BUFFALO MONITOR

for HC11 Development Boards

START-UP

Applying power to the board 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.

Both COM ports will display the following monitor prompt:

```
BUFFALO 3.4AX - CMD11 User Fast Friendly Aid to Logical Operation
```

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

a. Disconnect power source.
b. Check all cabling and power connections.
c. Check hardware options and preparation procedures.
d. Check all components for proper PCB placement.
e. Check PC baud rate and terminal software setup for 9600 baud, 8 bits, 1 stop, no parity.

PROGRAM DESCRIPTION

The monitor program is contained in EPROM (external to the MCU) at locations $EOOO-$FFFF. It is a useful tool to help debug software written for the MCU. It uses a command driven interface which allows users type commands at a prompt. The monitor executes entered commands and the prompt reappears upon completion. However, if a command is entered which causes execution of user target code (i.e., GO) then control may or may not return to the monitor.

The monitor program uses the MCU internal RAM located at $0036-$00FF. The control registers are located at $1000-$103F. The monitor program also uses Output Compare 5 (OC5) and XIRQ for the TRACE instruction, therefore OC5 and XIRQ should not be used in user routines being traced. Jumper JP13 should be installed to enable this operation.

It should be noted (when designing code) that the BUFFALO uses the MCU on-chip RAM locations $0036-$00FF leaving only 54 bytes for the user (i.e., $0000-$0035). However, external ram is normally available from address $100 to $7FFF depending on the size of ram installed.

OPERATING PROCEDURES

The BUFFALO monitor program is the resident firmware on the development board, 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 MONITOR operation while the user types a command line. Command processing begins only after the command line has been terminated by depressing the keyboard carriage return (Return) key.

COMMAND LINE FORMAT

The command line format is as follows:

```
     ><command> [<parameters>](RETURN)
```

where:

>	MONITOR monitor prompt.
<command>	Command mnemonic(single letter for most commands).
<parameters>	Expression or address.
(RETURN)	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 syntactical variable
[ ]	Enclose optional fields
[ ]...	Enclose optional fields repeated

These characters are not entered by the used, 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.

TABLE 1. Monitor Program Commands

ASM [<address>]	Assembler/disassembler
BF <addr1> <addr2> <data>	Block fill memory with data
BR [-] [<address>]...	Breakpoint set
BULK	Bulk erase EEPROM
BULKALL	Bulk erase EEPROM + CONFIG register
CALL [<address>]	Execute subroutine
G [<address>]	Execute program
HELP	Display monitor commands
LOAD <host download command>	Download (S-records*) via host port
LOAD <T>	Download (S-records*) via terminal port
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
RM[p,y,x,a,b,c,s,]	Register modify
T [<n>]	Trace $1-$FF instructions
TM	Enter transparent mode
VERIFY <host download command>	Compare memory to download data via host port
VERIFY <T>	Compare memory to download data via terminal

Table 2 lists the compatible commands that are applicable to all revisions of the BUFFALO monitor program. In most cases the initial single letter of the command mnemonic or a specific symbol (shown below) can be used. A minimum number of characters must be entered to at least guarantee uniqueness from other commands (i.e., MD = MOVE, ME = MEMORY).

TABLE 2, Monitor Program Compatible Commands

ASM [<ADDRESS>]	ASSEM
BF <addr1> <addr2> <data>	FILL
BR [-] [<address>]...	BREAK
BULK	BULK
BULKALL	BULKA
CALL [<address>]	CALL
G [<address>]	GO
HELP	HELP, ?
LOAD <host download command>	LOAD
LOAD <T>	LOAD
MD [<addr1> [<addr2>]]	DUMP
MM [<address>]	MEMORY, /
MOVE <addr1> <addr2> [<dest>]	MOVE
P	PROCEED
RM [P,X,Y,A,B,C,S,]	REGISTER
T [<n>]	TRACE
TM	HOST
VERIFY <host download command>	VERIFY
VERIFY <T>	VERIFY
ASM Assembler/Disassembler	ASM

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:

(1) Underscore entries are user-entered on the terminal keyboard.
(2) Command line input is entered when the keyboard (RETURN) key is depressed.

Typical example of this explanation is as follows:

```
>MD F000 F100
```

ASM

ASM [<address>]

where: <address> is the starting address for the assembler operation. Assembler operation defaults to internal RAM if no address if 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 and instruction, the machine code is disassemble 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 designed 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 00,X). (c.) Direct and extended addressing is specified by the length for 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. Each of the instruction pairs Arithmetic Shift Left (ASL)/Logical Shift Left (LSL) have the same opcode, so disassembly always displays the ASL mnemonic. If the assembler tries to assemble at an address that is not in RAM, an invalid address message "rom-xxx" 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.

(RETURN) 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                    DESCRIPTION

>ASM 0200                   Immediate mode addressing, requires, # before
0200 STOP $FFFF             operand.
     >LDAA #55
      86 55
0202 STOP $FFFF             Direct mode addressing.
     >STAA C0
      97 C0
0204 STOP $FFFF             Index mode, if offset = 0 (,X) will not be
     >LDS 0,X                     accepted.
     AE 00
0206 STOP $FFFF             Branch out of range message.
     >BRA 0230

Branch out of range

0206 STOP $FFFF             Branch offsets calculated automatically,
     >BRA 0230              address required as conditional branch operand.
     20 28
0208 STOP $FFFF
     >(CTRL)A                    Assembler operation terminated.

>
```

Refer to the end of this chapter for additional operational information pertaining to the use of the assembler/disassembler.

BF - Block Fill

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 allows the user to repeat a specific byte throughout a determined user memory range. If an invalid address is specified, an invalid address is specified, an invalid address message "rom-xxxx" is displayed on the terminal CRT (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.

Breakpoint Set - BR

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 location before returning control back to the monitor program.

SWI opcode cannot be executed or breakpointed in user code because the monitor program uses the SWI vector. Only RAM locations can be breakpointed. Branch on self instructions cannot 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>.

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
0203 0205 0207 0209                breakpoint being set.
>

>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 can be
                                        breakpointed.

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 ERASE EEPROM - BULK

BULK

The bulk command allows the user to erase all MCU EEprom locations ($B600-$B7FF). A delay loop is built in such that the erase time is about 5ms when running at 2 MHz E clock. This command is only applicable for A38P and A95J mask sets, and all future mask sets.

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 all MCU EEPROM locations ($B600-$B7FF).
>	Prompt indicates erase sequence completed.

BULK ERASE EEPROM AND CONFIG REGISTER - BULKALL

BULKALL

The bulkall command allows the user to erase all MCU EEPROM locations ($B600-$B7FF) including the configuration (CONFIG) register location ($103F). A delay loop is built in such that the erase time is about 5 ms when running at 2 MHz E Clock.

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 one or two EEPROM locations/configuration register location using the MM or MD command.

CAUTION

Caution should be observed when erasing MCU EEPROM locations. MONITOR MCU configuration (CONFIG) register ROMON bit is cleared to disable MCU internal ROM, thereby allowing external EPROM containing the BUFFALO program to control MONITOR operations.

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

CALL

```
     CALL [<address>]

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 MONITOR reset switch S1 is
activated (pressed).

     EXAMPLE PROGRAM CALL, G, AND P COMMAND EXAMPLES

     >ASM 0200                      0206 STX $FFFF
                                   >NOP
     0200 STX $FFF                   01
          >LDAA #44                 0207 STX $FFFF
          86 44                    >NOP
     0202 STX $FFFF                  01
          >STAA 07FC                0208 STX $FFFF
          B7 07 FC                 >RTS
     0205 STX $FFFF                  39
          >NOP                      0209 STX  $FFFF
           01                      >(CTRL)A

     EXAMPLE                                DESCRIPTION

>CALL 0200                             Execute program subroutine.

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

GO

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

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 MONITOR reset switch S1 is activated (pressed).

NOTE

Refer to example program shown and insert breakpoints at locations $0205 and $0207 for the following G command example.

```
     EXAMPLE                        DESCRIPTION

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

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

HELP

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

LOAD

```
     LOAD      <Host download command>

     LOAD      <T>
```

where: <host download command> download S-records to MONITOR via host port.

<T> download S-records to MONITOR via terminal port.

The LOAD command moves (downloads) object data in S-record format from an external host computer to MONITOR. As the MONITOR 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 CRT (xxxx = invalid address).

```
EXAMPLES                          DESCRIPTION

>LOAD T                           MONITOR download command (via terminal port)
```

You can now us your PC Terminal program to send a file to the board.

MEMORY DISPLAY - MD

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

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 0230 0220

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

>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  ................
```

MEMORY MODIFY - MM

MM [<address>]

CAUTION - Caution should be observed when modifying EEPROM locations. MONITOR MCU CONFIG register ROMON bit is cleared to disable MCU internal ROM.

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 submodes 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 55A	Change data at 0700 and backup one location.
06FF FF AA(RETURN)	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 <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 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 created in user RAM using the assemble, debugged using the monitor, and then programmed into EEPROM with the MOVE command.

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

CAUTION

Caution should be observed when moving data into EEPROM locations. MONITOR MCU CONFIG register ROMON bit is cleared to disable MCU internal ROM.

```
     Example                              Description

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

PROCEED

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 show for the following P command example. Breakpoints have been inserted at locations $0205 and $0207

EXAMPLE	DESCRIPTION
>G 0200	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

REGISTER MODIFY - RM

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

The RM command is used to modify the MCU program counter (P), Y index (Y), X index (X), A acculmulator (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.
```

TRACE - T

```
       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 setting the OC5 interrupt to time out after the first cycle of the first opcode fetched.

(Install JP13 on CMD11A Board to connect OC5 to XIRQ for Trace to operate)

```
     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
>
```

TRANSPARENT MODE - TM

TM

The TM connects the MONITOR host port to the terminal port, which allows direct communication between the terminal and the host computer. All I/O between the ports are ignored by the MONITOR 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.

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

```
     VERIFY <T>

where: <host download command>      compare memory to host port download
                                 data.

       <T>                          compare memory to terminal port
                                 download data.
```

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

```
          EXAMPLES                      DESCRIPTION

>VERIFY cat trial.out                     Enter verify command.
                                       cat trial.out
done                                   Verification completed.
>

>VERIFY cat trial.out                     Enter verify command.
                                    cat trial.out
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.

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 nmemocs. 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.0 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.) Index 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 00,X). (c.) Direct and extending addressing is specified by the length of the address operand (1 or 2 digits specifies direct, 3 0r 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.

Assembler/disassembler subcommands are as follows. If the assembler directs 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.
(RETURN)	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.

When a new source line is assembles, the assembler overwrites what was previously in memory. If now new source line is submitted, or if there is an error in the source line, then the contents of memory remain unchanged. Each of the instruction pairs Arithmetic Shift Left (ASL)/Logical Shift Left (LSL) have the same opcode, so disassembly always displays the ASL mnemonic. If the assembler tries to assemble at an address that is not in RAM, an invalid address message "rom-xxx" is

displayed on the terminal CRT (xxxx = invalid address).

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 MONITOR (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 TM and LOAD commands. The TM (Transport Mode) command connects the MONITOR host port to the terminal port, which allows direct communication between the terminal and host computer. All I/O between the ports are ignored by the MONITOR until the exit command (CTRL)A is entered from the terminal. The LOAD command moves data information in S-record format from an external host computer to the MONITOR 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 MONITOR baud rates are identical.

NOTE
IBM PC to MONITOR interconnection is accomplished by a single RS-232C cable assembly. This cable is connected to the MONITOR terminal I/O port connector COM1 for downloading operations.

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

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

```
Kermit-MS>CONNECT                   Return to monitor BUFFALO program.

>(CTRL)]C
Kermit-MS>EXIT                      Exit Kermit program.
```

INTERRUPT VECTORS

Interrupt vectors residing in MCU internal Rom are accessible as follows. Each vector is assigned a three byte field residing in MONITOR memory map locations $0000-$0100. 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. The following Table lists the interrupt vectors and associated three byte field.

Interrupt Vector Jump Table

INTERRUPT VECTOR	FIELD
Serial communications Interface (SCI)	$00C4-$00C6
Serial Peripheral Interface (SPI)	$00C7-$00C9
Pulse Accumulator Input Edge	$00CA-$00CC
Pulse Accumulator Overflow	$00CD-$00CF
Timer Overflow	$00DO-$00D2
Timer Output Compare 5	$00D3-$00D5
Timer Output Compare 4	$00D6-$00D8
Timer Output Compare 3	$00D9-$00DB
Timer Output Compare 2	$00DC-$00DE
Timer Output Compare 1	$00DF-$00E1
Timer Input Capture 3	$00E2-$00E4
Timer Input Capture 2	$00E5-$00E7
Timer Input Capture 1	$00E8-$00EA
Real Time Interrupt	$00EB-$00ED
IRQ	$00EE-$00FO
XIRQ	$00F1-$00F3
Software Interrupt (SWI)	$00F4-$00F6
Illegal Opcode	$00F7-$00F9
Computer Operating Properly (COP)	$00FA-$00FC
Clock Monitor	$00FD-$00FF

To use vectors specified in the table, the user must insert a jump extended opcode in the byte field of the vector required. For example, for the IRQ vector, the following is performed:

a. Place $7E (JMP) at location $00EE.
b. Place IRQ service routine address at locations $00EF and $00F0.

```
$00EE     7E  80  00   JMP IRQ SERVICE
```

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 are sent to the terminal port. Redirection of the I/O port is achieved by placing the specified value (O=SCI, 1=ACIA)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 whitespace (space, comma, tab).
DCHEK	Test character in accumulator A and return with Z bit set if character is delimiter (carriage return or whitespace).
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 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.
OUTCCRLF	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 na 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 ($04)
$FFCA	JMP	OUTSTRGO	Same as OUTSTRG except leading carriage return and line fees is skipped
$FFCD	JMP	INCHAR	Input ASCII character and echo back

S-RECORD INFORMATION

The Motorola S-record format was devised for the purpose of encoding programs 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 loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record.
Checksum	2	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 MONITOR 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-S8	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 ma be 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.

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


     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.

     00   Four-character 2-byte address field; hexadecimal address 0000,
     00             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