CONTENTS

START-UP

Applying power to the EVB causes a Power On Reset (POR) to occur. This POR condition causes the MCU and user I/O port circuitry to be reset, and the monitor invoked. Board COM1 port will display the monitor prompt at 9600 baud, 8,n,1:

```
AXIOM MANUFACTURING CME12D60, PRESS KEY TO START MONITOR…
```

At this state the monitor is waiting to start or launch board support utilities. User should press any alphanumeric key on the terminal keyboard to launch the debug monitor. The debug monitor prompt will display similar text:

```
Axiom MON12 – HC12 Monitor / Debugger Vxx.x
   Type “Help” for commands…
>
```

If the EVB monitor prompt is not displayed (as shown above), press the user reset switch on the board. If the EVB monitor prompt cannot be displayed, the following steps are to be performed.

a. Disconnect EVB power source.

b. Check all EVB cabling and power connections.

c. Check hardware options, configuration switches, and preparation procedures. Verify HC12 mode of operation (Expanded).

d. Check all EVB components for proper PCB placement.

e. Check PC baud rate and terminal software setup for 9600 baud, 8 bits, 1 stop, no parity.

MONITOR PROGRAM

INTRODUCTION

This chapter provides the overall description of the monitor program. This description will enable the user to understand the basic structure of the program. This program is ported from the Motorola BUFFALO monitor.

PROGRAM DESCRIPTION

The monitor program supplied for the EVB is called MON12. This program is firmware based and communicates via the HC12 Serial Communications Interface (SCI0) port. User must be aware that the monitor will initialize the hardware to operate this port and that user programs can override proper operation of the port to support the monitor. The monitor uses polling methods to determine port status so interrupt operation with the SCI port would cause an interference with the monitor.

The monitor program is contained in EPROM (external to the MCU). Having the monitor program in EPROM external to the MCU at locations $COOO-$FFFF is a great advantage because it allows the user to add instructions to customize the monitor for specific requirements.

The main module of the monitor program includes all the parts required by any of the individual command modules. Therefore the main module is a kernel of the BUFFALO monitor program, and consists of five parts which are as follows:

a. Initialization

b. Command interpreter

c. I/O routines

d. Utility subroutines

e. Command table

INITIALIZATION

This part of the main module contains all of the reset initialization code. In this section, internal RAM locations are set up, and the I/O channels for the terminal is set up. The monitor will adjust the memory map of the HC12 device for maximum use. The HC12D60 version of the monitor moves the Register space from the Reset default address of $0000 to $0800 hex and disables the internal Flash memories. This provides maximum user memory space in external ram memory for debugging.

COMMAND INTERPRETER

The next section of the main module is the command interpreter. American Standard Code for Information Interchange (ASCII) characters are read from the terminal into the input buffer until a carriage return or a slash (/) is received. The command field is then parsed out of the input buffer and placed into the command buffer. A table of commands is then searched and if a match is found, the corresponding command module is called as a subroutine. All commands return back to the command interpreter upon completion of the operation.

I/O ROUTINES

The I/O section of the main module consists of a set of supervisor routines, and three sets of driver routines. The supervisor routines are INIT, INPUT, and OUTPUT. These routines determine which driver subroutine to call to perform the specific action. Each set of driver routines consists of an initialization routine, an input routine, and an output routine. One set of drivers is for the SCI port and these routines are called ONSCI, INSCI, and OUTSCI.

All I/O communications are controlled by three RAM locations (IODEV, EXTDEV, and HOSTDEV). EXTDEV specifies the external device type (0=none, l=SCI1). HOSTDEV specifies which I/O port is used for host communications (O=SCI, 1=SCI1). IODEV instructs the supervisor routine which port/driver routine to use (0=SCI, 1=SCI1).

The INIT routines set up a serial transmission format of eight data bits, two stop bits, and no parity. For the SCI, the baud rate is set to 9600 for a 16MHz crystal (8MHz E-clock). To achieve a different baud rate the baud rate control registers can be modified using the MM command. For the SCI, this means modifying the SCOBDH and SCOBDL registers (refer to MCU data sheet, SCI baud rate selection).

The INPUT routine reads from the specified port. If a Character is received, the character is returned in accumulator A. If no character is received a zero (0) is returned in accumulator A. This routine does not wait for a character to be received before returning (this function performed by the INCHAR subroutine).

The OUTPUT routine takes the ASCII character loaded in accumulator A and writes to the specified I/O port. This routine waits until the character is transmitted before returning.

UTILITY SUBROUTINES

Several subroutines exist that are available for performing I/O tasks. A jump table has been set up in ROM directly beneath the interrupt vectors. To use these subroutines, execute a jump to subroutine (JSR) command to the appropriate entry in the jump table. By default, all I/O performed with these routines is sent to the terminal port. Redirection of the I/O port is achieved by placing the specified value (O=SCI, 1=SCI1) into RAM location IODEV.

Utility subroutines available to the user are as follows:

UPCASE If character in accumulator A is lower case alpha, convert to upper case.

WCHEK Test character in accumulator A and return with Z bit set if character is white space (space, comma, tab).

DCHEK Test character in accumulator A and return with Z bit set if character is delimiter (carriage return or white space).

INIT Initialize I/O device.

INPUT Read I/O device.

OUTPUT Write I/O device.

OUTLHLF Convert left nibble of accumulator 'A' contents to ASCII and output to terminal port.

OUTRHLF Convert right nibble of accumulator 'A' contents to ASCII and output to terminal port.

OUTA Output accumulator A ASCII character.

OUTlBYT Convert binary byte at address in index register X to two ASCII characters and output. Returns address in index register X pointing to next byte.

OUTlBSP Convert binary byte at address in index register X to two ASCII characters and output followed by a space. Returns address in index register.

OUT2BSP Convert two consecutive binary bytes starting address in index register X to four ASCII characters and output followed by a space. Returns address in index register X pointing to next byte.

OUTCRLF Output ASCII carriage return followed by a line feed.

OUTSTRG Output string of ASCII bytes pointed to by address in index register X until character is an end of transmission ($04).

OUTSTRGO Same as OUTSTRG except leading carriage return and line feed is skipped.

INCHAR Input ASCII character to accumulator A and echo back. This routine loops until character is actually received.

Utility jump subroutines for performing I/O tasks are shown below. These subroutines are in ROM and are programmed as jumps. To use the jump subroutine, execute a JSR to the applicable address shown below.

$FFAO	JMP	UPCASE	Convert character to uppercase
$FFA3	JMP	WCHEK	Test character for whitespace
$FFA6	JMP	DCHEK	Check character for delimiter
$FFA9	JMP	INIT	Initialize I/O device
$FFAC	JMP	INPUT	Read I/O device
$FFAF	JMP	OUTPUT	Write I/O device
$FFB2	JMP	OUTLHLF	Convert left nibble to ASCII and output
$FFB5	JMP	OUTRHLF	Convert right nibble to ASCII and output
$FFB8	JMP	OUTA	Output ASCII character
$FFBB	JMP	OUTlBYT	Convert binary byte to 2 ASCII characters and output
$FFBE	JMP	OUT1BSP	Convert binary byte to 2 ASCII characters and output followed by space
$FFCl	JMP	OUT2BSP	Convert 2 consecutive binary bytes to 4 ASCII characters and output followed by space.
$FFC4	JMP	OUTCRLF	Output ASCII carriage return followed by line feed
$FFC7	JMP	OUTSTRG	Output ASCII string until end of transmission (EOT / $04)
$FFCA	JMP	OUTSTRGO	Same as OUTSTRG except leading carriage return and line feed is skipped
$FFCD	JMP	INCHAR	Input ASCII character and echo back

COMMAND TABLE

The command table consists of three lines for each entry. The first byte is the number of characters in the command name. The second entry is the ASCII command name. The third entry is the starting address of the command module. As and example:

```
FCB     2           2 characters in command name
FCC     'MM'        ASCII literal command name string
FDB     #MEMORY     Jump address for command module
```

Each command in the Monitor program is a individual module. Thus, to add or delete commands, all that is required is to include a new command table.

INTERRUPT VECTORS

Interrupt vectors reside in Monitor Program memory and are accessible as pseudo vectors in a ram based look-up table. Each vector is assigned a two byte field residing in HC12 internal Ram. The Ram table may be relocated for different types of HC12 devices so that it remains in internal ram. Each pseudo vector operates similar to the firmware interrupt vectors, the pseudo vector contents are the memory address of the interrupt service routine. An interrupt service can be located anywhere in the 64K byte address map by the user except at address $0000. If an interrupt occurs for an empty vector, it will be trapped by the monitor. The monitor trap service will provide the vector offset number for the trapped interrupt. This number represents the pseudo vector location in the Ram table. To recover from a trapped interrupt, the EVB board must be Reset. The following Table lists the pseudo and associated hardware interrupt vectors:

Ram Table base address for the HC12D60 = $600.

INTERRUPT VECTOR RAM TABLE

Pseudo Vector Number Or Table Offset	HC12 INTERRUPT VECTOR	Vector ID
00	$FFC0	UNUSED VECTOR
02	$FFC2	PLL/ CGM LOCK
04	$FFC4	MSCAN TX
06	$FFC6	MSCAN RX
08	$FFC8	MSCAN ERROR
0A	$FFCA	PULSE ACC. B OVERFLOW
0C	$FFCC	MODULUS COUNTER
0E	$FFCE	PORT G/H KEY WAKE-UP
10	$FFD0	MSCAN WAKE-UP
12	$FFD2	ADC 0/1
14	$FFD4	SCI 1
16	$FFD6	SCI 0
18	$FFD8	SPI
1A	$FFDA	PULSE ACC. A INPUT
1C	$FFDC	PULSE ACC. A OVERFLOW
1E	$FFDE	TIMER OVERFLOW
20	$FFE0	TIMER 7
22	$FFE2	TIMER 6
24	$FFE4	TIMER 5
26	$FFE6	TIMER 4
28	$FFE8	TIMER 3
2A	$FFEA	TIMER 2
2C	$FFEC	TIMER 1
2E	$FFEE	TIMER 0
30	$FFF0	RTI
32	$FFF2	IRQ
34	$FFF4	XIRQ
36	$FFF6	SWI
38	$FFF8	TRAP
3A	$FFFA	COP
3C	$FFFC	CLM
3E	$FFFE	RESET, Not available to user.

To use vectors specified in the table, the user must insert the address of the interrupt service routine during code initialization into the pseudo interrupt table. For an example, for the IRQ vector, the following is performed:

```
Example:    IRQ Service routine label = IRQ_SRV
            Ram Vector Table address is defined – RIV_TBL EQU $600

Place IRQ service routine address in the table –

            MOVW  #IRQ_SRV,RIV_TBL+$32
```

After debug and the user application is to be programmed for stand alone operation without the Monitor, the actual interrupt vector table should be added and the ram vector table loading deleted.

OPERATING INSTRUCTIONS

INTRODUCTION

This chapter provides the necessary information to initialize and operate the EVB in a target system environment. Information consists of the control switch description, operating limitations, command line format, monitor commands, and operating procedures. The operating procedures consist of assembly/disassembly and downloading descriptions and examples.

RESET SWITCH

The EVB provides a user reset switch. This switch is a momentary action push-button switch that resets the EVB MCU circuits. Momentary pressing of the switch will cause the Monitor to restart and the EVB to be re-initialized.

LIMITATIONS

The MCU SCI has been set for 9600 baud using an 8MHz E clock external bus. This baud rate can be changed by software by re-programming the SC0BDL register in the ONSCI subroutine of the monitor program.

The RS-232C handshake lines are not used on the COM1 SCI port. A delay of approximately 100 milliseconds is present between successive characters sent to terminal computer during the execution of the LOAD command in the monitor program.

The monitor program will use a portion of the MCU internal RAM resources and may move internal memory or registers to provide the most complete memory map for operation. See the EVB board manual for memory mapping details and memory use.

Monitor operation typically requires the external bus to be enabled on the MCU (Expanded Mode). The MCU external bus requires I/O port use of at least two ports (16 bits) and some control lines. The MCU expanded bus ports cannot be applied by the user for I/O functions.

OPERATING PROCEDURES

The EVB is a simplified debugging/evaluating tool designed to operate in either the debugging or evaluation (emulation) mode of operation.

DEBUGGING MODE

The first mode of operation allows the user to debug user code under control of the monitor program. User code assembled or compiled on a host computer is downloaded to the EVB user RAM (see EVB manual for Ram memory location) in Motorola S-record format (LOAD command). The monitor program is then used to debug the assembled user code by the use of Breakpoint and Trace command operations to isolate any code errors. Having the monitor program in EPROM external to the MCU ($C000-$FFFF) allows the user to add instructions to customize the monitor for specific requirements.

EVALUATION MODE

The second mode of operation allows the user to evaluate (emulate) user code in a target system environment utilizing the memory of the MC68HC912 MCU. This is accomplished by relocating the code from RAM memory locations (EVB user RAM) to MCU programmable memory locations (Typically $8000 - $FFFF, see MCU user manual). The MCU internal ram and EEprom are also available to the user in this mode. The EVB then operates the user application in the single chip mode of operation or in expanded mode from the EVB boards program memory area.

MONITOR PROGRAM

The monitor program is the resident firmware for the EVB, which provides a self contained operating environment. The monitor interacts with the user through predefined commands that are entered from a terminal. The user can use any of the commands supported by the monitor.

A standard input routine controls the EVB operation while the user types a command line. Command processing will begin only after the command line has been terminated by depressing the keyboard carriage return (Return or Enter) key.

COMMAND LINE FORMAT

The command line format is as follows:

```
      >[command] [<parameters>](<ENTER)
where:
      >                 EVB monitor prompt or cursor.

      command           Command mnemonic(single letter for most commands).

      <parameters>      Expression or address, maybe optional.

      (<ENTER)          ENTER or RETURN keyboard key - depressed to enter
                         command.
NOTES:
(1) The command line format is defined using special characters which have the
following syntactical meanings:

      [ ]   Enclose optional fields, note that last command will be repeated if
            a new command is not entered.

      < >   Enclose syntactical variable

      [ ]... Enclose optional fields repeated

These characters are not entered by the user, but are for definition purposes
only.

(2) Fields are separated by any number space, comma, or tab characters.

(3) All input numbers are interpreted as hexadecimal.
```

(4) All input commands can be entered either upper or lower case lettering. All input commands are converted automatically to upper case lettering except for downloading commands sent to the host computer, or when operating in the transparent mode.

(5) A maximum of 35 characters may be entered on a command line. After the 36th character is entered, the monitor automatically terminates the command entry and the terminal CRT displays the message "Too Long".

(6) Command line errors may be corrected by backspacing (CTRL-H) or by aborting the command (CTRL-X or DELETE).

(7) After a command has been entered, pressing (RETURN) a second time will repeat the command.

MONITOR COMMANDS

The monitor BUFFALO program commands are listed alphabetically by mnemonic in Table 1. Each of the commands are described in detail following the tabular command listing.

Additional terminal keyboard functions are as follows:

(CTRL)A	Exit transparent mode or assembler
(CTRL)B	Send break command to host in transparent mode
(CTRL)H	Backspace
(CTRL)J	Line feed <1f>
(CTRL)W	Wait/freeze screen (Note 1)
(DELETE)	Abort/cancel command
(RETURN)	Enter command/repeat last command

NOTES:

(1) Execution is restarted by any terminal keyboard key.

(2) When using the control key with a specialized command such as (CTRL)A, the (CTRL) key is depressed and held, then the A key is depressed. Both keys are then released.

Command line input examples in this chapter are amplified with the following:

Underscore entries are user-entered on the terminal keyboard.

Command line input is entered when the keyboard (RETURN) key is depressed.

Typical example of this explanation is as follows:

```
>MD F000 F100
```

COMMAND TABLE

COMMAND	**DESCRIPTION**
BF <addr1> <addr2> <data>	Block fill memory with data
BR [-] [<address>]...	Breakpoint set or clear
BULK	Bulk erase EEPROM
CALL [<address>]	Execute subroutine
G [<address>]	Execute program
HELP or ?	Display monitor commands
LOAD	Download (S-records*) via terminal port
MD [<addr1>] [<addr2>]	Dump memory to terminal
MM [<address>]	Memory modify
MOVE <addr1> <addr2> [<destination>]	Move memory to new location
P	Proceed/continue from breakpoint
RM [p,y,x,a,b,c,s,]	Register modify
T [<n>]	Trace $1-$FF instructions
TM	Enter transparent mode
VERIFY	Compare memory to download data (S record)

NOTE: * Refer to Appendix A for S-record information.

MONITOR COMMAND DESCRIPTIONS

BF - BLOCK FILL MEMORY

```
     BF <address1> <address2> <data>
```

where:

`<address1>` Lower limit for fill operation.

`<address2>` Upper limit for fill operation.

`<data>` Fill pattern hexadecimal value.

The BF command allows the user to repeat a specific byte throughout a memory range. If an invalid address is specified, an invalid address message "rom-xxxx" is displayed. (xxxx = invalid address).

EXAMPLES	DESCRIPTION
>BF 0200 0230 FF	Fill each byte of memory from 0200 through 0230 with data pattern FF.
>BF 0200 0200 0	Set location 0200 to 0. (Use MM command also)

BR - BREAKPOINT SET or CLEAR

```
BR [-][<address>]...
```

where:

`[<address>]` by itself sets a breakpoint at this address.

`[-]` by itself removes (clears) all breakpoints.

`[-]` proceeding `[<address.]...` removes individual or multiple addresses from breakpoint table.

The BR command sets the address into the breakpoint address table. During program execution, a halt occurs to the program execution immediately preceding the execution of any instruction address in the breakpoint table. A maximum of four breakpoints may be set. After setting the breakpoint, the current breakpoint addresses, if any, are displayed. Whenever the G, CALL, or P commands are invoked, the monitor program inserts breakpoints into the user code at the addresses specified in the breakpoint table.

Breakpoints are accomplished by the placement of a software interrupt (SWI) at each address specified in the breakpoint address table. The SWI service routine saves and displays the internal machine state, then restores the original instructions at the breakpoint location before returning control back to the monitor program.

The SWI instruction cannot be executed in user code because the monitor program uses the SWI vector. Breakpoints can only be used in RAM memory locations.

NOTE:
Breakpoints should not be set on branch, jump, JSR, or test and branch instructions. Setting a breakpoint on these types of instructions may prevent the Proceed command from operating correctly.

COMMAND FORMATS	DESCRIPTION
BR	Display all current breakpoints.
BR <address>	Set breakpoint.
BR <addr1> <addr2>...	Set several breakpoints.
BR -	Remove all breakpoints.
BR -<addr1> <addr2>...	Remove <addr1> and add <addr2>.
BR <addr1> - <addr2>...	Add <addr1>, clear all entries, then add <addr2>.
BR <addr1> -<addr2>...	Add <addr1>, then remove addr2>.

EXAMPLES	DESCRIPTION
>BR 0203	Set breakpoint at address location 0203.
0203 0000 0000 0000 >	
>BR 0203 0205 0207 0209	Sets four breakpoints. Breakpoints at same address will result in only one breakpoint being set.
0203 0205 0207 0209 >	
>BR	Display all current breakpoints.
0203 0205 0207 0209 >	
>BR – 0209	Remove breakpoint at address location 0209.
0203 0205 0207 0000 >	
>BR 0209 -	Clear breakpoint table and add C009.
0209 0000 0000 0000 >	
>BR -	Remove all breakpoints.
0000 0000 0000 0000 >	

```
>BR E000                      Only RAM locations breakpoint.

rom-E000                      Invalid Address message.
0000 0000 0000 0000
>

>BR 0205 0207 0209 0211 0213  Maximum of four breakpoints cam be set.

Full                          Buffer full message.
0205 0207 0209 0211
>
```

BULK - BULK ERASE EEPROM

The bulk command allows the user to erase all internal MCU EEprom locations. A delay loop is built in such that the erase time is about 10ms when running at 8 MHz E clock. If EEPROM protection or lock bits are set prior to this command, then those protected locations will not be erased.

NOTE:
No erase verification message will be displayed upon completion of the bulk EEPROM erase operation. User must verify erase operation by examining one or two EEPROM locations using the MM or MD command.

```
EXAMPLE                       DESCRIPTION

>BULK                         Bulk erase entire MCU EEPROM locations

>                             Prompt indicates erase sequence completed.
```

CALL - EXECUTE SUBROUTINE

where: <address> is the starting address where user program subroutine execution begins.

The CALL command allows the user to execute a user program subroutine. Execution starts at the current program counter (PC) address location, unless a starting address is specified. Two extra bytes are placed onto the stack before the return from interrupt (RTI) is issued so that the first unmatched return from subroutine (RTS) encountered will return control back to the monitor program. Thus any user program subroutine can be called and executed via the monitor program. Program execution continues until a breakpoint encountered, or the EVB reset switch is activated (pressed).

```
EXAMPLE                              DESCRIPTION

>CALL 0200                           Execute program subroutine.

P-0200 Y-DEFE X-F4FF A-44 B-FE C-D0 S-004A
                                     Return displays status of registers at time
                                     RTS instruction is encountered (except P
                                     register contents).
```

GO - GO EXECUTE PROGRAM

where: <address> is the starting address where user program execution (free run in real time). The user may optionally specify a starting address where execution is to begin. Execution starts at the current program counter (PC) address location, unless a starting address is specified. Program execution continues until a breakpoint us encountered, or the EVB reset switch is activated (pressed).

```
EXAMPLE                         DESCRIPTION

>GO 0200                        Begin program execution at PC address location
                                0200.

P-0205 Y-0000-X-00CD A-44 B-FB C-DO S-004A
>                               Breakpoint sample encountered at 0205.
```

? / HELP - DISPLAY MENU

The HELP command enables the user available EVB command information to be displayed for quick reference purposes.

EXAMPLE

```
      >HELP
OR
      >?
```

Output String:

```
BF    <addr1> <addr2> [>data>]      Block fill
BR    [-] [<addr>]                  Set / Clear breakpoint
BULK                                Erase the EEPROM
CALL  [<addr>]                      Call user subroutine.
G0    [<addr>]                      Execute user code.
LOAD                                Load S record file
VERIFY                              Verify memory with S record
MD    [<addr1>] [<addr2>]           Memory display
MM    [<addr>]                      Memory modify
      /                             Open same address
      CTRL-H or A                   Open previous address
      CTRL-J                        Open next address
      SPACE                         Open next address
      RETURN                        Quit
      <addr>0                       Compute offset to <addr>
MOVE  <s1> <s2> [<d>]               Block move memory
P                                   Proceed/continue execution.
RM [P,Y,X,A,B,C, OR S]              Register modify.
T[<n>]                              Trace n instructions.
TM                                  Transparent mode, COM1 to COM2
      CTRL-A                        Exit
      CTRL B                        Send break
      CTRL-H                        Backspace
      CTRL-W                        Wait for any key.
      CTRL-X or DELETE              Abort/cancel command.
      RETURN                        Repeat last command.
```

LOAD - RECEIVE S-RECORD FILE

The LOAD command receives (downloads) object data in S-record format (see Appendix A) from the host terminal computer to the EVB. As the EVB monitor processes only valid S-record data, it is possible for the monitor to hang up during a load operation. If an S-record starting address points to and invalid memory location, the invalid address message "error addr xxxx" is displayed on the Terminal {xxxx = invalid address). A valid load will be followed by a 'Done' message.

```
EXAMPLES                DESCRIPTION

>LOAD                   LOAD command entered to download data from host. User
                        should now send the S record file with host terminal
                        software.
done
>                       Done message with command prompt returns after S record
                        received

>LOAD                   LOAD command entered.

error addr E000         Invalid address message.
>                       S-records must be downloaded into RAM.
```

MD - MEMORY DISPLAY

```
where: <address1>       Memory starting address (optional).

       [<address2>]     Memory ending address (optional).
```

The MD command allows the user to display a block of user memory beginning at address1 and continuing to address2. If address2 is not entered, 9 lines of 16 bytes are displayed beginning at address1. If address1 is greater than address2, the display will default to the first address. If no addresses are specified, 9 lines of 16 bytes are displayed near the last memory location accessed.

EXAMPLES

```
>MD
F7D0 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA   ................
F7E0 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA   ................
F7F0 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA   ................
F800 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA   ................
F810 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA   ................
F820 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA   ................
F830 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA   ................
F840 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA   ................
F850 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA   ................
>
```

```
>MD 0200 0220

0200  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0210  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0220  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................

>MD 0230 0220

0230  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
>
```

MM - MEMORY MODIFY

where: <address> is the memory location at which to start display/modify.

The MM command allows the user to examine/modify contents in user memory at specified locations in an interactive manner. Once entered, the MM command has several sub modes of operation that allow modification and verification of data. The following subcommands are recognized.

CTRL J or (Space Bar	Examine/modify next location.
CTRL H or A	Examine/modify previous location.
/	Examine/modify same location.
RETURN	Terminate MM operation.
O	Compute branch instruction relative offset.

If an invalid address is specified, the invalid address message "rom" is displayed on the terminal CRT>

EXAMPLES	DESCRIPTION
>MM 0700	Display memory location 0700.
0700 44 66/	Change data at 0700 and re-examine location.
0700 66 55^A	Change data at 0700 and backup one location.
06FF FF AA(<Enter)	Change data at 06FF and terminate MM operation.
>MM 013C	Display memory location.
013C F7 C18E0 51 013C F7	Compute offset, result = $51.
>MM 0200	Examine location $0200.
0200 55 80 C2 00 CE C4	Examine next location(s) using (Space Bar).

MOVE - MOVE MEMORY CONTENTS

```
MOVE <address1> <address2> [<destination>]

where:
      <address1>        Memory starting address.

      <address2>        Memory ending address.

      [<dest>]          Destination starting address (optional).
```

The MOVE command allows the user to copy/move memory to new memory location. If the destination is not specified, the block of data residing from address1 to address2 will be moved up one byte. Using the MOVE command on EEPROM locations will program EPROM cells.

The MOVE command is useful when programming EEPROM. As an example, a program is loaded in user RAM using the LOAD command, debugged using the monitor, and then programmed into EEPROM with the MOVE command (note addressing must be relative).

No messages will be displayed on the terminal CRT upon completion of the copy / move operation, only the prompt is displayed.

```
EXAMPLE                 DESCRIPTION

>MOVE E000 E7FF 0200    Move data from locations $E000-$E7FF to
                        locations $0200-$09FF.
>
```

P - PROCEED FROM BREAKPOINT

This command is used to proceed or continue program execution without having to remove assigned breakpoints. This command is used to bypass assigned breakpoints in a program executed by the G command.

NOTE

Breakpoints have been inserted at locations $0205 and $0207 for example.

```
EXAMPLE                                     DESCRIPTION

>G 0200S                                    Start execution at 0200.

P-0205 Y-7982 X-FF00 A-44 B-70 C-DO S-004A    Breakpoint encountered at
>                                               0205.

>P                            Continue execution.

P-0207 Y-7982 X-FF00 A-44 B-70 C-C0 S-004A    Breakpoint encountered at
                                                0207
>
```

RD - REGISTER DISPLAY

The RD command is used to display the current user MCU core registers (PC, x, y, a, b, c, sp).

EXAMPLE	DESCRIPTION
>RD	Display registers
P-0200 Y-798 X-FF00 A-44 B-70 C-C0 S-0054	
>	

RM - REGISTER MODIFY

RM [p, y, x, a, b, c, s]

The RM command is used to modify the MCU program counter (P), Y index (Y), X index (X), A accumulator (A), B accumulator (B), C accumulator (C), and stack pointer (S) register contents.

EXAMPLE	DESCRIPTION
>RM	Display P register contents.
P-0200 Y-798 X-FF00 A-44 B-70 C-C0 S-0054	
P-0207 0220	Modify P register contents.
>	
>RM X	Display X register contents.
P-C007 Y-7982 X-FF00 A-44 B-70 C-C0 S-0054	
X-FF00 0220	Modify X register contents.
>	
>RM	Display P register contents.
P-0220 Y0DEFE X-0220 A-DF B-DE C-D0 S-0054	
P-0220 (SPACE BAR)	Display remaining registers.
Y-DEFE (SPACE BAR)	
X-0220 (SPACE BAR)	
A-DF (SPACE BAR)	
B-DE (SPACE BAR)	
C-DO (SPACE BAR)	
S-0054 (SPACE BAR)	(SPACE BAR) entered following stack pointer display will terminate RM command.

T - TRACE

```
T [<n>]
```

where: <n> is the number (in hexadecimal, $1-FF max.) of instructions to execute.

The T command allows the user to monitor program execution on an instruction-by-instruction basis. The user may optionally execute several instructions at a time by entering a count value (up to $FF). Execution starts at the current program counter (PC). The PC display with the event message is of the next instruction to be executed.

The trace command operates by installing a SWI instruction at the next instruction opcode. Current version does not compute relative addressing or test and branch offsets to anticipate program counter modification. This means jump and branch instructions will not be traced. A test and branch instruction will only trace if condition is false. Trace of Jump to Subroutine (JSR or BSR) instructions will execute the subroutine before the next trace step break. Interrupts will not be traced unless a breakpoint is installed in the interrupt service routine.

```
EXAMPLES                                    DESCRIPTION

>T                                          SINGLE TRACE

Op- 86
P-0202 Y-DEFE X-FFFF A-44 B-00 C-00 S-004B
>

>T 2                                        MULTIPLE TRACE (2)

Op-B7
P-0205 Y-DEFE X-FFFF A-44 B-00 C-00 S-004B

Op-01
P-0206 Y-DEFE X-FFFF A-44 B-00 C-00 S-004B
>
```

TM - TRANSPARENT MODE

The TM command connects the EVB COM1 and COM2 ports, which allows direct communication between the COM1 terminal port to a device on the COM2 port. All I/O between the ports are ignored by the EVB until the exit character is entered from the terminal.

Note that EVB boards without a COM2 port do not support this command.

The TM subcommands are as follows:

```
     (CTRL)A   Exit from transparent mode
     (CTRL)B   Send break to COM2 port.

 EXAMPLE                          DESCRIPTION
```

```
>TM                                 Enter transparent mode.

.                               Host computer input
.
.
(CTRL)A                             Task completed. Enter exit command.
>                                   Exit transparent mode.
```

Refer to the downloading procedures at the end of this chapter for additional information pertaining to the use of the TM command.

VERIFY - VERIFY MEMORY AGAINST S RECORD

The VERIFY command is similar to the LOAD command except that the VERIFY command instructs the EVB to compare the downloaded S-record data to the data stored in memory.

```
EXAMPLES                            DESCRIPTION

>VERIFY                            Enter verify command.

                                    Send S Record file from host now.

done                                Verification completed.
>

>VERIFY                            Enter verify command.

                                    Send S record file from host now.
Mismatch encountered.

error addr E000                     Error message displaying first byte
                                    address.
```

Refer to the downloading procedures at the end of this chapter for additional information pertaining to the use of the LOAD command.

DOWNLOADING PROCEDURES

This portion of text describes the downloading procedures. The downloading operation enables the user to transfer information from a host computer to the EVB (or target system memory) using the LOAD command. The VERIFY command is used to compare the S-record data to memory data.

Specific downloading procedures are described enabling the user to IBM Personal Computer (PC) host computer system. Downloading operations are accomplished utilizing the LOAD command. The LOAD command moves data information in S-record format (see Appendix A) from an external host computer to the EVB user RAM.

The following pages provide examples and descriptions of how to perform downloading operations in conjunction with an IBM PC host computer.

Prior to performing any IBM PC operation, ensure that both IBM PC and EVB baud rates are identical.

NOTE

IBM PC to EVB interconnection is accomplished by a single RS-232C cable assembly (usually provided with EVB). This cable is connected to the EVB terminal I/O port connector COM1 for downloading operations.

To perform the IBM PC to EVB downloading procedure, perform/observe the following:

EXAMPLE	DESCRIPTION
>LOAD	EVB download command entered.

Using Axiom AXIDE Terminal program select Upload Arrow on tool bar, file to send, and send file now.

Using HyperTerminal program or similar, select text file transfer, file to send, and send now.

Done. >	EVB prompt after file received.

Note error messages may be presented such as “Too Long”. Make sure S record is located in RAM memory area on EVB and that the ENTER or RETURN key is pressed after the LOAD command is typed in to execute the LOAD command first. If prompt does not return after download, S record maybe invalid or maybe located in Monitor Ram use area. Review S record location and format.

APPENDIX A: S-RECORD INFORMATION

INTRODUCTION

The Motorola S-record format was devised for the purpose of encoding programs (object code) or data files in a printable format for transportation between computer systems. This transportation process can therefore be monitored and the S-records can be easily edited.

S-RECORD CONTENT

When observed, 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.

Five fields which compromise 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	Character pair count in the record, excluding the type and record length.
Address	4,6, or 8	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 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-
Checksum	2	Least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing systems.

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 manual for that program must be consulted.

NOTE

The EVB monitor supports only the S1 and S9 records. All data before the first S1 record is ignored. Thereafter, all records must be S1 type until the S9 record terminates data transfer.

An S-record format may contain the following record types:

S0 Header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. The address field is normally zeroes.

S1 Code/data record and the 2-byte address at which the code/data is to reside.

S2-S3 Code/data record is 3 or 4 byte address. Not supported by monitor.

S7-8 Termination record for a block of S2 or S3 records.

S9 Termination record for a block of S1 records. Address fields may optionally contain the 2-byte address of the instruction to which control is to be passed. If not specified, the first entry point specification encountered in the input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. Normally, only one header record is used, although it is possible for multiple header records to occur.

S-RECORD CREATION

S-record format programs maybe produce by several dump utilities, debuggers, or several cross assemblers or cross linkers. Several programs are available for downloading a file in S-record format from a host system to an 8-bit or 16-bit microprocessor-based system.

S-RECORD EXAMPLE

Shown below is a typical S-record format, as printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
```

```
          S9030000FC
```

The above format consists of an S0 header record, four S1 code/data records, and an S9 termination record.

The S0 header record is comprised of the following character pairs:

```
      S0    S-record type S0, indicating a header record.

      06    Hexadecimal 06 (decimal 06), indicating six character pairs
            (or ASCII bytes) follow.

      0000  Four-character 2-byte address field, zeroes.

      48
      44    ASCII H, D, and R - "HDR".
      52

      1B    Checksum of SO record.
```

The first S1 code/data record is explained as follows:

```
      S1    S-record type S1, indicating a code/data record to be
            loaded/verified at a 2-byte address.

      13    Hexadecimal 13 (decimal 19), indicating 19 character pairs,
            representing 19 bytes of binary data, follow.

      0000  Four-character 2-byte address field; hexadecimal address 0000,
            indicates location where the following data is to be loaded.
```

The next 16 character pairs 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

28 5F                         BHCC  $0161
24 5F                         BCC   $0163
22 12                         BHI   $0118
22 6A                         BHI   $0172
00 04 24                      BRSET 0,$04,$012F
29 00                         BHCS  $010D
08 23 7C                      BRSET 4,$23,$018C

.    (Balance of this code is continued in the code/data fields
.    of the remaining S1 records, and stored in memory location
.    0010, etc..)

2A   Checksum of the first S1 record.
```

The second and third S1 code/data records each also contain $13 (19) character pairs and are ended with checksums 13 and 51, respectively. The fourth S1 code/data record contains 07 character paris and has a checksum of 92.

The S9 termination record is explained as follows:

```
     S9     S-record type S9, indicating a termination record.

     03     Hexadecimal 03, indicating three character pairs (3 bytes)
                  follow.

     00     Four-character 2-byte address field, zeroes.
     00

     FC     Checksum of 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 shown below.

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

TEXT MESSAGES

```
What?
Too Long
Full
Op-
rom-
Command?
Bad Argument
No host port available
done
checksum error
error addr
```