**M68HC11EVBU/D**
**REV 3**

April 1997

# M68HC11EVBU

# UNIVERSAL EVALUATION BOARD

# USER’S MANUAL

**Information contained in this document applies to REVision (B) M68HC11EVBU Universal Evaluation Boards.**

# CHAPTER 3

# MONITOR PROGRAM

## 3.1 INTRODUCTION

This chapter is a description of the monitor program. This description enables you to understand the basic structure of the program.

## 3.2 PROGRAM DESCRIPTION

The BUFFALO (bit user fast friendly aid to logical operations) monitor program is supplied with the EVBU. The EVBU monitor program is stored in the MCU internal ROM and communicates via the MCU serial communications interface (SCI). Refer to the buf32.asm file on the EVBU diskettes for additional information pertaining to the monitor (BUFFALO) program.

The BUFFALO monitor program consists of five parts (or sections) as follows:

1. Initialization
2. Command interpreter
3. I/O routines
4. Utility subroutines
5. Command table

### 3.2.1 Initialization

This part of BUFFALO contains all of the reset initialization code. In this section, internal RAM locations are set up, and the I/O channel for the terminal is set up. To set up the terminal I/O port, BUFFALO must determine if the terminal is connected to the SCI (as in the EVBU) or to an external ACIA or DUART. This is accomplished by sending a sign-on message to all ports and then waiting for you to type carriage return **<CR>** on whichever device is the terminal port. When BUFFALO recognizes a carriage return from a port, that port is then used for all subsequent terminal I/O operations. The EVBU terminal device is normally connected to the SCI.

### 3.2.2 Command Interpreter

The next section of BUFFALO 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 control back to the command interpreter upon completion of the operation.

### 3.2.3 I/O Routines

The I/O section of BUFFALO 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. The second set of drivers is for a DUART and these routines are called ONUART, INUART, and OUTUART. The third set of drivers is for an ACIA and these routines are called ONACIA, INACIA, and OUTACIA.

All I/O communications are controlled by three RAM locations (IODEV, EXTDEV, and HOSTDEV). EXTDEV specifies the external device type (0=none, 1=ACIA, 2=DUART). HOSTDEV specifies which I/O port is used for host communications (0=SCI, 1=ACIA, 3=DUARTB). IODEV instructs the supervisor routine which port/driver routine to use (0=SCI, 1=ACIA, 2=DUARTA, 3=DUARTB).

The INIT routines set up a serial transmission format of eight data bits, one stop bit, and no parity. For the SCI, the baud rate is set to 9600 for an 8 MHz crystal (2 MHz E-clock). A different baud rate can be achieved by modifying address location $102B (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 to accumulator A. If no character is received, a logic zero (0) is returned to accumulator A. This routine does not wait for a character to be received before returning (that function is performed by the INCHAR utility subroutine).

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

### 3.2.4 Utility Subroutines

Several subroutines exist that are available for performing I/O tasks. A jump table has been set up in ROM directly before 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 are sent to the terminal port. Redirection of the I/O port is achieved by placing the specified value (0=SCI, 1=ACIA, 2=DUARTA, 3=DUARTB) into RAM location IODEV. Utility subroutines available to the user are listed in Table 3-1.

**Table 3-1. Utility Subroutine Jump Table**

| Address | Routine | Description |
|---|---|---|
| $FF7C | .WARMST | Go to ">" prompt point (skip BUFFALO... message). |
| $FF7F | .BPCLR | Clear breakpoint table. |
| $FF82 | .RPRINT | Display user's registers. |
| $FF85 | .HEXBIN | Convert ASCII character in A register to 4-bit binary number. Shift binary number into SHFTREG from the right. SHFTREG is a 2-byte (4 hexadecimal digits) buffer. If A register is not hexadecimal, location TMP1 is incremented and SHFTREG is unchanged. |
| $FF88 | .BUFFAR | Read 4-digit hexadecimal argument from input buffer to SHFTREG. |
| $FF8B | .TERMAR | Read 4-digit hexadecimal argument from terminal device to SHFTREG. |
| $FF8E | .CHGBYT | Write value (if any) from SHFTREG+1 to memory location pointed to by X. (Operation also applicable to EEPROM locations.) |
| $FF91 | .READBU | Read next character from INBUFF. |
| $FF94 | .INCBUF | Increment pointer into input buffer. |
| $FF97 | .DECBUF | Decrement pointer into input buffer. |
| $FF9A | .WSKIP | Read input buffer until non-whitespace character found. |
| $FF9D | .CHKABR | Monitor input for (CTRL)X, (DELETE), or (CTRL)W requests. |
| $FFA0 | .UPCASE | If character in accumulator A is lower case alpha, convert to upper case. |
| $FFA3 | .WCHEK | Test character in accumulator A and return with Z bit set if character is white space (space, comma, tab). |
| $FFA6 | .DCHEK | Test character in accumulator A and return with Z bit set if character is delimiter (carriage return or white space). |

**Table 3-1. Utility Subroutine Jump Table (continued)**

| Address | Routine | Description |
|---|---|---|
| $FFA9 | .INIT | Initialize I/O device. |
| $FFAC | .INPUT | Read I/O device. |
| $FFAF | .OUTPUT | Write I/O device. |
| $FFB2 | .OUTLHL | Convert left nibble of accumulator A contents to ASCII and output to terminal port. |
| $FFB5 | .OUTRHL | Convert right nibble of accumulator A contents to ASCII and output to terminal port. |
| $FFB8 | .OUTA | Output accumulator A ASCII character. |
| $FFBB | .OUT1BY | 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. |
| $FFBE | .OUT1BS | Convert binary byte at address in index register X to two ASCII characters and output followed by a space. Returns address in index register X pointing to next byte. |
| $FFC1 | .OUT2BS | Convert two consecutive binary bytes starting at 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. |
| $FFC4 | .OUTCRL | Output ASCII carriage return followed by a line feed. |
| $FFC7 | .OUTSTR | Output string of ASCII bytes pointed to by address in index register X until character is an end of transmission ($04). |
| $FFCA | .OUTST0 | Same as OUTSTR except leading carriage return and line feed is skipped. |
| $FFCD | .INCHAR | Input ASCII character to accumulator A and echo back. This routine loops until character is actually received. |
| $FFD0 | .VECINIT | Used during initialization to preset indirect interrupt vector area in RAM. This routine or a similar routine should be included in a user program which is invoked by the jump to $B600 feature of BUFFALO. |

When accessing BUFFALO utility routines, always reference the routines by the applicable address ($FF7C through $FFD0) in the jump table rather than the actual address in the BUFFALO monitor program. Jump table addresses remain the same when a new version of BUFFALO is developed even though the actual addresses of the routine may change. Programs that reference routines by the jump table addresses are not required to be changed to operate on revised versions of the BUFFALO monitor program.

### 3.2.5 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 an example:

```
FCB     3           3 characters in command name
FCC     'ASM'       ASCII literal command name string
FDB     ASM         Jump address for command module
```

Each command in the BUFFALO program is a individual module. Thus, to add or delete commands, all that is required is to include a new command module or delete an existing module and/or delete the entry in the command table. This procedure may be difficult to accomplish with the M68HC11EVBU because the BUFFALO monitor is contained in ROM. However with the standard M68HC11EVB, you may change commands, as this version of the BUFFALO monitor is contained in EPROM.

## 3.3 INTERRUPT VECTORS

Interrupt vectors residing in MCU internal ROM are accessible as follows. Each vector is assigned a three byte field residing in EVBU memory map locations $0000-$00FF. This is where the monitor program expects the MCU RAM to reside. Each vector points to a three byte field which is used as a jump table to the vector service routine. Table 3-2 lists the interrupt vectors and associated three byte field.

**Table 3-2. Interrupt Vector Jump Table**

| Interrupt Vector | Field |
|---|---|
| Serial Communications Interface (SCI) | $E0C4 – $E0C6 |
| Serial Peripheral Interface (SPI) | $E0C7 – $E0C9 |
| Pulse Accumulator Input Edge | $E0CA – $E0CC |
| Pulse Accumulator Overflow | $E0CD – $E0CF |
| Timer Overflow | $E0D0 – $E0D2 |
| Timer Output Compare 5 | $E0D3 – $E0D5 |
| Timer Output Compare 4 | $E0D6 – $E0D8 |
| Timer Output Compare 3 | $E0D9 – $E0DB |
| Timer Output Compare 2 | $E0DC – $E0DE |
| Timer Output Compare 1 | $E0DF – $E0E1 |

**Table 3-2. Interrupt Vector Jump Table (continued)**

| Interrupt Vector | Field |
|---|---|
| Timer Input Capture 3 | $E0E2 – $E0E4 |
| Timer Input Capture 2 | $E0E5 – $E0E7 |
| Timer Input Capture 1 | $E0E8 – $E0EA |
| Real Time Interrupt | $E0EB – $E0ED |
| IRQ | $E0EE – $E0F0 |
| XIRQ | $E0F1 – $E0F3 |
| Software Interrupt (SWI) | $E0F4 – $E0F6 |
| Illegal Opcode | $E0F7 – $E0F9 |
| Computer Operating Properly (COP) | $E0FA – $E0FC |
| Clock Monitor | $E0FD – $E0FF |

To use vectors specified in Table 3-2, you must insert a jump extended opcode in the three byte field of the vector required. For an example, for the IRQ vector, the following is performed:

1. Place $7E (JMP) at location $00EE.
2. Place IRQ service routine address at locations $00EF and $00F0.
3. The following is an example where the IRQ service routine starts at $0100:

```
$00EE    7E  01  00    JMP IRQ SERVICE
```

During initialization BUFFALO checks the first byte of each set of three locations. If a $7E jump opcode is not found, BUFFALO will install a jump to a routine called STOPIT. This assures there will be no uninitialized interrupt vectors which would cause undesirable operation during power up and power down. If an interrupt is accidentally encountered, the STOPIT routine will force a STOP instruction sequence to be executed. A user may replace any of the JMP STOPIT instructions with a JMP to a user written interrupt service routine. If a reset is issued via switch S1, BUFFALO will not overwrite these user jump instructions so they need not be re-initialized after every reset.

# CHAPTER 4
# OPERATING INSTRUCTIONS

## 4.1 INTRODUCTION

This chapter provides the necessary information to initialize and operate the EVBU. 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.

## 4.2 CONTROL SWITCH

Use user reset switch S1 to reset the EVBU and MCU circuits. This switch is a momentary action pushbutton switch.

## 4.3 LIMITATIONS

The default baud rate for the MC68HC11 MCU SCI is 9600. using a 2 MHz E clock. You can reset the baud rate using the memory modify (MM) command to reprogram the BAUD register in the MCU. Or by using instructions in the user program.

The EVBU can transfer data faster than some terminal devices can receive, which at certain times, can cause missing characters on the terminal display screen. Memory display (MD), trace (TRACE), and help (HELP) commands may be affected by this problem. You can either ignore the problem, switch to a slower baud rate, or use a different communications program. When using the MD or TRACE commands, the missing character problem can be resolved by displaying fewer address locations or tracing fewer instructions at a time, respectively.

The monitor program uses the MCU internal RAM located at $0048-$00FF. The control registers are located at $1000-$103F. The monitor program also uses Output Compare 5 (OC5) for the TRACE instruction, therefore TRACE cannot be used in user programs which use OC5. Since PROCEED AND STOPAT commands indirectly use the TRACE function these commands also rely on the OC5 to XIRQ connection via jumper header J7.

The EVBU allows you to use all the features of the BUFFALO evaluation software, however it should be noted (when designing code) that BUFFALO uses the MCU on-chip RAM locations $0047-$00FF leaving approximately 325 bytes for the user (i.e., $0000-$0047 and $0100-$01FF). 512 bytes of EEPROM ($B600-B7FF) and approximately 325 bytes of RAM ($0000-0047) + ($0100-$01FF) are available for user developed software.

Table 4-1 is a lists of the BUFFALO monitor memory map limitations.

**Table 4-1. Monitor Memory Map Limitations**

| Address | Restrictions |
|---|---|
| \$0000–\$0047 | Available to user. (BUFFALO sets default value of the user stack pointer at location \$0047.) |
| \$0048–\$0065 | BUFFALO monitor stack area. |
| \$0066–\$00C3 | BUFFALO variables. (Refer to the buf32.asm file on the EVBU diskette). |
| \$00C4–\$00FF | Interrupt pseudo vectors (jumps). |
| \$0100–\$01FF | User available. |
| \$1000–\$103F | MCU control registers. Although RAM and registers can be moved in the memory map, BUFFALO expects RAM at \$0000 (actually requires \$0048-\$00FF) and registers at \$1000-\$103F. |
| \$4000 | Some versions of EVBs have a D flip-flop addressed at this location. During initialization, BUFFALO 3.2 writes \$00 to location \$4000 and various monitor operations cause \$00 or \$01 to be written to \$4000 to retain compatibility. (Refer to the buf32.asm file on the EVBU diskette for additional information on DFLOP, TARGCO, and HOSTCO).<br><br>Since the EVBU has no memory or peripherals located at \$4000, these writes should not concern most EVBU users. |
| \$9800–\$9801 | BUFFALO supports serial I/O to a terminal via a ACIA (external IC) located at \$9800 in the memory map. During initialization, BUFFALO 3.2 reads and writes to location \$9800 and 9801 to see if a ACIA is present in the system. If a ACIA is installed on the EVBU wire-wrap area and connected to the MCU, refer to the buf32.asm file on the EVBU diskette to understand implications. |
| \$D000–\$D00F | BUFFALO supports serial I/O to a terminal and/or host via a DUART (external IC) located at \$D000 in the memory map. During initialization, BUFFALO 3.2 reads and writes to location \$D00C to see if a DUART is present in the system. If a DUART is installed on the EVBU wire-wrap area and connected to the MCU, refer to the buf32.asm file on the EVBU diskette. |

## 4.4 OPERATING PROCEDURES

The EVBU is a simplified debugging/evaluating tool designed for debugging user programs and evaluation of MC68HC11 family devices. Use jumper header J2 to determine weather the BUFFALO monitor prompt is displayed, or if a jump to internal EEPROM is executed. Refer to paragraph 2.3.1 for additional program execution selection information.

At reset, the monitor detects the state of the PE0 line. If a low state is detected, the monitor program is executed and the prompt displayed. If a high state is detected, the monitor automatically jumps directly to EEPROM (address location $B600) and executes user program code without displaying the monitor prompt.

### 4.4.1 Debugging/Evaluation

The debugging/evaluation operation lets you debug user code under control of the BUFFALO monitor program. User code can be assembled in one of two methods. The first method assembles code using the BUFFALO monitor one-line assembler.

The second method, you assemble code on a host computer and then download it to the EVBU user RAM or EEPROM in Motorola S-record format. The monitor program is then used to debug the assembled user code.

A download to EEPROM will work if the baud rate is slow enough to allow EEPROM programming. Since erasure and programming both require 10 milliseconds, a slow baud rate (300 baud) will have to be used to ensure enough time between characters. If the EEPROM is bulk erased prior to downloading, 600 baud allows enough time between characters.

### 4.4.2 Alternate Baud Rates

The following text assumes that a personal computer (PC) is used as the terminal device, and a user program is assembled on the PC to produce an S-record object file which is to be downloaded into EVBU EEPROM. For this example, no assumptions are made about the previous EEPROM contents. During the download operation, BUFFALO determines (on a byte-by-byte basis) whether or not erasure is required prior to programming a downloaded value into each EEPROM location. Since erasure and programming both require 10 milliseconds, a slow baud rate (300 baud) will have to be used to ensure enough time between characters.

At the start of this procedure, the PC will be operating a terminal emulator program such as PROCOMM, KERMIT, MacTerminal, or Red Ryder; and the BUFFALO prompt > will be displayed. First change the BAUD register from $30 (selects default 9600 baud rate for SCI) to $35 (to select 300 baud) with a memory modify (MM) command as follows:

```
>MM 102B<CR>  BOLD entries are user entered on the terminal keyboard.
102B 30 35<CR>
```

Since the communication baud rate changes when pressing the carriage return **<CR>** key after typing 35, you may observe a few invalid characters on the terminal display screen which can be ignored.

Next change the communication program baud rate to 300 baud. If using an IBM-PC with a PROCOMM terminal emulator program, use the alt-P window. Hold down the (alt ) keyboard key while pressing the p keyboard key. A window appears on the terminal display screen to change the baud rate. If using an Apple Macintosh with a MacTerminal terminal emulator program, use the pull-down menu to change the baud rate.

Next press the keyboard **<CR>** key to resume communications with the EVBU as follows:

```
<CR>
102B 35<CR>
>
```

At this point all BUFFALO commands should operate normally except the display will be noticeably slower due to the slow baud rate.

To download the S-record file to the EVBU EEPROM, type the LOAD T command and a carriage return as follows:

```
>LOAD T<CR>
```

At this point BUFFALO is waiting for the S-record file from the PC. Instruct the PC to send the S-record file to the EVBU using simple ASCII file transfer protocol. If using an IBM-PC, use the alt-S window to setup the American standard code for information interchange (ASCII) transfer perimeters (this only needs to be performed once). To invoke the file transfer, press the page-up keyboard key (may be shared with the 9 key on the numeric keypad on some PC keyboards), and follow instructions on the display screen. If using a Apple Macintosh, use the pull-down menu to send the file.

Upon completion of the S-record transfer, the following message is displayed on the terminal display screen:

```
done
>
```

After downloading the S-record file, the EVBU may be reset (via S1) to return to 9600 baud operation. When the EVBU is returned to 9600 baud operation, the terminal emulator must also be changed back to 9600 baud operation.

### 4.4.3 Monitor Program

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

**NOTE**

EVBU contains no hardware to support the host related commands.
(e.g., ACIA, DUART)

A standard input routine controls the EVBU operation while you enter a command line. Command processing begins only after the command line has been terminated by depressing the keyboard carriage return **<CR>** key.

## 4.5 COMMAND LINE FORMAT

The command line format is as follows:

```
><command> [<parameters>]<CR>
```

where:

| | |
|---|---|
| > | EVBU monitor prompt. |
| <command> | Command mnemonic (single letter for most commands). |
| <parameters> | Expression or address. |
| **<CR>** | ENTER keyboard key – depressed to enter command. |

**NOTES**

1. The command line format is defined using special characters which have the following syntactical meanings:

   | | |
   |---|---|
   | < > | Enclose syntactical variable |
   | [ ] | Enclose optional fields |
   | [ ]... | 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 of 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. Pressing <CR> will repeat the most recent command. The LOAD command is an exception.

## 4.6 MONITOR COMMANDS

The BUFFALO monitor commands are listed alphabetically, by mnemonic, in Table 4-2. Each of the commands are described in detail following Table 4-2. In most cases the initial single letter of the command mnemonic or a specific symbol can be used. A minimum number of characters must be entered to at least guarantee uniqueness from other commands (i.e., MO = MOVE, ME = MEMORY). If the letter M is entered, BUFFALO uses the first command in Table 4-2 that starts with M.

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 <lf> |
| (CTRL)W | Wait/freeze screen (1) |
| (CTRL)X | Abort/cancel command |
| (DELETE) | Abort/cancel command |
| **<CR>** | 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:

**BOLD** entries are user-entered on the terminal keyboard.

Command line input is entered when the carriage return key **<CR>** key is depressed.

Typical example of this explanation is:

```
>MD F000 F100<CR>
```

## Table 4-2. Monitor Program Commands

| Command | Description |
|---|---|
| **ASM** [<address>] | Assembler/disassembler |
| *ASSEM* | *(same as ASM)* |
| **BF** <addr1> <addr2> <data> | Block fill memory with data |
| **BR** [-] [<address>]... | Breakpoint set |
| *BREAK* | *(same as BR)* |
| **BULK** | Erase all EEPROM locations |
| *BULKA* | *(same as BULKALL)* |
| **BULKALL** | Bulk erase EEPROM + CONFIG register (1) |
| **CALL** [<address>] | Execute subroutine |
| *COPY* | *(same as MOVE)* |
| *DUMP* | *(same as MD)* |
| *ERASE* | *(same as BULK)* |
| *FILL* | *(same as BF)* |
| **G** [<address>] | Execute program |
| *GO* | *(same as G)* |
| **HELP** | Display monitor commands |
| *HOST* | *(same as TM)* |
| **LOAD** <T> | Download (S-records*) via terminal port (2) |
| *MEMORY* | *(same as MM)* |
| **MD** [<addr1> [<addr2>]] | Dump memory to terminal |
| **MM** [<address>] | Memory modify |
| **MOVE** <addr1> <addr2> [<dest>] | Move memory to new location |
| **P** | Proceed/continue from breakpoint |
| *PROCEED* | *(same as P)* |
| *RD* | *(same as RM)* |
| *READ* | *(same as MOVE)* |

Table 4-2. Monitor Program Commands (continued)

| Command | Description |
|---|---|
| *REGISTER* | *(same as RM)* |
| **RM** [p,x,y,a,b,c,s] | Register modify/display user registers |
| **STOPAT** <address> | Stop at address |
| **T** [<n>] | Trace $1-$FF instructions |
| **TM** | Enter transparent mode |
| *TRACE* | *(same as T)* |
| **VERIFY** <T> | Compare memory to download data via terminal port |
| **XBOOT** [<address1> [<address2>]] | Send program to another M68HC11 via bootstrap mode |
| *?* | *(same as HELP)* |
| *[<address>]/* | *(same as MM [<address>])* |

**NOTES**

1. On newer MC68HC11 mask sets, CONFIG can only be changed in special test or bootstrap modes of operation.
2. * Refer to Appendix A for S-record information.

## ASM Assembler/Disassembler

### 4.6.1 Assembler/Disassembler

```
ASM [<address>]
```

where:

| | |
|---|---|
| <address> | is the starting address for the assembler operation. Assembler operation defaults to internal RAM if no address is given. |

The assembler/disassembler is an interactive assembler/editor. Each source line is converted into the proper machine language code and is stored in memory overwriting previous data on a line-by-line basis at the time of entry. In order to display an instruction, the machine code is disassembled and the instruction mnemonic and operands are displayed. All valid opcodes are converted to assembly language mnemonics. All invalid opcodes are displayed on the terminal CRT as "ILLOP".

The syntax rules for the assembler are as follows: (a.) All numerical values are assumed to be hexadecimal. Therefore no base designators (e.g., $ = hex, % = binary, etc.) are allowed. (b.) Operands must be separated by one or more space or tab characters. (c.) Any characters after a valid mnemonic and associated operands are assumed to be comments and are ignored.

Addressing modes are designated as follows: (a.) Immediate addressing is designated by preceding the address with a # sign. (b.) Indexed addressing is designated by a comma. The comma must be preceded a one byte relative offset (even if the offset is 00), and the comma must be followed by an X or Y designating which index register to use (e.g., LDAA 0,X). (c.) Direct and extended addressing is specified by the length of the address operand (1 or 2 digits specifies direct, 3 or 4 digits specifies extended). Extended addressing can be forced by padding the address operand with leading zeros. (d.) Relative offsets for branch instructions are computed by the assembler. Therefore the valid operand for any branch instruction is the branch-if-true address, not the relative offset.

When a new source line is assembled, the assembler overwrites what was previously in memory. If no new source line is submitted, or if there is an error in the source line, then the contents of memory remain unchanged. Four instruction pairs have the same opcode, so disassembly will display the following mnemonics:

Arithmetic Shift Left (ASL)/Logical Shift Left (LSL) displays as ASL
Arithmetic Shift Left Double (ASLD)/Logical Shift Left Double (LSLD) displays as LSLD
Branch if Carry Clear (BCC)/Branch if Higher or Same (BHS) displays as BCC
Branch if Carry Set (BCS)/Branch if Lower (BLO) displays as BCS

## ASM Assembler/Disassembler

If the assembler tries to assemble at an address that is not in RAM or EEPROM, an invalid address message "rom-xxxx" is displayed on the terminal CRT (xxxx = invalid address).

Assembler/disassembler subcommands are as follows. If the assembler detects an error in the new source line, the assembler will output an error message and then reopen the same address location.

| | |
|---|---|
| / | Assemble the current line and then disassemble the same address location. |
| ^ | Assemble the current line and then disassemble the previous sequential address location. |
| <CR> | Assemble the current line and then disassemble the next opcode address. |
| (CTRL)J | Assemble the current line. If there isn't a new line to assemble, then disassemble the next sequential address location. Otherwise, disassemble the next opcode address. |
| (CTRL)A | Exit the assembler mode of operation. |

**Examples:**

```
>ASM 0100<CR>

0100 STX  $FFFF    >LDAA #55<CR>  Immediate mode addressing, requires #
     86 55                        before operand.
0102 STX  $FFFF    >STAA C0<CR>   Direct mode addressing.
     97 C0
0104 STX  $FFFF    >LDS 0,X<CR>   Index  mode, if offset = 0 (,X) will not
     AE 00                        be accepted.
0106 STX  $FFFF    >BRA C500<CR>  Branch out of range message.

Branch out of range
0106 STX  $FFFF    >BRA 0130<CR>  Branch offsets calculated automatically,
     20 28                        address required as branch operand.
0108 STX  $FFFF    >(CTRL)A       Assembler operation terminated.
>
```

**NOTE**

In the above example memory locations $0100-$0108 previously contained $FF data which disassembles to `STX  $FFFF`.

## BF Block Fill

### 4.6.2 Block Fill

```
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 lets you repeat a specific pattern throughout a determined user memory range in RAM or EEPROM. If an invalid address is specified, an invalid address message "rom-xxxx" is displayed on the terminal CRT (xxxx = invalid address).

**Examples:**

| | |
|---|---|
| **>BF 0100 01FF FF<CR>** | Fill each byte of memory from 0100 through 01FF with data pattern FF. |
| **>BF B700 B700 0<CR>** | Set location B700 to 0. |

## BR Breakpoint Set

### 4.6.3 Breakpoint Set

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

where:

| | |
|---|---|
| [-] 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 address 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 opcodes at the breakpoint locations before returning control back to the monitor program.

SWI opcodes cannot be executed or breakpointed in user code because the monitor program uses the SWI vector. RAM or EEPROM locations can be breakpointed.

| 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>. |

## BR Breakpoint Set

**Examples:**

| | |
|---|---|
| `>`**`BR 0103<CR>`** | Set breakpoint at address location 0103. |

```
0103 0000 0000 0000
>
```

| | |
|---|---|
| `>`**`BR 0103 0105 0107 0109<CR>`** | Sets four breakpoints. Breakpoints at same address results in only one breakpoint being set. |

```
0103 0105 0107 0109
>
```

| | |
|---|---|
| `>`**`BR<CR>`** | Display all current breakpoints. |

```
0103 0105 0107 0109
>
```

| | |
|---|---|
| `>`**`BR -0109<CR>`** | Remove breakpoint at address location 0109. |

```
0103 0105 0107 0000
>
```

| | |
|---|---|
| `>`**`BR – 0109<CR>`** | Clear breakpoint table and add 0109. |

```
0109 0000 0000 0000
>
```

| | |
|---|---|
| `>`**`BR -<CR>`** | Remove all breakpoints. |

```
0000 0000 0000 0000
>
```

| | |
|---|---|
| `>`**`BR E000<CR>`** | Only RAM or EEPROM locations can be breakpointed. |
| `rom-E000` | Invalid address message. |

```
0000 0000 0000 0000
>
```

| | |
|---|---|
| `>`**`BR 0105 0107 0109 0111 0113<CR>`** | Maximum of four breakpoints can be set. |
| `Full` | Buffer full message. |

```
0105 0107 0109 0111
>
```

## BULK Erase All EEPROM Locations

### 4.6.4 Erase All EEPROM Locations

The BULK command lets you erase all MCU EEPROM locations ($B600-$B7FF). A delay loop is built in such that the erase time is 10 ms when running at 2 MHz E clock.

**NOTE**

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

**Example:**

```
>BULK<CR>                 Bulk erase all MCU EEPROM locations ($B600-$B7FF).
>                         Prompt indicates erase sequence completed.
```

## BULKALL Erase All EEPROM & Config. Reg.

### 4.6.5 Erase All EEPROM Locations & the CONFIG Register

```
BULKALL
```

The BULKALL command lets you erase all MCU EEPROM locations ($B600-$B7FF) including the configuration (CONFIG) register location ($103F) on older MC68HC11A8 MCU mask sets. A delay loop is built in such that the erase time is about 10 ms when running at 2 MHz E clock. The MC68HC11E9 MCU CONFIG register cannot be changed in normal operating modes.

**NOTE**

No erase verification message will be displayed upon completion of the bulkall EEPROM and configuration register erase operation. User must verify erase operation by examining EEPROM locations or the configuration register location using the MM or MD command.

**Example:**

```
>BULKALL<CR>              Bulk erase all MCU EEPROM ($B600-$B7FF) and
                          configuration register ($103F) locations.
>                         Prompt indicates erase sequence completed.
```

## CALL Execute Subroutine

### 4.6.6 Execute Subroutine

```
CALL [<address>]
```

where:

| | |
|---|---|
| <address> | the starting address where user subroutine begins. |

The CALL command allows the user to execute a user subroutine program. 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 BUFFALO monitor calls the subroutine so that the first unmatched return from subroutine (RTS) encountered will return control back to the monitor program. Thus any user subroutine can be called and executed via the monitor program. Program execution continues until an unmatched RTS is encountered, a breakpoint is encountered, or the EVBU reset switch S1 is activated (pressed).

**EXAMPLE PROGRAM: for CALL, G, P, and STOPAT command examples**

```
>ASM 0100<CR>

0100 STX  $FFFF    >LDAA #44<CR>
86 44
0102 STX  $FFFF    >STAA 01FC<CR>
B7 01 FC
0105 STX  $FFFF    >NOP<CR>
01
0106 STX  $FFFF    >NOP<CR>
01
0107 STX  $FFFF    >NOP<CR>
01
0108 STX  $FFFF    >RTS<CR>
39
0109 STX  $FFFF    >(CTRL)A
```

## CALL Execute Subroutine

**Example:**

```
>CALL 0100<CR>         Execute program subroutine.

P-0100 Y-DEFE X-F4FF A-44 B-FE C-D0 S-0047  Displays register status at time
>                      RTS encountered (except P register contains original call
                       address or a breakpoint address if encountered).
```

## G Execute Program

### 4.6.7 Execute Program

```
G [<address>]
```

where:

| | |
|---|---|
| <address> | is the starting address where program execution begins. |

The G command allows the user to initiate 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 is encountered, or the EVBU reset switch S1 is activated (pressed).

**NOTE**

Refer to example program shown on **page 4-16** and insert breakpoints at locations $0105 and $0107 for the following G command example.

**Example:**

```
>G 0100<CR>                    Begin program execution at PC address location 0100.
P-0105 Y-DEFE X-F4FF A-44 B-FE C-D0 S-0047  Breakpoint encountered at
>                                           0105.
```

### 4.6.8 Help

```
HELP
```

The HELP command enables the user available EVBU command information to be displayed on the terminal CRT for quick reference purposes.

**EXAMPLE**

```
>HELP<CR>

ASM [<addr>]  Line assembler/disassembler.
    /        Do same address.      ^      Do previous address.
    CTRL-J   Do next address.    RETURN   Do next opcode.
    CTRL-A   Quit.
BF <addr1> <addr2> [<data>]  Block fill.
BR [-][<addr>]  Set up breakpoint table.
BULK  Erase the EEPROM.      BULKALL  Erase EEPROM and CONFIG.
CALL [<addr>]  Call user subroutine. G [<addr>]  Execute user code.
LOAD, VERIFY [T] <host download command>     Load or verify S-records.
MD [<addr1> [<addr2>]]  Memory dump.
MM [<addr>]  Memory modify.
    /        Open same address.      CTRL-H or  ^  Open previous address.
    CTRL-J   Open next address. SPACE       Open next address.
    RETURN   Quit.              <addr>O     Compute offset to <addr>.
MOVE <s1> <s2> [<d>]  Block move.
P  Proceed/continue execution.
RM [P, Y, X, A, B, C, or S]  Register modify.
T [<n>]  Trace n instructions.
TM  Transparent mode (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 Load S-Records

### 4.6.9 Load S-Records

```
LOAD <host download command>  (NOT APPLICABLE TO EVBU)
LOAD <T>
```

where:

| | |
|---|---|
| <host download command> | download S-records via host port. |
| <T> | download S-records to EVBU via terminal port. |

**NOTE**

As equipped from the factory the EVBU only supports the LOAD T variation of the load command

The LOAD command moves (downloads) object data in S-record format (see Appendix A) from an external host computer to the EVBU. As the EVBU 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 an invalid memory location, the invalid address message "error addr xxxx" is displayed on the terminal CRT (xxxx = invalid address).

**Examples:**

```
>LOAD T<CR>             LOAD command entered to download data from host computer
done                    to EVBU via terminal port.
>
```

```
>LOAD T<CR>             LOAD command entered.
error addr E000         Invalid address message.
>                       S-records must be downloaded  into  RAM or EEPROM.
```

Refer to paragraph 4.8 DOWNLOADING PROCEDURES for additional information pertaining to the use of the LOAD command. Refer to paragraph 4.4.2 Alternate Baud Rates for information pertaining to slower baud rates which are required when downloading directly to EEPROM.

### 4.6.10 Memory Display

```
MD [<address1> [<address2>]]
```

where:

| | |
|---|---|
| <address1> | Memory starting address (optional). |
| [<address2>] | Memory ending address (optional). |

The MD command lets you 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 defaults to the first address. If no addresses are specified, 9 lines of 16 bytes are displayed near the last memory location accessed.

Each displayed line of memory consists of a four digit hexadecimal address (applicable to the memory location displayed), followed by 16 two digit hexadecimal values (contents of the sixteen memory locations), followed by the ASCII equivalents (if applicable) of the the 16 memory locations. Since not all 8-bit values correspond to a displayable ASCII character, some of the character positions at the end of a line may be blank.

## MD Memory Display

### EXAMPLES

```
>MD E61F<CR>

E610 F1 34 02 54 4D EE E4 04 54 45 53 54 FE 29 FF 42  4 TM   TEST ) B
E620 55 46 46 41 4C 4F 20 33 2E 32 20 28 69 6E 74 29 UFFALO 3.2 (int)
E630 20 2D 20 42 69 74 20 55 73 65 72 20 46 61 73 74 –  Bit User Fast
E640 20 46 72 69 65 6E 64 6C 79 20 41 69 64 20 74 6F  Friendly Aid to
E650 20 4C 6F 67 69 63 61 6C 20 4F 70 65 72 61 74 69  Logical Operati
E660 6F 6E 04 57 68 61 74 3F 04 54 6F 6F 20 4C 6F 6E on What? Too Lon
E670 67 04 46 75 6C 6C 04 4F 70 D2 20 04 72 6F 6D 2D g Full Op-  rom-
E680 04 43 6F 6D 6D 61 6E 64 3F 04 42 62 64 20 61 72  Command? Bad ar
E690 67 75 60 65 6E 74 04 4E 6F 20 68 6F 73 74 20 70 gument No host p
>

>MD 0130 0120<CR>

0130    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
>

>MD 0100 0120<CR>

0100    86 04 B7 01 FC 01 01 01 39 FF FF FF FF FF FF FF           9
0110    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0120    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
>
```

## MM Memory Modify

### 4.6.11 Memory Modify

```
MM [<address>]
```

where:

| | |
|---|---|
| <address> | 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. The MM command will also erase any EEPROM location, and will reprogram the location with the corresponding value (EEPROM locations treated as if RAM).

Once entered, the MM command has several submodes of operation that allow modification and verification of data. The following subcommands are recognized.

| | |
|---|---|
| (CTRL)J or (SPACE BAR) or + | Examine/modify next location. |
| (CTRL)H or ^ or - | Examine/modify previous location. |
| / | Reexamine/modify same location. |
| **<CR>** | Terminate MM operation. |
| O | Compute branch instruction relative offset. |

If an attempt is made to change an invalid address, the invalid address message "rom" is displayed on the terminal CRT. An invalid address is any memory location which cannot be read back immediately after a change in order to verify that change.

## MM Memory Modify

**Examples:**

```
>MM 0180<CR>              Display memory location 0180.

0180 FF 66/<CR>           Change data at 0180 and reexamine location.
0180 66 55 ^<CR>          Change data at 0180 and backup one location.
017F FF AA<CR>            Change data at 017F and terminate MM operation.


>MM 013C<CR>              Display memory location.

013C FF 018EO<CR> 51      Compute offset, result = $51.
013C FF


>MM 0100<CR>              Examine location $0100.

0100 86 04 B7 01 FC 01         Examine next location(s) using (SPACE BAR).


>MM B700<CR>              Examine EEPROM location $B700.

B700 FF 52<CR>            Change data at location $B700.

>MM B700<CR>              Reexamine EEPROM location $B700.

B700 52
>
```

## MOVE Move Memory

### 4.6.12 Move Memory

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

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 locations. 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 EEPROM cells.

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

**Example:**

```
>MOVE E000 E0FF 0100<CR>     Move data from locations $E000-$E0FF to locations
                             $0100-$01FF.
>
```

## P Proceed/Continue from Breakpoint

### 4.6.13 Proceed/Continue

```
P
```

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**

Refer to example program shown on **page 4-16** for the following P command example. Breakpoints have been inserted at locations $0105 and $0107 (refer to example on **pages 4-16 and 4-17**).

**Example:**

```
>G 0100<CR>                                Start execution at 0100.

P-0105 Y-DEFE X-F4FF A-44 B-FE C-D0 S-0047  Breakpoint encountered at
                                           0105. Continue execution.
>P<CR>

P-0107 Y-DEFE X-F4FF A-44 B-FE C-90 S-0047  Breakpoint encountered at
                                           0107.
>
```

### 4.6.14 Register Modify/Display

```
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), condition code register (C), and stack pointer (S) register contents.

**Examples:**

```
>RM<CR>                    Display P register contents.
P-0108 Y-7982 X-FF00 A-44 B-70 C-C0 S-0047
P-0108 0100<CR>            Modify P register contents.

>


>RM X<CR>                  Display X register contents.
P-0100 Y-7982 X-FF00 A-44 B-70 C-C0 S-0047
X-FF00 1000<CR>            Modify X register contents.

>


>RM<CR>                    Display P register contents.
P-0100 Y-7982 X-1000 A-44 B-70 C-C0 S-0047
P-0100 (SPACE BAR)         Display remaining registers.
Y-7982 (SPACE BAR)
X-1000 (SPACE BAR)
A-44 (SPACE BAR)
B-70 (SPACE BAR)
C-C0 (SPACE BAR)
S-0047 (SPACE BAR)         Entering a  (SPACE BAR) following the stack pointer
>                          display will terminate RM command.
```

### 4.6.15 Stop at Address

```
STOPAT <address>
```

where:

| | |
|---|---|
| <address> | The specified user program counter (PC) stop address. |

The STOPAT command causes a user program to be executed one instruction at at time until the specified address is encountered. Execution begins with the current user PC address and stops just before execution of the instruction at the specified stop address. The STOPAT command should only be used when the current value of the user PC register is known. (e.g., after a breakpoint is reached or after an RD command is used to set the user PC)

The STOPAT command has an advantage over breakpoints in that a stop address can be a ROM location while breakpoints only operate in RAM or EEPROM locations. Since the STOPAT command traces one instruction at a time with a hidden return to the monitor after each user instruction, some user programs will appear to execute slowly.

The stop address specified in the STOPAT command must be the address of an opcode just as breakpoints can only be set at opcode addresses.

**NOTE**

Refer to example program shown on **page 4-16** for the following STOPAT command example. The RD command was used prior to this example to set the user PC register to $0100.

**Example:**

```
>STOPAT 0108<CR>        Execute example program until $0108 is reached.
P-0108 Y-DEFE X-F4FF A-44 B-FE C-90 S-0047
>
```

### 4.6.16 Trace

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

where:

| | |
|---|---|
| <n> | is the number (in hexadecimal, $1-FF max.) of instructions to execute. A default value of 1 is used if <n> is not specified. |

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). Each event message line includes a disassembly of the instruction that was traced and a register display showing the CPU state after execution of the traced instruction. The trace command operates by setting the OC5 interrupt to time out after the first cycle of the first user opcode fetched.

**NOTE**

The RD command was used to set the user PC register to $FF85 prior to starting the following trace examples.

**SINGLE TRACE EXAMPLE**

```
>T<CR>
JMP            $E1F7 P-E1F7 Y-FFFF X-FFFF A-44 B-FF C-10 S-0046
>
```

**MULTIPLE TRACE EXAMPLES**

```
>T 2<CR>
PSHA                 P-E1F8 Y-FFFF X-FFFF A-44 B-FF C-10 S-0046
PSHB                 P-E1F9 Y-FFFF X-FFFF A-44 B-FF C-10 S-0045
>T 3<CR>
PSHX                 P-E1FA Y-FFFF X-FFFF A-44 B-FF C-10 S-0043
JSR   $E19D          P-E19D Y-FFFF X-FFFF A-44 B-FF C-10 S-0041
CMPA  #$61           P-E19F Y-FFFF X-FFFF A-44 B-FF C-19 S-0041
>T 4<CR>
BLT   $E1A7          P-E1A7 Y-FFFF X-FFFF A-44 B-FF C-19 S-0041
RTS                  P-E1FD Y-FFFF X-FFFF A-44 B-FF C-19 S-0043
CMPA  #$30           P-E1FF Y-FFFF X-FFFF A-44 B-FF C-19 S-0043
BLT   $E223          P-E223 Y-FFFF X-FFFF A-44 B-FF C-19 S-0043
>
```

## TM Transparent Mode

### 4.6.17 Transparent Mode

```
TM (NOT APPLICABLE TO EVBU)
```

The TM command connects the EVBU host port to the terminal port, which allows direct communication between the terminal and a host computer. All I/O between the ports are ignored by the EVBU until the exit character is entered from the terminal.

The TM subcommands are as follows:

| | |
|---|---|
| (CTRL)A | Exit from transparent mode. |
| (CTRL)B | Send break to host computer. |

**NOTE**

TM command can only be used if a host I/O port is installed on the EVBU wire-wrap area.

**Example**

```
>TM<CR>                   Enter transparent mode.

appslab login: bill<CR>  Host computer login response.
Password:xxxxxxxx<CR>    Host computer password.


"System Message"


$
.
.
.
$(CTRL)A                  Task completed. Enter exit command.
>                         Exit transparent mode.
```

### 4.6.18 Verify

```
VERIFY <host download command>     (NOT APPLICABLE TO EVBU)
VERIFY <T>
```

where:

| | |
|---|---|
| <host download command> | compare memory to host port download data. |
| <T> | compare memory to terminal port download data. |

**NOTE**

As equipped from the factory the EVBU only supports the VERF <T> variation of the verify command.

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

| EXAMPLES | DESCRIPTION |
|---|---|
| >**VERIFY T<CR>** | Enter verify command. |
| Done | Verification completed. |
| > | |

| EXAMPLES | DESCRIPTION |
|---|---|
| >**VERIFY T<CR>** | Enter verify command. |
| | Mismatch encountered. |
| error addr E000 | Error message displaying first address that failed to verify. |
| > | |

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

### 4.6.19 Transfer Data Bootstrap Mode

```
XBOOT [<address1> [<address2>]]
```

where:

| | |
|---|---|
| <address1> | Starting address. |
| <address2> | Ending address. |

The XBOOT command loads/transfers a block of data from address1 through address2 via the serial communications interface (SCI) to another MC68HC11 MCU device which has been reset in the bootstrap mode. A leading control character of $FF is sent prior to sending the data block. This control character is part of the bootstrap mode protocol and establishes the baud rate for the rest of the transfer.

If only one address is provided, the address will be used as the starting address and the block size will default to 256 bytes. If no addresses are provided, the block of addresses from $C000 through $C0FF is assumed by the BUFFALO monitor program.

**NOTE**

The MC68HC11A8 MCU requires a fixed block size of 256 bytes for bootloading while the MC68HC11E9 MCU can accept a variable length block of 1 to 512 bytes.

The XBOOT command generates SCI transmitter output signals at 7812.5 baud which are intended for another MC68HC11 MCU device operating in the bootstrap mode. These signals appear as nonsense data to the terminal display used for normal communication with the EVBU. After using the XBOOT command the EVBU must be reset by pressing the reset switch S1 before normal communications can resume.

## XBOOT Transfer Data Bootstrap Mode

The following procedure describes the use of the XBOOT command. Before initiating the XBOOT command, the EVBU should be prepared as follows:

1. Cut J9 cut-trace short on PCB solder side at location J9.
2. Install fabricated jumper on jumper header J9, pins 1 and 2.

After preparing the EVBU, perform the following:

1. Assemble or fill EVBU MCU EEPROM (locations $B600-$B6FF) with program to be bootloaded (transmitted/transferred) to target MC68HC11 MCU device.
2. Enter XBOOT command and addresses without pressing carriage return <CR> key as follows:

   ```
   >XBOOT B600 B6FF(Do not press the ENTER key.)
   ```

3. Remove previously installed fabricated jumper from jumper header J9.
4. Connect jumper wire from jumper header J9 pin 2 to RxD input of target MC68HC11 MCU device.
5. Reset target MC68HC11 MCU device in bootstrap mode.
6. Press carriage return <CR> key to invoke XBOOT command.

   Since TxD is not connected to the terminal, the user will not observe any changes on the terminal display CRT. The bootload process takes approximately a third of a second to finish.

7. Disconnect jumper wire installed in step d.
8. Install fabricated jumper removed in step c.
9. Press EVBU reset switch S1 to restore normal EVBU operation.

## 4.7 ASSEMBLY/DISASSEMBLY PROCEDURES

The assembler/disassembler is an interactive assembler/editor. Each source line is converted into the proper machine language code and is stored in memory overwriting previous data on a line-by-line basis at the time of entry. In order to display an instruction, the machine code is disassembled and the instruction mnemonic and operands are displayed. All valid opcodes are converted to assembly language mnemonics. All invalid opcodes are displayed on the terminal CRT as "ILLOP".

The syntax rules for the assembler are as follows:

- All numerical values are assumed to be hexadecimal. Therefore no base designators (e.g., $ = hex, % = binary, etc.) are allowed.
- Operands must be separated by one or more space or tab characters.
- Any characters after a valid mnemonic and associated operands are assumed to be comments and are ignored.

Addressing modes are designated as follows:

- Immediate addressing is designated by preceding the address with a # sign.
- Indexed addressing is designated by a comma. The comma must be preceded a one byte relative offset (even if the offset is 00), and the comma must be followed by an X or Y designating which index register to use (e.g., LDAA 0,X).
- Direct and extended addressing is specified by the length of the address operand (1 or 2 digits specifies direct, 3 or 4 digits specifies extended). Extended addressing can be forced by padding the address operand with leading zeros.
- Relative offsets for branch instructions are computed by the assembler. Therefore the valid operand for any branch instruction is the branch-if-true address, not the relative offset.

When a new source line is assembled, the assembler overwrites what was previously in memory. If no new source line is submitted, or if there is an error in the source line, then the contents of memory remain unchanged. Four instruction pairs have the same opcode, so disassembly will display the following mnemonics:

- Arithmetic Shift Left (ASL)/Logical Shift Left (LSL) displays as ASL
- Arithmetic Shift Left Double (ASLD)/Logical Shift Left Double (LSLD) displays as LSLD
- Branch if Carry Clear (BCC)/Branch if Higher or Same (BHS) displays as BCC
- Branch if Carry Set (BCS)/Branch if Lower (BLO) displays as BCS

If the assembler tries to assemble at an address that is not in RAM or EEPROM, an invalid address message "rom-xxxx" is displayed on the terminal CRT (xxxx = invalid address).

Assembler/disassembler subcommands are as follows. If the assembler detects an error in the new source line, the assembler will output an error message and then reopen the same address location.

| | |
|---|---|
| / | Assemble the current line and then disassemble the same address location. |
| ^ | Assemble the current line and then disassemble the previous sequential address location. |
| **<CR>** | Assemble the current line and then disassemble the next opcode address. |
| (CTRL)J | Assemble the current line. If there isn't a new line to assemble, then disassemble the next sequential address location. Otherwise, disassemble the next opcode address. |
| (CTRL)A | Exit the assembler mode of operation. |

## 4.8 DOWNLOADING PROCEDURES

This portion of text describes the EVBU downloading procedures. Downloading operations allow Motorola's S-record files to be transferred from a personal computer to the EVBU or to be verified against data in EVBU memory. S-record files are made up of data and checksum values in a special format which facilitates downloading. Appendix A describes the S-record format in detail.

In a normally configured EVBU, all data transfers including monitor communications and download data utilize the terminal I/O port connector P1. Since there are no separate host communication ports available on the EVBU, only the LOAD <T> and VERIFY <T> variations of the load and verify commands are applicable.

The setup for downloading includes a personal computer (e.g., IBM-PC or Macintosh), a serial interface cable to connect the personal computer to the EVBU connector P2, and the EVBU with an applicable power source. A software terminal emulator program is also required. Some typical terminal emulator programs for the IBM-PC include PROCOMM and KERMIT. Typical terminal emulator programs for the Macintosh include MacTerminal and Red Ryder.

S-record programs for downloading are created by assembling programs on the personal computer (PC). The steps needed to develop a program are described briefly as follows:

1. Assembly language program is entered into a text file on the PC. A text editor is used to create this text file which is called a source program.
2. An assembler program operating on the PC is used to translate the source program into an S-record object file and/or listing file. Buf32.asm file on the EVBU diskette is an example of a large listing.
3. After the creation of the S-record files, the files are downloaded to the EVBU as shown in the following step-by-step procedures.

### 4.8.3 IBM-PC (with KERMIT) to EVBU

To perform the IBM-PC to EVBU downloading procedure with KERMIT, perform/observe the following:

| EXAMPLE | DESCRIPTION |
|---|---|
| `C>`**`KERMIT<CR>`**<br>`IBM-PC Kermit-MS VX.XX`<br>`Type ? for help` | IBM-PC prompt. Enter KERMIT program. |
| `Kermit-MS>`**`SET BAUD 9600<CR>`** | Set IBM-PC baud rate. |
| `Kermit-MS>`**`CONNECT<CR>`** | Connect IBM-PC to EVBU. |
| `[Connecting to host, type Control-] C to return to PC]` | |
| **`<CR>`**<br>`>`**`LOAD T<CR>`** | EVBU download command (via terminal port) entered. |
| **`(CTRL)]C`**<br>`Kermit-MS>`**`PUSH<CR>`** | |
| `The IBM Personal Computer DOS`<br>`Version X.XX (C)Copyright IBM Corp 1981, 1982, 1983` | |
| `C>`**`TYPE (File Name) > COM1<CR>`** | Motorola S-record file name. |
| `C>`**`EXIT<CR>`** | S-record downloading completed. |
| `Kermit-MS>`**`CONNECT<CR>`** | Return to EVBU monitor program. |
| `>`**`(CTRL)]C`**<br>`Kermit-MS>`**`EXIT<CR>`** | Exit KERMIT program. |

### 4.8.4 IBM-PC (with PROCOMM) to EVBU

To perform the IBM-PC to EVBU downloading procedure with PROCOMM, perform/observe the following:

1. Start the PROCOMM.EXE program.
2. Setup PROCOMM to match EVBU baud rate and protocol (type (Alt)P, then the number 5) as follows:

   9600 baud, no parity, 8-bits, 1-stop bit, full duplex

3. Setup ASCII transfer parameters (type (Alt)S, then the number 6) as follows:

   Echo Local – Yes

   Expand Blank Lines – Yes

   Pace Character – 0

   Character pacing – 25 (1/1000 second)

   Line Pacing – 10

   CR Translation – None

   LF Translation – None

   Save above settings to disk for future use.

4. Apply power to EVBU.
5. Press IBM-PC keyboard carriage return <CR> key to display applicable EVBU monitor prompt.
6. Enter EVBU monitor download command as follows:

   ```
   >LOAD T<CR>
   ```

7. Instruct PROCOMM to send the S-record file by pressing the Pg Up key on the PC, then follow PROCOMM instructions on the display screen to select the S-record file. (Use the ASCII transfer protocol.)

   Motorola S-record file is now transferred to the EVBU.

   Upon completion of the S-record transfer, the following message is displayed:

   ```
   done
   >
   ```

# APPENDIX A
# S-RECORD INFORMATION

## A-1. INTRODUCTION

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.

## A-2 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 5 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 load-able data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record). |
| Checksum | 2 | The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields. |

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

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

## A-3 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 EVBU 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 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. 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-S8 | Not applicable to EVBU. |
| 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. 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. Normally, only one header record is used, although it is possible for multiple header records to occur.

## A-4 S-RECORD CREATION

S-record format programs may be produced 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.

## A-5 S-RECORD EXAMPLE

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

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The above module 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 6), indicating six character pairs (or ASCII bytes) follow. |
| 00<br>00 | Four-character 2-byte address field, zeroes. |
| 48<br>44<br>52 | ASCII H, D, and R – "HDR". |
| 1B | Checksum of S0 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. |
| 00<br>00 | 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 52, respectively. The fourth S1 code/data record contains 07 character pairs 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<br>00 | Four-character 2-byte address field, zeroes. |
| 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 | |
| 5 | 3 | 3 | 1 | 3 | 1 | 3 | 3 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 2 | 3 | 8 | 3 | 5 | 4 | 6 | ••• | 3 | 2 | 4 | 1 |
| 0101 | 0011 | 0011 | 0001 | 0011 | 0001 | 0011 | 0011 | 0011 | 0000 | 0011 | 0000 | 0011 | 0000 | 0011 | 0000 | 0011 | 0010 | 0011 | 1000 | 0011 | 0101 | 0100 | 0110 | ••• | 0011 | 0010 | 0100 | 0001 |