask.
IOSATW	$08 (8)	2	Attributes word.
IOSREC	$0A (10)	2	Record (block) size in bytes.
IOSSPT	$18 (24)	1	Sectors/track.
IOSHDS	$19 (25)	1	Number of heads on drive.
IOSTRK	$1A (26)	2	Number of cylinders.
IOSILV	$1C (28)	1	Interleave factor on media.
IOSSOF	$1D (29)	1	Spiral offset.

CONFIGURATION AREA BLOCK #1 (CFGA) (cont'd)

LABEL	OFFSET$(&)	LENGTH (BYTES)	CONTENTS
IOSPSM	$1E (30)	2	Physical sector size of media in bytes.
IOSSHD	$20 (32)	2	Starting head number.
IOSPCOM	$24 (36)	2	Precompensation cylinder.
IOSSR	$27 (39)	1	Stepping rate code.
IOSRWCC	$28 (40)	2	Reduced write current cylinder number.
IOSECC	$2A (42)	2	ECC data burst length.
IOSEATM	$2C (44)	2	Extended attributes mask.
IOSEPRM	$2E (46)	2	Extended parameters mask.
IOSEATW	$30 (48)	2	Extended attributes word.
IOSGPB1	$32 (50)	1	Gap byte 1.
IOSGPB2	$33 (51)	1	Gap byte 2.
IOSGPB3	$34 (52)	1	Gap byte 3.
IOSGPB4	$35 (53)	1	Gap byte 4.
IOSSSC	$36 (54)	1	Spare sectors count.
IOSRUNIT	$37 (55)	1	Reserved area units.
IOSRSVC1	$38 (56)	2	Reserved count 1.
IOSRSVC2	$3A (58)	2	Reserved count 2.

IOSATM and IOSEATM

A "1" in a particular bit position indicates that the corresponding attribute from the attributes (or extended attributes) word should be used to update the configuration. A "0" in a bit position indicates that the current attribute should be retained.

IOSATM ATTRIBUTE MASK BIT DEFINITIONS

LABEL	BIT POSITION	DESCRIPTION
IOADDEN	0	Data density.
IOATDEN	1	Track density.
IOADSIDE	2	Single /double sided.
IOAFRMT	3	Floppy disk format.
IOARDISC	4	Disk type.
IOADDEND	5	Drive data density.
IOATDEND	6	Drive track density.
IOARIBS	7	Embedded servo drive seek.
IOADPCOM	8	Post-read/pre-write precompensation.
IOASIZE	9	Floppy disk size.
IOATKZD	13	Track zero data density.

At the present, all IOSEATM bits are undefined and should be set to 0.

IOSPRM and IOSEPRM

A "1" in a particular bit position indicates that the corresponding parameter from the configuration area (CFGA) should be used to update the device configuration. A "0" in a bit position indicates that the parameter value in the current configuration will be retained.

IOSPRM PARAMETER MASK BIT DEFINITIONS

LABEL	BIT POSITION	DESCRIPTION
IOSRECB	0	Operating system block size.
IOSSPTB	4	Sectors per track
IOSHDSB	5	Number of heads.
IOSTRKB	6	Number of cylinders.
IOSILVB	7	Interleave factor.
IOSSOFB	8	Spiral offset.
IOSPSMB	9	Physical sector size.
IOSSHDB	10	Starting head number.
IOSPCOMB	12	Precompensation cylinder number.
IOSSRB	14	Step rate code.
IOSRWCCB	15	Reduced write current cylinder number and ECC data burst length.

IOSEPRM PARAMETER MASK BIT DEFINITIONS

LABEL	BIT POSITION	DESCRIPTION
IOAGPB2	1	Gap byte 2.
IOAGPB3	2	Gap byte 3.
IOAGPB4	3	Gap byte 4.
IOASSC	4	Spare sector count.
IOARUNIT	5	Reserved area units.
IOARVC1	6	Reserved count 1.
IOARVC2	7	Reserved count 2.

IOSATW and IOSEATW

Contains various flags that specify characteristics of the media and drive.

IOSATW BIT DEFINITIONS

BIT NUMBER	DESCRIPTION
Bit 1	Track density: 0 = Single density (48 TPI) 1 = Double density (96 TPI)
Bit 2	Number of sides: 0 = Single-sided floppy 1 = Double-sided floppy
Bit 3	Floppy disk format: 0 = Motorola format (sector numbering) 1 to N on side 0 N+1 to 2N on side 1 1 = Standard IBM format 1 to N on both sides
Bit 4	Disk type: 0 = Floppy disk 1 = Hard disk
Bit 5	Drive data density: 0 = Single density (FM encoding) 1 = Double density (MFM encoding)
Bit 6	Drive track density: 0 = Single density 1 = Double density
Bit 7	Embedded servo drive: 0 = Do not seek on head switch 1 = Seek on head switch
Bit 8	Post-read/pre-write precompensation: 0 = Pre-write 1 = Post-read
Bit 9	Floppy disk size: 0 = 5-1/4 inch floppy 1 = 8-inch floppy
Bit 13	Track zero density: 0 = Single density (FM encoding) 1 = Same as remaining tracks
Unused bits	All unused bits must be set to 0.

At the present, all IOSEATW bits are undefined and should be set to 0.

PARAMETER FIELD DEFINITIONS

PARAMETER	DESCRIPTION
Record (Block)	Number of bytes per record (block). Must be an integer size multiple of the physical sector size.
Sectors/track	Number of sectors per track.
Number of heads	Number of recording surfaces for the specified device.
Number of cylinders	Number of cylinders on the media.
Interleave factor	This field specifies how the sectors are formatted on a track. Normally, consecutive sectors in a track are numbered sequentially in increments of 1 (interleave factor of 1). The interleave factor controls the physical separation of logically sequential sectors. This physical separation gives the host time to prepare to read the next logical sector without requiring the loss of an entire disk revolution.
Physical sector size	Actual number of bytes per sector on media.
Spiral offset	Used to displace the logical start of a track from the physical start of a track. The displacement is equal to the spiral offset times the head number, assuming that the first head is 0. This displacement is used to give the controller time for a head switch when crossing tracks.
Starting head number	Defines the first head number for the device.
Precompensation cylinder	Defines the cylinder on which precompensation begins.
Stepping rate code	The step rate is an encoded field used to specify the rate at which the read/write heads can be moved when seeking a track on the disk. The encoding is as follows:

PARAMETER FIELD DEFINITIONS (cont'd)

PARAMETER	DESCRIPTION			
	Step Rate Code	Winchester Hard Disks	5-1/4 inch Floppy	8-inch Floppy
	000	0 msec	12 msec	6 msec
	001	6 msec	6 msec	3 msec
	010	10 msec	12 msec	6 msec
	011	15 msec	20 msec	10 msec
	100	20 msec	30 msec	15 msec

D

Reduced write current cylinder	This field specifies the cylinder number at which the write current should be reduced when writing to the drive. This parameter is normally specified by the drive manufacturer.
ECC data burst length	This field defines the number of bits to correct for an ECC error when supported by the disk controller.
Gap byte 1	This field contains the number of words of zeros that are written before the header field in each sector during format.
Gap byte 2	This field contains the number of words of zeros that are written between the header and data fields during format and write commands.
Gap byte 3	This field contains the number of words of zeros that are written after the data fields during format commands.
Gap byte 4	This field contains the number of words of zeros that are written after the last sector of a track and before the index pulse.
Spare sectors count	This field contains the number of sectors per track allocated as spare sectors. These sectors are only used as replacements for bad sectors on the disk.

PARAMETER FIELD DEFINITIONS (cont'd)

PARAMETER	DESCRIPTION
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 = 0), or the number of cylinders (IOSRUNIT = 1) reserved for the alternate mapping area on the disk.
Reserved count 2	This field specifies the number of tracks (IOSRUNIT = 0), or the number of cylinders (IOSRUNIT = 1) reserved for use by the controller.

APPENDIX E
DISK CONTROLLER DATA

Disk Controller Modules Supported

The following VMEbus disk/tape controller modules are supported by 143Bug. The default address for each type of controller is FIRST ADDR and the controller can be addressed by FIRST CLUN during commands BH, BO, or IOP, or during TRAP #15 calls .DSKRD or .DSKWR. Note that if another one of the same type of controller is used, the second one must have its address changed by its onboard jumpers and/or switches, so that it matches SECOND ADDR and can be called up by SECOND CLUN. Additionally, if an MVME319, MVME320, and/or MVME321 are used together, the 143Bug firmware automatically assigns one of them to its default conditions (FIRST CLUN and FIRST ADDR), and also automatically assigns the other(s) to the next available higher CLUN. The priority for assigning CLUN $00 is: first MVME327A, MVME321, MVME320, MVME319, then MVME323.

E

CONTROLLER TYPE	FIRST CLUN FIRST ADDR	SECOND CLUN SECOND ADDR
MVME327A - SCSI Controller	$00 through $07 $FFFFA600	
MVME319 - SCSI/Floppy/Tape Controller	$00 $FFFF0000	$07 $FFFF0200
MVME320 - Winchester/Floppy Controller	$00 $FFFFB000	$06 $FFFFAC00
MVME321 - Winchester/Floppy Controller	$00 $FFFF0500	$01 $FFFF0600
MVME323 - ESDI Controller	$08 or greater $FFFFA000	$08 or greater $FFFFA200
MVME350 - Streaming Tape Controller	$04 $FFFF5000	$05 $FFFF5100
MVME360 - SMD Controller	$02 $FFFF0C00	$03 $FFFF0E00

Disk Controller Default Configurations

SYSTEM V/68

Controller LUN 0
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = 182Mb CDC WREN III (512b sectors)

Controller LUN 1
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = 150Mb MICROPOLIS (512b sectors)

Controller LUN 2
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = 300Mb CDC WREN IV (512b sectors)

Controller LUN 3
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = 80Mb SEAGATE (512b sectors)

Controller LUN 4
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = ARCHIVE Streaming Tape Drive

Controller LUN 5
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = ARCHIVE Streaming Tape Drive

VERSAdos

Controller LUN 0
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = 182Mb CDC WREN III (256b sectors)

Controller LUN 1
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = 150Mb MICROPOLIS (256b sectors)

Controller LUN 2
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = 300Mb CDC WREN IV (256b sectors)

Controller LUN 3
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = 80Mb SEAGATE (256b sectors)

Controller LUN 4
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = ARCHIVE Streaming Tape Drive

Controller LUN 5
- Controller Type : MVME327A
- Controller Address : $FFFFA600
- Number of Devices : 1
- Devices : DLUN 0 = ARCHIVE Streaming Tape Drive

SYSTEM V/68 or VERSAdos

Controller LUN 8 or greater

Controller Type	:	MVME319
Controller Address	:	$FFFF0000
Number of Devices	:	8
Devices	:	DLUN 0 = 40Mb Winchester hard drive (NOTE) WIN40
		DLUN 1 = 40Mb Winchester hard drive (NOTE) WIN40
		DLUN 2 = 40Mb Winchester hard drive (NOTE) WIN40
		DLUN 3 = 40Mb Winchester hard drive (NOTE) WIN40
		DLUN 4 = 8" DS/SD Motorola format floppy drive FLP8
		DLUN 5 = 8" DS/SD Motorola format floppy drive FLP8
		DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive FLP5
		DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive FLP5

E

Controller LUN 8 or greater

Controller Type	:	MVME319
Controller Address	:	$FFFF0200
Number of Devices	:	8
Devices	:	DLUN 0 = 40Mb Winchester hard drive (NOTE) WIN40
		DLUN 1 = 40Mb Winchester hard drive (NOTE) WIN40
		DLUN 2 = 40Mb Winchester hard drive (NOTE) WIN40
		DLUN 3 = 40Mb Winchester hard drive (NOTE) WIN40
		DLUN 4 = 8" DS/SD Motorola format floppy drive FLP8
		DLUN 5 = 8" DS/SD Motorola format floppy drive FLP8
		DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive FLP5
		DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive FLP5

NOTE

Devices 0 through 3 are accessed via the SCSI interface on the MVME319. An ADAPTEC ACB-4000 Winchester Disk Controller module is required tointerface between the SCSI and the disk drive. Refer to the *MVME319 Intelligent Disk/Tape Controller User's Manual* for further information.

Controller LUN 8 or greater

Controller Type	:	MVME320	
Controller Address	:	$FFFFB000	
Number of Devices	:	4	
Devices	:	DLUN 0 = 40Mb Winchester hard drive	WIN40
		DLUN 1 = 40Mb Winchester hard drive	WIN40
		DLUN 2 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
		DLUN 3 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5

Controller LUN 8 or greater

Controller Type	:	MVME320	
Controller Address	:	$FFFFAC00	
Number of Devices	:	4	
Devices	:	DLUN 0 = 40Mb Winchester hard drive	WIN40
		DLUN 1 = 40Mb Winchester hard drive	WIN40
		DLUN 2 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
		DLUN 3 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5

Controller LUN 8

Controller Type	:	MVME321	
Controller Address	:	$FFFF0500	
Number of Devices	:	8	
Devices	:	DLUN 0 = 40Mb Winchester hard drive	WIN40
		DLUN 1 = 40Mb Winchester hard drive	WIN40
		DLUN 2 = 40Mb Winchester hard drive	WIN40
		DLUN 3 = 40Mb Winchester hard drive	WIN40
		DLUN 4 = 5-1/4" DS/DD 96 TPI floppy driv	FLP5
		DLUN 5 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
		DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
		DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5

Controller LUN 8 or greater

Controller Type	:	MVME321	
Controller Address	:	$FFFF0600	
Number of Devices	:	8	
Devices	:	DLUN 0 = 40Mb Winchester hard drive	WIN40
		DLUN 1 = 40Mb Winchester hard drive	WIN40
		DLUN 2 = 40Mb Winchester hard drive	WIN40
		DLUN 3 = 40Mb Winchester hard drive	WIN40
		DLUN 4 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
		DLUN 5 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
		DLUN 6 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5
		DLUN 7 = 5-1/4" DS/DD 96 TPI floppy drive	FLP5

Controller LUN 8 or greater

Controller Type	:	MVME323	
Controller Address	:	$FFFFA000	
Number of Devices	:	1	
Devices	:	DLUN 0 = 182Mb CDC WREN III (512b sectors)	WREN
		DLUN 1 = 182Mb CDC WREN III (512b sectors)	WREN
		DLUN 2 = 182Mb CDC WREN III (512b sectors)	WREN
		DLUN 3 = 182Mb CDC WREN III (512b sectors)	WREN

Controller LUN 8 or greater

Controller Type	:	MVME323	
Controller Address	:	$FFFFA200	
Number of Devices	:	1	
Devices	:	DLUN 0 = 182Mb CDC WREN III (512b sectors)	WREN
		DLUN 1 = 182Mb CDC WREN III (512b sectors)	WREN
		DLUN 2 = 182Mb CDC WREN III (512b sectors)	WREN
		DLUN 3 = 182Mb CDC WREN III (512b sectors)	WREN

Controller LUN 8 or greater

Controller Type	:	MVME350
Controller Address	:	$FFFF5000
Number of Devices	:	1
Devices	:	DLUN 0 = QIC-02 Streaming Tape Drive

Controller LUN 8 or greater

Controller Type	:	MVME350
Controller Address	:	$FFFF5100
Number of Devices	:	1
Devices	:	DLUN 0 = QIC-02 Streaming Tape Drive

Controller LUN 8 or greater

Controller Type	:	MVME360
Controller Address	:	$FFFF0C00
Number of Devices	:	4 (NOTE)
Devices	:	DLUN 0 = 2333K Fujitsu SMD drive (512b sectors) FJI20
		DLUN 1 = null device (SMD half) SMDHALF
		DLUN 2 = 2333K Fujitsu SMD drive (512b sectors) FJI10
		DLUN 3 = null device (SMD half) SMDHALF

Controller LUN 8 or greater

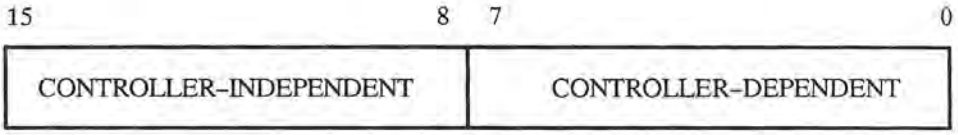
Controller Type	:	MVME360
Controller Address	:	$FFFF0E00
Number of Devices	:	4 (NOTE)
Devices	:	DLUN 0 = 2322K Fujitsu SMD drive (256b sectors) FJI10V
		DLUN 1 = null device (SMD half) SMDHALF
		DLUN 2 = 80Mb fixed CMD drive FXCMD80
		DLUN 3 = 16Mb removable CMD drive RMCMD16

NOTE

Only two physical SMD drives may be connected to an MVME360 controller, but the drive may be given two DLUNs, as is the case for Controller LUN 3 above.

APPENDIX F
DISK COMMUNICATION STATUS CODES

The status word returned by the disk TRAP #15 routines flags an error condition if it is nonzero. The most significant byte of the status word reflects controller independent errors, and they are generated by the disk trap routines. The least significant byte reflects controller dependent errors, and they are generated by the controller. The status word is shown below:



F

CONTROLLER-INDEPENDENT STATUS CODES

$00	No error detected.
$01	Invalid controller type.
$02	Invalid controller LUN.
$03	Invalid device LUN.
$04	Controller initialization failed.
$05	Command aborted via break.
$06	Invalid command packet.
$07	Invalid address for transfer.
$50	Block conversion error.

MVME327A Controller Dependent Status Codes

Fatal Error Codes

CODE (HEX)	ERROR DESCRIPTION	NOTES
$00	Good	1
	$01–0F Command Parameter Errors	
$01	Bad descriptor	2,3
$02	Bad command	2,3
$03	Unimplemented command	
$04	Bad drive	3
$05	Bad logical address	3
$06	Bad scatter-gather table	
$07	Unimplemented device	
$08	Unit not initialized	3
	$10–1F Media Areas	
$10	No ID found on track	3
$11	Seek error	3
$12	Relocated track error	3
$13	Record not found, bad ID	3
$14	Data sync fault	3
$15	ECC error	3
$16	Record not found	3
$17	Media error	3
	$20–2F Drive Errors	
$20	Drive fault	3
$21	Write protected media	3
$22	Motor not on	3
$23	Door open	3
$24	Drive not ready	3
$25	Drive busy	3

Fatal Error Codes (cont'd)

CODE (HEX)	ERROR DESCRIPTION	NOTES
	$30-3F VME DMA Errors	
$30	VMEbus error	3,4
$31	Bad address alignment	3
$32	Bus time-out	3
$33	Invalid DMA transfer count	3
	$40-4F Disk Format Errors	
$40	Not enough alternates	3
$41	Format failed	3
$42	Verify error	3
$43	Bad format parameters	3
$44	Cannot fix bad spot	3
$45	Too many defects	3
	$80-FF MVME327A Specific Errors	
$80	SCSI error, additional status available	3
$81	Indeterminate media error, no additional information	3
$82	Indeterminate hardware error	3
$83	Blank check (EOD or corrupted WORM)	3
$84	Incomplete extended message from target	3,5
$85	Invalid reselection by an unthreaded target	3,5
$86	No status returned from target	3,5
$87	Message out not transferred to target	3,5
$88	Message in not received from target	3,5

Fatal Error Codes (cont'd)

CODE (HEX)	ERROR DESCRIPTION	NOTES
$89	Incomplete data read to private buffer	3
$8A	Incomplete data write from private buffer	3
$8B	Incorrect CDB size was given	3,5
$8C	Undefined SCSI phase was requested	3,5
$8D	Time-out occurred during a select phase	3
$8E	Command terminated due to SCSI bus reset	3
$8F	Invalid message received	3,5
$90	Command not received	6
$91	Unexpected status phase	3
$92	SCSI script mismatch	3,5,9
$93	Unexpected disconnect caused command failure	3
$94	Request sense command was not successful	10
$95	No write descriptor for controller drive	7
$96	Incomplete data transfer	3
$97	Out of local resources for command processing	11
$98	Local memory resources lost	
$99	Channel reserved for another VME host	12
$9A	Device reserved for another SCSI device	12
$9B	Already enabled, expecting target response	6
$9C	Target not enabled	6

Fatal Error Codes (cont'd)

CODE (HEX)	ERROR DESCRIPTION	NOTES
$9D	Unsupported controller type	7
$9E	Unsupported peripheral device type	7
$9F	Block size mismatch	8
$A0	Invalid cylinder number in format defect list	7
$A1	Invalid head number in format defect list	7
$A2	Block size mismatch--nonfatal	8
$A3	Our SCSI ID was not changed by command	13
$A4	Our SCSI ID has changed	6,14
$A5	No target enable has been completed	6
$A6	Cannot do longword transfers	7
$A7	Cannot do DMA transfers	7
$A8	Invalid logical block size	7,8
$A9	Sectors per track mismatch	7
$AA	Number of heads mismatch	7
$AB	Number of cylinders mismatch	7
$AC	Invalid floppy parameter(s)	
$AD	Already reserved	12
$AE	Was not reserved	12
$AF	Invalid sector number	7
$CC	Self test failed	

NOTES: 1. The termination transfer count is always valid for a command that transfers data.

2. The bad byte is indicated by its offset value in status parameter

3. Additional status information may be available in the additional error code/status field of the BPP packet.

Fatal Error Codes (cont'd)

CODE (HEX)	ERROR DESCRIPTION	NOTES
	4. VMEbus error address contained in error status address field of BPP packet is currently not valid.	
	5. SCSI processing may not have finished. A SCSI bus reset command may need to be executed to put the SCSI bus in a known state. Refer to caution regarding SCSI bus resets.	
	6. Target mode only.	
	7. Designated parameter is in error.	
	8. Block size requested does not correspond to block size of device.	
	9. Target device did not behave as indicated by the script in the SCSI specific packet.	
	10. Error condition flagged by target device cannot be reported. Probably due to a hardware problem.	
	11. Command cannot be executed because local resources required exceed available local resources. Resubmit command when MVME327A is less busy.	
	12. Valid for the reserve/release commands.	
	13. Set SCSI address command was unsuccessful because at least one SCSI command was in progress.	
	14. Set SCSI address command was issued and all pending target wait commands were returned. Set SCSI address command may or may not be successful.	

F

MVME319 CONTROLLER-DEPENDENT STATUS CODES

$00	Correct execution without error.
$01	Data CRC/ECC error.
$02	Disk write protected.
$03	Drive not ready.
$04	Deleted data mark read.
$05	Invalid drive number.
$06	Invalid disk address.
$07	Restore error.
$08	Record not found.
$09	Sector ID CRC/ECC error.
$0A	VMEbus DMA error.
$0F	Controller error.
$10	Drive error.
$11	Seek error.
$19	I/O DMA error.

F

MVME320 CONTROLLER-DEPENDENT STATUS CODES

$00	Correct execution without error.
$01	Nonrecoverable error which cannot be completed (auto retries were attempted).
$02	Drive not ready.
$03	Reserved.
$04	Sector address out of range.
$05	Throughput error (floppy data overrun).
$06	Command rejected (illegal command).
$07	Busy (controller busy).
$08	Drive not available (head out of range).
$09	DMA operation cannot be completed (VMEbus error).
$0A	Command abort (reset busy).
	$0B-$FF Not used.

MVME321 CONTROLLER-DEPENDENT STATUS CODES

	*** General Error Codes ***
$00	Correct execution without error.
$17	Time-out.
$18	Bad drive.
$1A	Bad command.
$1E	Fatal error.
	*** Hard Disk Error Codes ***
$01	Write protected disk.
$02	Sector not found.
$03	Drive not ready.
$0	Drive fault or time-out on recalibrate.
$05	CRC or ECC error in data field.
$06	UPD7261 FIFO overrun/underrun.
$07	End of cylinder.
$08	Illegal drive specified.
$09	Illegal cylinder specified.
$0A	Format operation failed.
$0B	Bad disk descriptor.
$0C	Alternate track error.
$0D	Seek error.
$0E	UPD7261 busy.
$0F	Data does not verify.
$10	CRC error in ID field.
$11	Reset request (missing address mark).
$12	Correctable ECC error.
$13	Abnormal command completion.
$20	Missing data mark.
	*** Floppy Disk Error Codes ***
$01	End-of-transfer size mismatch
$02	Bad TPI combination specified.
$03	Drive motor not coming on.
$04	Disk door open.
$05	Command not completing.
$06	Bad restore operation.
$07	Illegal side reference on device.

MVME321 CONTROLLER-DEPENDENT STATUS CODES (CONT'D)

$08	Illegal track reference on device.
$09	Illegal sector reference on device.
$0A	Illegal step rate specified.
$0B	Bad density specified.
$0C	Write protected disk.
$0D	Format error.
$0E	Can't find side, track, or sector.
$0F	CRC error in ID field(s).
$10	CRC error in data field.
$11	DMA underrun.
$20	Bad disk size in descriptor.

MVME323 CONTROLLER-DEPENDENT STATUS CODES

$00	Correct execution without error.
$10	Disk not ready.
$11	Not used.
$12	Seek error.
$13	ECC code error–data field.
$14	Invalid command code.
$15	Illegal fetch and execute command.
$16	Invalid sector in command.
$17	Illegal memory type.
$18	Bus time–out.
$19	Header checksum error.
$1A	Disk write protected.
$1B	Unit not selected.
$1C	Seek error time–out.
$1D	Fault time–out.
$1E	Drive faulted.
$1F	Ready time–out.
$20	End of medium.
$21	Translation fault.

MVME323 CONTROLLER-DEPENDENT STATUS CODES (CONT'D)

$5E–5F	Not used.
$60	IOPB failed.
$61	DMA failed.
$62	Illegal VME address.
$63–69	Not used.
$6A	Unrecognized header field.
$6B	Mapped header error.
$6C–6E	Not used.
$6F	No spare sector enabled.
$70–76	Not used.
$77	Command aborted.
$78	ACFAIL detected.
$79–EF	Not used.
$F0–FE	Unforeseen error – call your Field Service representative, and tell them the IOPB and UIB information that was available at the time that the error occurred.
$FF	Command not implemented.

MVME350 CONTROLLER-DEPENDENT STATUS CODES

$00	Correct execution without error.
$01	Block in error not located.
$02	Unrecoverable data error.
$03	End of media.
$04	Write protected.
$05	Drive offline.
$06	Cartridge not in place.
$0D	No data detected.
$0E	Illegal command.
$12	Tape reset did not occur.
$17	Time-out.

MVME350 CONTROLLER-DEPENDENT STATUS CODES (CONT'D)

$18	Bad drive.
$1A	Bad command.
$1E	Fatal error.

MVME360 CONTROLLER-DEPENDENT STATUS CODES

$18	Bad drive.
$1A	Bad command.
$1E	Fatal error.
$00	Correct execution without error.
$10	Disk not ready.
$11	Not used.
$12	Seek error.
$13	ECC code error-data field.
$14	Invalid command code.
$15	Illegal fetch and execute command.
$16	Invalid sector in command.
$17	Illegal memory type.
$18	Bus time-out.
$19	Header checksum error.
$1A	Disk write protected.
$1B	Unit not selected.
$1C	Seek error time-out.
$1D	Fault time-out.
$1E	Drive faulted.
$1F	Ready time-out.
$20	End of medium.
$21	Translation fault.
$22	Invalid header pad.
$23	Uncorrectable error.
$24	Translation error, cylinder.
$25	Translation error, head.

MVME360 CONTROLLER-DEPENDENT STATUS CODES (CONT'D)

$26	Translation error, sector.
$27	Data overrun.
$28	No index pulse on format.
$29	Sector not found.
$2A	ID field error -- wrong head.
$2B	Invalid sync in data field.
$2C	No valid header found.
$2D	Seek time-out error.
$2E	Busy time-out.
$2F	Not on cylinder.
$30	RTZ time-out.
$31	Invalid sync in header.
$32-3F	Not used.
$40	Unit not initialized.
$41	Not used.
$42	Gap specification error.
$43-4A	Not used.
$4B	Seek error.
$4C-4F	Not used.
$50	Sectors per track error.
$51	Bytes per sector specification error.
$52	Interleave specification factor.
$53	Invalid head address.
$54	Invalid cylinder address.
$55-5C	Not used.
$5D	Invalid DMA transfer count.
$5E-5F	Not used.
$60	IOPB failed.
$61	DMA failed.
$62	Illegal VME address.
$63-69	Not used.
$6A	Unrecognized header field.
$6B	Mapped header error.
$6C-6E	Not used.
$6F	No spare sector enabled.

MVME360 CONTROLLER-DEPENDENT STATUS CODES (CONT'D)

$70–76	Not used.
$77	Command aborted.
$78	ACFAIL detected.
$79–EF	Not used.
$F0–FE	Unforeseen error – call your Field Service representative, and tell them the IOPB and UIB information that was available at the time that the error occurred.
$FF	Command not implemented.

F

INDEX

Numbers

143Bug Assembler Addressing Modes, 4-8

143Bug Generalized Exception Handler, 2-13

143Bug system console, 1-5

143Bug Vector Table and Wordspace, 2-10

143-Bug, 1-1

143-Diag, 1-1

A

AB command, 3-4

Abort, 1-12

ABORT switch, 1-6

Address, 2-3

address formats, 2-5

addressing modes, 4-7, 4-10

Allowed operators, 4-9

arithmetic operators, 2-3

assembler/disassembler, 4-12

assembler/disassembler function, 4-1

Assembler Output/Program Listings, 4-14

ASCII string, 2-3

AUTOBOOT, 1-7

B

backspace, 2-2

Battery Backed Up RAM (BBRAM), 1-13

Battery low (data may be corrupted), 1-6

baud rate, 1-5, A-5

BBRAM (see Battery Backed Up RAM)

BC command, 3-5

BF command, 3-7

BH Bootstrap and Halt, 1-18

BH command, 3-10

binary, 2-3

BI command, 3-11

BINDEC, 5-42

BM command, 3-12

boldface string, 2-2

Boldface brackets, 2-3

BO And BH Commands, D-1

BO Bootstrap Operating System, 1-18

BO command, 3-14

BR command, 3-17

BRA (see Branch Always)

branch always (BRA), 4-5

branch conditionally true (BT), 4-5

BRD_ID, 5-49

Break, 1-13

brief parity test command MT K, 6-42

BS command, 3-18

BT (see Branch Conditionally True)

bus error test command (BERR), 6-84

BV command, 3-21

C

CA30 A basic data caching test, 6-15

CA30 B data cache tag RAM test, 6-16

CA30 C data cache data RAM test, 6-18

CA30 D data cache valid flags test, 6-19

CA30 E data cache burst fill test, 6-20

CA30 F basic intsruction caching test, 6-21

CA30 G unlike instruction function codes test, 6-22
CA30 H disable test, 6-23
CA30 I clear test, 6-24
Calculate BCD equivalent, 5-42
Calling System Utilities From User Programs, 2-9
cancel line, 2-2
carriage return, 2-3
CHANGEV, 5-43
Character Set, 4-7
Check for break, 5-10
CHK_SUM, 5-48
CHKBRK, 5-10
Clear (Zero) Error Counters - Command ZE, 6-7
COLD and WARM reset modes, 1-12
Command Entry and Directories, 6-4
command identifier, 2-1
command line editing, 2-1
Command Parameter Errors, F-2
command parsing, 6-3
Compare two strings (pointer/count), 5-45
Comparison with MC68030 Resident Structured Assembler, 4-2
Configuration Area Block 1 (CFGA), D-1, D-2
Continue System Start Up, A-3
control codes, 2-1
Count, 2-3
CPU Addressing Modes test failed, 1-6
CPU Instruction test failed, 1-6
CPU Register test failed, 1-6
Creating a New Vector Table, 2-12
CS command, 3-23

INDEX

D

Date initialization for RTC, 5-36
DBF (see Decrement and Branch Never True)
DBRA (see Decrement and Branch Always)
DC command, 3-26
DC.W Define Constant Directive, 4-11
DC.W directive, 4-11
Delay, 5-34
Debugger Commands, 2-1, 3-1, 3-2
debugger directory, 1-1, 1-21
decrement and branch always (DBRA), 4-5
decrement and branch never true (DBF), 4-5
decimal, 2-3
default input port, 5-1
default output port, 5-1
Default 143Bug Controller and Device Parameters, 1-19
delete, 2-2
Delimiter, 2-3
Description Of MVME143Bug, 1-1
diagnostic directory, 1-1, 1-21
diagnostic guide, 6-1
Diagnostic Facilities, 1-21
Diagnostic Monitor, 6-3
Directives, 4-2
Disassembled Source Line, 4-4
Disk Communication Status Codes, F-1
Disk configure, 1-18
Disk configure function, 5-14
Disk control, 1-18
Disk control function, 5-23
Disk Controller Default Configurations, E-2

Disk Controller Modules Supported, E–1
Disk format, 1–18
Disk Format Errors, F–3
Disk format function, 5–20
Disk I/O Error Codes, 1–19
Disk I/O via 143Bug Commands, 1–17
Disk I/O via 143Bug System Calls, 1–18
Disk I/O Support, 1–17
Disk read, 1–18
Disk read function, 5–11
Disk write, 1–18
Disk write function, 5–11
Display Error Counters - Command DE, 6–7
Display Pass Count - Command DP, 6–7
Display System Test Errors, A–10
Display time from RTC, 5–37
DIVU32, 5–47
Drive Errors, F–2
DSKCFIG, 5–14
DSKCTRL, 5–23
DSKFMT, 5–20
DSKRD, 5–11
DSKWR, 5–11
DU command, 3–28
dual console mode of operation, A–4
Dump Memory to Tape, A–10

E

EEP command, 3–32
ellipsis three dots, 2–3
Entering a Source Line, 4–13
Entering And Debugging Programs, 2–9
Entering And Modifying Source Programs, 4–12
Entering Branch and Jump Addresses, 4–14
Entering Debugger Command Lines, 2–1
ENV command, 3–34
Erase Line, 5–30
ERASLN, 5–30
error messages, B–1
error reporting, 6–3
Exception Processing test failed, 1–6
Exception Vectors Used by 143Bug, 2–10
Expression, 2–3
Expression as a Parameter, 2–3
extended confidence tests, A–3
extended parity test command MT L, 6–44

F

Floating Point Coprocessor Registers, 4–6
floating point coprocessor test command (FPC), 6–85
Flow Diagram of 147Bug Normal Operational Mode, 1–2
Flow Diagram of 147Bug System Operational Mode, 1–3, A–2
Formats for Debugger Address Parameters, 2–5, 2–6
format/vector, 1–12
fully filled address translation cache (ATC) test command MMU G, 6–57
FUNCTION CODE SUPPORT, 2–15

G

GD command, 3-36

generalized exception handler, 2-11, 2-12, 2-13

Generate checksum for address range, 5-48

GN command, 3-38

Go to System Debugger, A-4

GO command, 3-40

GT command, 3-43

H

hardware handshaking, 1-5

HDS-300, 1-10

HDS-400, 1-10

HE command, 3-45

Help Command HE, 6-5

hexadecimal, 2-3

Hit Page, 6-79

HOW TO USE THIS MANUAL, 1-4

I

INCHR, 5-4

INLN, 5-6

INSTAT, 5-5

indirect page test command MMU J, 6-61

indirect page test command MMU Z 2, 6-78

initialize the system, 1-12, 1-13

Initiate service call, A-4

input/output facilities, 6-3

Installation and Startup, 1-4

interactive assembler/editor, 4-1

interrupt handling, 6-3

invalid page test command MMU P, 6-66

invalid segment test command MMU Q, 6-67

Input character routine, 5-4

Input line routine, 5-6

Input serial port status, 5-5

Invoking System Calls Through TRAP 15, 5-1

Invoking the Assembler/Disassembler, 4-12

IOC command, 3-46

IOC I/O Control, 1-18

IOP command, 3-47

IOP Physical I/O to Disk, 1-17

IOSATM ATTRIBUTE MASK BIT DEFINITIONS, D-3

IOSATW BIT DEFINITIONS, D-6

IOSEPRM PARAMETER MASK BIT DEFINITIONS, D-5

IOSPRM PARAMETER MASK BIT DEFINITIONS, D-4

IOT command, 3-51

IOT I/O Teach, 1-18

italic string, 2-2

L

label field, 4-3

level seven interrupt, 1-12

limited confidence test suite, A-1

LO command, 3-58

logical block, 1-17

Loop-Continue Mode - Prefix LC, 6-6

Loop-On-Error Mode - Prefix LE, 6-6

lower-limit violation test command MMU W, 6-71

M

MA command, 3-62

MAE command, 3-65

Main Processor Registers, 4-6

Manual mode, A-4

manual modem connect mode, A-9

MAL command, 3-67

MAR command, 3-68

march address test command MT E, 6-33

MAW command, 3-68

MC68030 Assembly Language, 4-1

MC68030 Cache Diagnostic Tests, 6-14

MC68030 MPU Diagnostic Tests, 6-9

MC68030 ONCHIP CACHE TESTS - Command CA30, 6-14

MD[S] command, 3-70

Memory Diagnostic Tests, 6-25

Memory Error Display Format, 6-47

Memory Management Unit (MMU), 2-14

Memory Management Unit Diagnostic Tests, 6-48

Memory Management Unit Support, 2-14

Memory Management Unit Tests - Command MMU, 6-48

memory map, 1-15

Memory Requirements, 1-13

Memory Tests - Command MT, 6-25

menu, A-1

MENU command, 3-72

Menu Details, A-3

Missed Page, 6-79

MM command, 3-73

MMU (see Memory Management Unit)

MMU failed test, 2-15

MMU passed test, 2-15

MMU Z - Sixteen-Bit Bus, 6-75

Mnemonics and Delimiters, 4-5

modify-bit and index test command MMU Y, 6-74

Monitor Start-Up, 6-4

MPAR (see Multiprocessor Address Register)

MPCR, 1-20

MPU A register test, 6-10

MPU B instruction test, 6-11

MPU C address mode test, 6-12

MPU D exception processing test, 6-13

MPU Tests For The MC68030 - Command MPU, 6-9

MS command, 3-76

multi-level directory, 6-3

Multiprocessor Address Register (MAR), 1-20

Multiprocessor Control Register (MPCR), 1-20

MULU32, 5-46

MVME319 Controller-Dependent Status Codes, F-7

MVME320 Controller-Dependent Status Codes, F-7

MVME321 Controller-Dependent Status Codes, F-8, F-9

MVME323 Controller-Dependent Status Codes, F-9, F-10

MVME327A Specific Errors, F-3

MVME350, 1-7

MVME350 Controller-Dependent Status Codes, F-10, F-11

MVME360, F-11, F-12

N

nibble mode test command, 6-46
NOAB command, 3-4
NOMA command, 3-62
NOMAL command, 3-67
Non-Verbose Mode - Prefix NV, 6-6
Non-volatile RAM access error, 1-6
NOPA command, 3-81
NOPF command, 3-86
NORB command, 3-88
Numeric values, 2-3

O

OBA command, 3-77
octal, 2-3
offset registers, 2-6
Operand Field, 4-4
operands types, 4-8
operation field, 4-3, 4-4
option field, 2-1
OUTCHR, 5-25
OUTLN, 5-26
Output character routine, 5-25
Output string along with (CR)(LF), 5-26
Output string to default output port, 5-26
Output string with CR and LF, 5-27
Output string with data, 5-31
Output string with data and (CR)(LF), 5-31
Output string with no CR or LF, 5-27
OUTSTR, 5-26
OVERVIEW OF DIAGNOSTIC FIRMWARE, 6-1

P

PA command, 3-81
page-descriptor modify-bit test command MMU L, 6-64
page-descriptor modify-bit test command MMU Z 1, 6-77
page-descriptor used-bit test command MMU K, 6-63
PARAMETER FIELD DEFINITIONS, D-7, D-8, D-9
Parse value, 5-43
PCRLF, 5-29
PF command, 3-82
Pointer/Count Format, 5-2
Pointer/Pointer Format, 5-2
Port Numbers, 2-8
power-up confidence test, 1-6
prefetch on invalid-page boundary test command MMU X, 6-72
PRESERVING THE DEBUGGER OPERATING ENVIRONMENT, 2-9
program counter, 1-12
program test command MT I, 6-39
Print (CR)(LF) sequence, 5-29
PS command, 3-87
pseudo-registers, 2-6
Pseudo Registers, 4-6

Q

quote mark, 2-4

R

RAM test failed, 1-6
random byte test command MT H, 6-37
range, 2-3

INDEX

RB command, 3-88
RD command, 3-89
Read line to fixed-length buffer, 5-9
Read Loop - Command RL.size, 6-8
read mapped ROM test command MMU F, 6-55
read/modify/write cycle test command MMU 0, 6-79
Read string into variable-length buffer, 5-7
Read the RTC registers, 5-38
READLN, 5-9
READSTR, 5-7
real-time clock test command (RTC), 6-82
REDIR, 5-39
REDIR_I, 5-40
REDIR_O, 5-40
Redirect I O function, 5-39
Redirect input, 5-40
Redirect output, 5-40
redisplay, 2-2
refresh test command MT G, 6-35
REMOTE command, 3-95
reset, 1-12
RESET command, 3-96
resident assembler, 4-2
resident structured assembler, 4-2
RESTARTING THE SYSTEM, 1-12
resume, 2-2
RETURN, 5-41
Return pointer to board ID packet, 5-49
Return to 147Bug, 5-41
RM command, 3-98
ROM test failed, 1-6
ROMboot , 1-8
root pointer (RP) register command MMU A, 6-50
RS command, 3-101
RTC_DSP, 5-37
RTC_DT, 5-36
RTC_RD, 5-38
RTC_TM, 5-35
rubout, 2-2

S

S-record format, C-1
Sample Table Walk Display, 6-81
SD command, 3-102
sector, 1-17
segment-descriptor used-bit test command MMU M, 6-65
Select Alternate Boot Device, A-3
Self Test Prefix/Command ST, 6-5
Send break, 5-33
set bus data width command MT D, 6-32
SET command, 3-103
set function code command MT A, 6-27
set start address command MT B, 6-28
set stop address command MT C, 6-30
shared DRAM, 1-13
SNDBRK, 5-33
Source Line Format, 4-3
SOURCE PROGRAM CODING, 4-3
Square brackets, 2-3
status register, 1-12
Stop-On-Error Mode - Prefix SE, 6-6
String Formats for I/O, 5-2
super_data space test command MMU D, 6-53

super_prog space test command MMU C, 6-52
Switch Directories - Command SD, 6-6
syntactic variables, 2-3
SYSCALL System Call Directive, 4-12
SYSTEM CALL ROUTINES, 5-2
system mode, 6-4, 6-5
SYSTEM MODE OPERATION, A-1
SYSTEM START-UP, 6-1

T

T command, 3-104
TA command, 3-107
Table Walk Display Format, 6-81
target code, 1-10
target vector table, 2-11
TAS test command MT J, 6-41
TC command, 3-108
Terminal mode, A-4, A-10
TIME command, 3-110
Time initialization for RTC, 5-35
Timer delay function, 5-34
TM command, 3-111
translation control (TC) register test command MMU B, 6-51
Transparent Mode, A-9
TRAP 15, 1-18, 2-1, 2-9, 4-12, 5-1
TRAP 15 handler, 5-1
TT command, 3-112

U

Unmodified Page, 6-79
Unsigned 32-bit x 32-bit divide, 5-47
Unsigned 32-bit x 32-bit multiply, 5-46
upper-limit violation test command MMU V, 6-70
user_data space test command MMU H, 6-59
user_program space test command MMU I, 6-60
user-program space test command MMU Z 0, 6-76
Using 143Bug Target Vector Table, 2-11
UTILITIES, 6-7

V

Valid expression examples, 2-5
VE command, 3-114
vector table area, 2-10
vertical bar, 2-2
VME DMA Errors, F-3
VOLUME ID BLOCK 0 (VID), D-1

W

wait, 2-2
walk a bit test command MT F, 6-34
WRITD, 5-31
WRITDLN, 5-31
WRITE, 5-27
WRITELN, 5-27
Write Loop - Command WL.size, 6-7
write/mapped-read pages test command MMU E, 6-54
write-protect page test command MMU R, 6-68
write-protect segment test command MMU S, 6-69
Write/Read Loop - Command WR.size, 6-8

X

XON XOFF, 1–5, 2–2

Z

Z8530 functionality test command SCC, 6–88
Zero Pass Count – Command ZP, 6–7

INDEX

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

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

ARTISAN® TECHNOLOGY GROUP