```
GETHEX      CMP.B    #$30,D0        IS HEX NO. < 0?
            BLT.S    ERROR          NOT A HEX NO.
            CMP.B    #$39,D0        IS HEX NO. > 9?
            BGT.S    GTHX2
GTHX1       AND.L    #$F,D0         SAVE ONLY LOWER 4 BITS
EXIT        BRA      *              END OF ROUTINE
GTHX2       CMP.B    #$41,D0        IS HEX NO. < 'A'?
            BLT.S    ERROR          NOT A HEX NO.
            CMP.B    #$46,D0        IS HEX NO. > 'F'?
            BGT.S    ERROR          NOT A HEX NO.
            SUB.B    #7,D0          MAKE IT SMALLER -- A=10
            BRA      GTHX1
ERROR       MOVE.L   #$FF,D0        ERROR CODE
            JMP      EXIT
```

NOTE: Converts ASCII digit in lowest 8-bit of register D0 into hex value. Returns equivalent 0-F or FF on error in D0.

FIGURE 4-1. Example Program to Convert ASCII Digit to Hexadecimal Value

For clarity, Figure 4-1 contains comments and labels. The program as it appears after entry into the Educational Computer is shown later. Also, Figure 4-2 shows the ASCII character set for better understanding of the program.

b7 b6 b5 / Bits						0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	Row	Hex / Column	0	1	2	3	4	5	6	7
0	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	C	FF	FS	,	<	L	\	l	|
1	1	0	1	13	D	CR	GS	-	=	M	]	m	}
1	1	1	0	14	E	SO	RS	.	>	N	^	n	~
1	1	1	1	15	F	SI	US	/	?	O	_	o	DEL

FIGURE 4-2. ASCII Character Set

4.3.1 Invoking the Assembler/Disassembler

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

```
MM <address> ;DI
```

where `CR` sequences to next instruction
`.CR` exits command

and

```
MD[<port number>] <address> [count];DI
```

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

The disassembled line can be an MC68000 instruction or a DC.W directive. If the disassembler recognizes a valid form of some instruction, the instruction will be returned; if not (random data occurs), the DC.W $XXXX (always hex) is returned. Because the disassembler gives precedence to instructions, a word of data that corresponds to a valid instruction will be returned as the instruction.

For the given example, the program will be entered starting at location $1000:

```
TUTOR  1.X > MM 1000;DI
001000    1005                    MOVE.B  D5,D0 ?
```

4.3.2 Entering a Source Line

A new source line is entered immediately following the "?", using the format discussed in paragraph 4.2.1:

```
TUTOR  1.X > MM 1000;DI
001000    1005                    MOVE.B  D5,D0 ?  CMP.B #$30,D0
```

When the carriage return is entered terminating the line, the old source line is erased from the terminal screen, the new line is assembled and displayed, and the next instruction in memory is disassembled and displayed:

```
TUTOR  1.X > MM 1000;DI
001000    0C000030                CMP.B   #$30,D0
001004    FFFF                    DC.W    $FFFF ?
```

NOTE

If a terminal with a printer only (no CRT) is used, such as a TI 700 series device, the printer will overwrite the previous line. Therefore, a clear printout of the new entry will not be made. This also happens if the printer on Port 3 is attached via the PA command. Refer to Appendix B for operation with mechanical terminals.

Another program line can now be entered. Program entry continues in like manner until all lines have been entered. A period is used to exit the MM command.

If an error is encountered during assembly of the new line, the assembler will display the line unassembled with an "X" under the field suspected of causing a problem, or an error message will be displayed. Errors are discussed in paragraph 4.3.5.

4.3.3 Program Entry/Branch and Jump Addresses

Figure 4-3 shows the example program as it is inputted to the educational computer assembler. Notice that the comments and labels used in Figure 4-1 are not allowed; absolute addresses must be used for BRA and JMP instructions.

```
CMP.B    #$30,D0
BLT      *
CMP.B    #$39,D0
BGT      *
AND.L    #$F,D0
BRA      *
CMP.B    #$41,D0
BLT      *
BGT      *
SUB.B    #7,D0
BRA      $100C
MOVE.L   #$FF,D0
JMP      $1012
```

a) First entry

```
CMP.B    #$30,D0
BLT      $1022
CMP.B    #$39,D0
BGT      $1014
AND.L    #$F,D0
BRA      *
CMP.B    #$41,D0
BLT      $1022
BGT      $1022
SUB.B    #7,D0
BRA      $100C
MOVE.L   #$FF,D0
JMP      $1012
```

b) With correct branch addresses

FIGURE 4-3. Example Program as Entered into Educational Computer

4.3.3.1 Entering Absolute Addresses. The absolute addresses are probably not known as the program is being entered. For example, when the second line is entered (BLT.S ERROR in Figure 4-1), the user does not know that the branch address (ERROR MOVE.B #$FF,D0) will be $1022. However, the user can instead enter an "*" for branch to self. After the correct address ($1022) is discovered, the second line can be re-entered using the correct value. This technique can be used for forward branches and jumps. It is not required for backward branches and jumps, such as the last line of the example, because the required address is already known. If the absolute address is not within the range of a short address, a long address must be specified by appending .L to the mnemonic (BGT.L *).

4.3.3.2 Desired Instruction Form. Care must be taken when entering source lines to ensure that the desired instruction form is entered. If the quick form of the instruction is wanted, it must be specified. For example:

```
005780   203C00000003   MOVE.L   #3,D0     Assembles to the 6-byte instruction.
```

whereas

```
005780   7003           MOVEQ.L  #3,D0     Assembles to the 2-byte instruction.
```

If the PC-relative addressing mode is desired, it must be specified. For example:

```
001000   41F803F0    LEA   $3F0,A0      Assembles $3F0 as an absolute address.
```

whereas

```
001000   41FAF3EE    LEA   $3F0(PC),A0  Assembles $3F0 as a PC-relative address.
```

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

```
007000     6022           BRA     *+$24

007000     6000FFFE       BRA.L   *

007000     60FE           BRA     *
```

4.3.4 Assembler Output/Program Listings

A listing of the program is obtained using the Memory Display (MD) command with the ;DI option. The MD command requires both the starting address and the byte count to be entered in the command line. When the ;DI option is invoked, the number of instructions disassembled and displayed will be equal to the number of instructions whose op code (first word of any instruction) is contained within the byte count. The DC.W directive will also be displayed for any words of data contained within the byte count.

Two techniques can be used to obtain a hard copy of the program using the MD command:

a. The Printer Attach (PA) command is first used to activate the Port 3 printer. A Memory Display (MD) to the terminal will then cause a listing on the terminal and on the printer.

b. An MD3 (Memory Display to Port 3) command using the ;DI option will cause a listing on the printer only.

Figure 4-4 shows a listing of the example program. Note in the example that $2D bytes are specified in the MD command. Only $2C bytes are required for the program, and the additional single byte does not constitute a valid DC.W or op code; therefore, the last byte is not displayed.

Note also that the listing does not correspond exactly to that of Figure 4-3. As discussed in paragraph 4.2.1.3, the disassembler displays in hexadecimal any number it interprets as an address; all other numbers are displayed in decimal.

```
TUTOR  1.X > MD 1000 2D;DI
001000    0C000030              CMP.B     #48,D0
001004    6D1C                  BLT.S     $001022
001006    0C000039              CMP.B     #57,D0
00100A    6E08                  BGT.S     $001014
00100C    02800000000F          AND.L     #15,D0
001012    60FE                  BRA.S     $001012
001014    0C000041              CMP.B     #65,D0
001018    6D08                  BLT.S     $001022
00101A    6E06                  BGT.S     $001022
00101C    04000007              SUB.B     #7,D0
001020    60EA                  BRA.S     $00100C
001022    203C000000FF          MOVE.L    #255,D0
001028    4EF81012              JMP       $1012
TUTOR  1.X >
```

FIGURE 4-4. Example Program Listing

4.3.5 Error Conditions and Messages

There are five different conditions that can result in error messages while using the assembler/disassembler. The response to the error condition can be to abort the command (and thus the assembler), or to cause the assembler to ask for a corrected input line. The error conditions are discussed in the following paragraphs and include bus and address trap errors, improper characters, numbers which are too large, and assembly errors.

4.3.5.1 Trap Errors. Two types of trap errors can be caused. One form, the bus trap error, may be encountered if a location is accessed at which there is no memory. Included in this error type are write cycles to ROM. The second form is an address trap error. Instructions must always begin on an even address; if not, an address trap error will result. Figure 4-5 shows examples of these conditions.

```
TUTOR  1.X > MM E000;DI

4CD5 0000E000 4CD4
BUS  TRAP ERROR
PC=009192 SR=2704=.S7..Z.. US=EFAD7EBF SS=000007B4          No memory
D0=0000E044 D1=01964D4D D2=FFF24D4D D3=00000000             at address
D4=0000B432 D5=00000000 D6=00000000 D7=00000FFD             $E000
A0=000080C2 A1=00008344 A2=00000454 A3=0000054E
A4=0000E000 A5=0000053A A6=0000053A A7=000007B4
--------------------009192    27AC1CFC003F          MOVE.L  7420(A4),63(A3,D0.W)
TUTOR  1.X > MM A000;DI
00A000   6502          BCS.S    $A004 ?     BEQ.S     $A000

1285 0000A000 1280
BUS  TRAP ERROR                                             ROM at address
PC=0091EE SR=2700=.S7..Z.. US=EFAD7EBF SS=000007B4          $A000; cannot
D0=67FE0067 D1=00000001 D2=650202FF D3=00000000             be written to
D4=FFFFFFFE D5=FFFFFFFE D6=00000002 D7=00000000
A0=000007F5 A1=0000A000 A2=000007B1 A3=00000817
A4=0000A000 A5=0000081A A6=0000081A A7=000007B4
--------------------0091EE    B400                  CMP.B   D0,D2
TUTOR  1.X > MM 3001;DI

4CD5 00003001 4CD4
ADDR TRAP ERROR                                             Instructions
PC=009192 SR=2704=.S7..Z.. US=EFAD7EBF SS=000007B4          must begin at
D0=00003044 D1=01964D4D D2=FFF24D4D D3=00000000             an even address
D4=FFFFB432 D5=00000000 D6=00000000 D7=00000FFD
A0=000080C2 A1=00008344 A2=00000454 A3=0000054E
A4=00003000 A5=0000053A A6=0000053A A7=000007B4
--------------------009192    27AC1CFC003F          MOVE.L  7420(A4),63(A3,D0.W)
```

FIGURE 4-5. Examples of Trap Errors

Also note that BUS and ADDRESS trap errors also cause display of the exception status from the stack, in hexadecimal characters:

XXXX AAAAAAAA IIII

where:

XXXX Are miscellaneous status bits:

0-2	Function code
3	Instruction/Not (0 = instruction, 1 = not)
4	Read/Write (0 = read, 1 = write)
5-15	Not defined

AAAAAAAA Is access address.

IIII Is instruction register (first word of instruction being processed).

For details on this display, refer to the bus error and address error descriptions in the MC68000 User's Manual, MC68000UM.

4.3.5.2 Improper Character. If a character appears in the operand field that does not belong to the class of characters specified or expected, an "X" will be printed beneath the character string suspected of containing the improper character, followed by a "?" to prompt re-entry of the line. For example, if a % (percent sign) is used to specify the binary class of characters, only the digits 0 and 1 will be accepted.

```
TUTOR  1.X > MM 6000;DI                                  S is not a decimal digit

006000    FFFF      DC.W     $FFFF ?   MOVE.W  #S',D0
006000              MOVE.W   #S',D0
                               X?

TUTOR  1.X > MM 6000;DI                                  9 is not an octal digit

006000    FFFF      DC.W     $FFFF ?   ADDA.L  #@974,A6
006000              ADDA.L   #@974,A6
                               X?


TUTOR  1.X > MM 6000;DI                                  P is not a decimal digit

006000    FFFF      DC.W     $FFFF ?   JMP  $4000+PC
006000              JMP      $4000+PC
                                    X?
```

FIGURE 4-6. Examples of Improper Characters

4.3.5.3 Number Too Large. Another error type involves numbers which are too large for the MC68000 to handle. Again, an "X" is printed under the number suspected of containing the error, followed by a "?". Figure 4-7 gives an example.

```
TUTOR  1.X > MM 4000;DI                          Value is larger than 32 bits

004000    FFFF       DC.W     $FFFF ? LEA.L  $937402110,A7
004000               LEA.L    $937402110,A7
                                X?
```

FIGURE 4-7. Example of a number which is too large

4.3.5.4 Assembly Errors. An assembly error can occur due to an invalid op code, an illegal addressing mode for a particular instruction, the format may be in error (leading space omitted as an example), or the source line incorrect in some other way. When the entry as written is not a valid MC68000 instruction, the assembler echoes the source line up to and including the field in which the error probably occurred. It also prints an "X" under the field suspected of containing an error, followed by a "?" to prompt re-entry of the line.

The entire line must be re-entered in its correct form. If the error has not been corrected or another is encountered, the error indicator will be returned. After all errors have been corrected and the source line represents a valid MC68000 instruction, the line will be assembled. The memory address, machine code, and source code will be displayed and the next line will be disassembled. A period (.) is used to exit the command. Examples of typical errors are shown in Figure 4-8.

Example 1 Invalid Op Code

```
006700  FFFF      DC.W      $FFFF ?   BEQU.S    $6754
                  BEQU.S
                  X?  BEQ.S $6754
006700  6752      BEQ.S     $6754
```

Example 2 Missing Leading Space

```
001100  FFFF      DC.W      $FFFF     ?OR.B     D5,(A6)

                  X?  OR.B  D5,(A6)
001100  8B16      OR.B      D5,(A6)
```

Example 3 Unrecognizable Op Code

```
005300  FFFF      DC.W      $FFFF ?   MULSW     52,D3
                  MULSW
                  X?  MULS.W     52,D3
005300  C7F80034  MULS.W     52,D3
```

Example 4 Invalid Size Extension

```
007200  FFFF      DC.W      $FFFF ?   MOVEQ.B   #2,D1
                  MOVEQ.B   #2,D1
                                X?    MOVEQ.L   #2,D1
007200  7202      MOVEQ.L   #2,D1
```

Example 5 Invalid Addressing Mode

```
001500  FFFF      DC.W      $FFFF  ?        ADDQ.B    #7,A0
                  ADDQ.B    #7,A0
                                X?    ADDQ.B    #7,(A0)
001500  5E10      ADDQ.B    #7,(A0)
```

Examples 6 and 7 Branch Address Too Large

```
004900  FFFF      DC.W      $FFFF  ?        BRA  $10000
                  BRA       $10000
                                  X?  BRA     $8000
004900  600036FE  BRA       $8000


004800  FFFF      DC.W      $FFFF  ?        BRA.S     $7000
                  BRA.S     $7000
                                X?    BRA.S     $4902
                  BRA.S     $4902
                                X?    BRA.S     $4860
004800  605E      BRA.S     $4860
```

FIGURE 4-8. Examples of Assembly Errors

4.4 TESTING/EXECUTING PROGRAMS

After program entry, the next step is to execute and debug the program. With the facilities provided by TUTOR, the user can run the program with trace capabilities. The following paragraphs describe techniques to help this process and, as before, the example program is used to illustrate.

4.4.1 System Initialization

The first step in running and testing a program is initialization of the processor registers and any peripheral devices, as required. For simple programs involving only the processor, this initialization concerns only the MC68000 registers:

a. Bit 13 of the status register must be initialized to select either the user state or the supervisory state for the MC68000. These operating modes are discussed in the MC68000 User's Manual.

b. The stack pointer(s) (User's Stack Pointer and/or Supervisor Stack Pointer) must be set to point to a valid RAM address. If a stack pointer is left pointing to non-existent memory or to ROM, a bus trap error will occur when the stack is used. Each stack pointer is a 32-bit register, and both must be initialized if both operational modes will be used.

   When writing programs on the MC68000 Educational Computer, the user must not position either stack pointer within the RAM allocated to TUTOR -- that is, RAM addresses below $900 should not be used. Also, since the stack grows from high memory to low memory (i.e., the stack pointer value decreases as information is placed on the stack), enough room should be left between the initial stack pointer value and $900. The size of the stack area should be large enough to accommodate the maximum number of words that will ever be stacked at one time. A final caution is to not overlap stack areas. NOTE: The Trace command requires a supervisor stack area of at least six bytes.

c. Address registers (A0-A6) should be initialized as required by the program.

d. Data registers (D0-D7) should be initialized as required by the program.

e. The program counter (PC) should be set to the beginning address of the program.

For simple instructional programs, register initialization through the TUTOR commands is acceptable. For more comprehensive programs, however, initialization of the processor and all other resources should be an integral part of the target program. Programs should be written with the concept that they will ultimately have to run in a stand-alone manner and they must control all resources. TUTOR would not be a part of the target program.

Figure 4-9 shows the initialization procedure for the ASCII to hex digit conversion routine. The registers are displayed with the DF command to check their contents. Bit 13 of the status register indicates the supervisory mode is operational. Because the program can run in either mode, it is not necessary to change modes. The program counter (PC) is set to $1000, which is the beginning address of the program. The supervisory stack pointer is set to $0F00 (the user's stack pointer is not used and thus is not initialized). Finally, $31 (ASCII value for 1) is entered into data register D0. The program expects to find an ASCII character in the lowest byte of D0.

```
TUTOR  1.X > DF
PC=00000000 SR=2704=.S7..Z.. US=00002000 SS=000007BC
D0=0000000A D1=00000000 D2=00000000 D3=00000000
D4=B0000018 D5=0000003F D6=00000000 D7=00000000
A0=00010040 A1=00000638 A2=00001000 A3=00000542
A4=00000544 A5=0000053A A6=0000053A A7=000007BC
--------------------000000    0000                  DC.W    $0000

TUTOR  1.X > .PC 1000

TUTOR  1.X > .SS 0F00

TUTOR  1.X > .D0 31

TUTOR  1.X > DF
PC=00001000 SR=2704=.S7..Z.. US=00002000 SS=00000F00
D0=00000031 D1=00000000 D2=00000000 D3=00000000
D4=B0000018 D5=0000003F D6=00000000 D7=00000000
A0=00010040 A1=00000638 A2=00001000 A3=00000542
A4=00000544 A5=0000053A A6=0000053A A7=00000F00
--------------------001000    0C000030              CMP.B   #48,D0

TUTOR  1.X > BR 1012

BREAKPOINTS
001012    001012
```

FIGURE 4-9. Initializing Registers and Setting Breakpoint for Example Program

4.4.2 Setting Breakpoints

If there are no errors in the program, processing should proceed to the termination instruction -- BRA.S $001012 -- at address $1012 (branch always to self). Register D0 should contain value '1' at that time. Normally, the processor would continue to loop at this address, but a breakpoint will be inserted so the program can be halted and the results checked. Figure 4-9 also shows this breakpoint being entered.

Up to eight breakpoints can be used at one time. With complex programs, breakpoints and the trace function are valuable debugging tools.

4.4.3 Program Execution

The GO (or G) command causes the user program to execute making use of breakpoints. Execution will stop when a breakpoint is encountered, when exception processing is caused by an abnormal program sequence, or when the user intervenes through the ABORT or RESET pushbuttons on the board. The GO command sequence begins by tracing one instruction, setting any breakpoints, and then freerunning. The GT and GD commands can also be used to execute a program with a temporary breakpoint and without breakpoints, respectively.

Figure 4-10 shows execution of the example program (GETHEX). Execution begins at address $1000 where the program counter was initialized. Execution is halted at the breakpoint address $1012. At this point, register D0 contains the value '1', which is the expected number. It appears that the conversion program has performed correctly.

To further test the program, a different ASCII value is entered; D0 is set to $45, which is the representation for 'E' -- a valid hex digit. Upon execution, the program still goes until the breakpoint at $1012; however, D0 now contains value 'FF', which indicates an invalid character. Therefore, there is an error in the program, and it must be further debugged.

```
TUTOR  1.X > G
PHYSICAL ADDRESS=00001000

AT BREAKPOINT
PC=00001012 SR=2700=.S7..... US=00002000 SS=00000F00
D0=00000001 D1=00000000 D2=00000000 D3=00000000
D4=B0000018 D5=0000003F D6=00000000 D7=00000000
A0=00010040 A1=00000638 A2=00001000 A3=00000542
A4=00000544 A5=0000053A A6=0000053A A7=00000F00
--------------------001012    60FE                BRA.S   $001012

TUTOR  1.X > .D0 45

TUTOR  1.X > G 1000
PHYSICAL ADDRESS=00001000

AT BREAKPOINT
PC=00001012 SR=2700=.S7..... US=00002000 SS=00000F00
D0=000000FF D1=00000000 D2=00000000 D3=00000000
D4=B0000018 D5=0000003F D6=00000000 D7=00000000
A0=00010040 A1=00000638 A2=00001000 A3=00000542
A4=00000544 A5=0000053A A6=0000053A A7=00000F00
--------------------001012    60FE                BRA.S   $001012
```

FIGURE 4-10. Execution of Example Program

4.4.4 Trace Mode

The trace mode is another major tool, other than breakpoints, used in debugging software. The basic trace command, TR or T, executes instructions, one at a time, beginning at the location pointed to by the program counter. After execution of each instruction, the processor registers are displayed. The trace command can be used to trace a single instruction or to trace multiple instructions if a <count> number is entered.

As shown in Figure 4-11, the example program will be traced, one instruction at a time, to discover the error(s) in it. Register D0 is again initialized to $45 and the program counter is set at $1000.

```
TUTOR  1.X > .D0 45 cr

TUTOR  1.X > .PC 1000 cr

TUTOR  1.X > DF cr
PC=00001000 SR=2700=.S7..... US=00000F00 SS=00000F00
D0=00000045 D1=0000C000 D2=FFFFC000 D3=00000000
D4=FFFFFFFF D5=00000000 D6=00000001 D7=00000012
A0=00001000 A1=12345678 A2=00000547 A3=0000C000
A4=00001030 A5=0000053A A6=00000541 A7=00000F00
--------------------001000    0C000030              CMP.B   #48,D0

TUTOR  1.X > T cr
PHYSICAL ADDRESS=00001000
PC=00001004 SR=2700=.S7..... US=00000F00 SS=00000F00
D0=00000045 D1=0000C000 D2=FFFFC000 D3=00000000
D4=FFFFFFFF D5=00000000 D6=00000001 D7=00000012
A0=00001000 A1=12345678 A2=00000547 A3=0000C000
A4=00001030 A5=0000053A A6=00000541 A7=00000F00
--------------------001004    6D1C                  BLT.S   $001022


TUTOR  1.X :> cr
PHYSICAL ADDRESS=00001004
PC=00001006 SR=2700=.S7..... US=00000F00 SS=00000F00
D0=00000045 D1=0000C000 D2=FFFFC000 D3=00000000
D4=FFFFFFFF D5=00000000 D6=00000001 D7=00000012
A0=00001000 A1=12345678 A2=00000547 A3=0000C000
A4=00001030 A5=0000053A A6=00000541 A7=00000F00
--------------------001006    0C000039              CMP.B   #57,D0


TUTOR  1.X :> cr
PHYSICAL ADDRESS=00001006
PC=0000100A SR=2700=.S7..... US=00000F00 SS=00000F00
D0=00000045 D1=0000C000 D2=FFFFC000 D3=00000000
D4=FFFFFFFF D5=00000000 D6=00000001 D7=00000012
A0=00001000 A1=12345678 A2=00000547 A3=0000C000
A4=00001030 A5=0000053A A6=00000541 A7=00000F00
--------------------00100A    6E08                  BGT.S   $001014
```

FIGURE 4-11. Trace Sequence for Example Program (sheet 1 of 2)

```
TUTOR  1.X :> cr
PHYSICAL ADDRESS=0000100A
PC=00001014 SR=2700=.S7..... US=00000F00 SS=00000F00
D0=00000045 D1=0000C000 D2=FFFFC000 D3=00000000
D4=FFFFFFFF D5=00000000 D6=00000001 D7=00000012
A0=00001000 A1=12345678 A2=00000547 A3=0000C000
A4=00001030 A5=0000053A A6=00000541 A7=00000F00
--------------------001014    0C000041              CMP.B    #65,D0


TUTOR  1.X :> cr
PHYSICAL ADDRESS=00001014
PC=00001018 SR=2700=.S7..... US=00000F00 SS=00000F00
D0=00000045 D1=0000C000 D2=FFFFC000 D3=00000000
D4=FFFFFFFF D5=00000000 D6=00000001 D7=00000012
A0=00001000 A1=12345678 A2=00000547 A3=0000C000
A4=00001030 A5=0000053A A6=00000541 A7=00000F00
--------------------001018    6D08                  BLT.S    $001022


TUTOR  1.X :> cr
PHYSICAL ADDRESS=00001018
PC=0000101A SR=2700=.S7..... US=00000F00 SS=00000F00
D0=00000045 D1=0000C000 D2=FFFFC000 D3=00000000
D4=FFFFFFFF D5=00000000 D6=00000001 D7=00000012
A0=00001000 A1=12345678 A2=00000547 A3=0000C000
A4=00001030 A5=0000053A A6=00000541 A7=00000F00
--------------------00101A    6E06                  BGT.S    $001022


TUTOR  1.X :> cr
PHYSICAL ADDRESS=0000101A
PC=00001022 SR=2700=.S7..... US=00000F00 SS=00000F00
D0=00000045 D1=0000C000 D2=FFFFC000 D3=00000000
D4=FFFFFFFF D5=00000000 D6=00000001 D7=00000012
A0=00001000 A1=12345678 A2=00000547 A3=0000C000
A4=00001030 A5=0000053A A6=00000541 A7=00000F00
--------------------001022    203C000000FF          MOVE.L   #255,D0
```

FIGURE 4-11. Trace Sequence for Example Program (sheet 2 of 2)

Each time the registers are displayed by the Trace or DF command, a line of source code is also displayed. The source line displayed is pointed to by the current program counter value and is the next instruction to be executed. The first instruction to be executed -- CMP.B #48,D0 -- is pointed to by the initial PC value and is shown following the first group of registers in Figure 4-11. Invoking the trace command causes this instruction to be executed. The resultant register values and the next instruction are then displayed. The PC has been incremented by four to point at the next instruction ($1004). Since D0 has a value greater than $30, the least significant four bits of the status register have been cleared as a result of the compare instruction.

Once the trace mode has been entered, the prompt includes a colon (i.e., TUTOR 1.X :>). While in the trace mode, entering the single character carriage return (CR) will cause one instruction to be traced. In the example, the next instruction -- BLT.S $001022 -- is executed by entering a carriage return following the prompt. Because the status bits from the previous instruction indicate that D0 is not less than $30, the branch to address $001022 is not made. Program tracing continues in this manner until an incorrect condition is found.

After the value in D0 is compared to $41 (CMP.B #65,D0), a branch is made to address $1022 if D0 is less than $41 (BLT.S $001022), or the next instruction (BGT.S $001022) causes a branch to $1022 if D0 is greater than $41. Thus, the only time the branch to the error sequence is not taken is if D0 equals $41. This, then, is incorrect since the sequence excludes hex digits $B through $F. The second branch should be taken only if D0 is greater than $46. A comparison between Figures 4-1 and 4-3 shows that the instruction CMP.B #$46,D0 has been omitted from the program as entered.

4.4.5 Inserting and Deleting Source Lines

Lines are added to or removed from source programs with the Block Move (BM) command. The assembler/disassembler does not support insert line and delete line commands. Insertions are accomplished by moving a portion of the program to a higher or lower memory location, thereby leaving a gap between two sections of the program. The new source lines can be inserted into this gap. A gap can be closed up or lines can be deleted by moving a section of the program in the opposite direction.

In Figure 4-12, the missing source line is inserted into the example program. The last five source lines are moved to a higher memory address, using the Block Move command. (Refer also to Figure 4-4 in paragraph 4.3 for memory addresses.) Since the number of bytes required by the additional line(s) is not known, the gap is made larger than necessary. The additional source line is inserted at address $101A, using the MEX68KECB assembler. Figure 4-12 shows the source lines as they look at this stage. The gap is closed up using BM, and a second source listing is shown.

Notice, however, that several of the branch addresses are now incorrect and would cause a branch to the wrong instruction. Since absolute addresses rather than labels are used, all branch and jump addresses must be checked any time any or all of the lines in a program are moved. If both the branch instruction and the destination of that instruction are within the same program section (i.e., the section that was moved and the section that was not moved), the branch addresses should be correct because they are assembled using relative addressing and no extra bytes have been inserted between the instruction and the destination. If, however, source lines have been inserted between the instruction and the destination, the branch address will be incorrect. The same rule also applies to any type of program counter relative addressing, not just to branch addresses.

Any absolute addresses, including jump addresses, will be incorrect if the destination has been moved to a different address. Otherwise, they should be correct.

```
TUTOR  1.X > BM 101A 102B 1020
PHYSICAL ADDRESS=0000101A 0000102B
PHYSICAL ADDRESS=00001020

TUTOR  1.X > MM 101A;DI
00101A 6E06      BGT.S   $001022?   CMP.B   #$46,D0
00101A 0C000046  CMP.B   #$46,D0
00101E 0007      DC.W      $0007 ?.

TUTOR  1.X > MD 1000 32;DI
001000    0C000030              CMP.B    #48,D0
001004    6D1C                  BLT.S    $001022
001006    0C000039              CMP.B    #57,D0
00100A    6E08                  BGT.S    $001014
00100C    02800000000F          AND.L    #15,D0
001012    60FE                  BRA.S    $001012
001014    0C000041              CMP.B    #65,D0
001018    6D08                  BLT.S    $001022
00101A    0C000046              CMP.B    #70,D0
00101E    0007                  DC.W     $0007
001020    6E06                  BGT.S    $001028
001022    04000007              SUB.B    #7,D0
001026    60EA                  BRA.S    $001012
001028    203C000000FF          MOVE.L   #255,D0
00102E    4EF81012              JMP      $1012

TUTOR  1.X > BM 1020 1031 101E
PHYSICAL ADDRESS=00001020 00001031
PHYSICAL ADDRESS=0000101E

TUTOR  1.X > MD 1000 30;DI
001000    0C000030              CMP.B    #48,D0
001004    6D1C                  BLT.S    $001022
001006    0C000039              CMP.B    #57,D0
00100A    6E08                  BGT.S    $001014
00100C    02800000000F          AND.L    #15,D0
001012    60FE                  BRA.S    $001012
001014    0C000041              CMP.B    #65,D0
001018    6D08                  BLT.S    $001022
00101A    0C000046              CMP.B    #70,D0
00101E    6E06                  BGT.S    $001026
001020    04000007              SUB.B    #7,D0
001024    60EA                  BRA.S    $001010
001026    203C000000FF          MOVE.L   #255,D0
00102C    4EF81012              JMP      $1012
```

FIGURE 4-12. Inserting Missing Source Line into Example Program

In all cases, the destination address should not be moved so that it is out of the range of the address mode being utilized. In the example, the instructions at addresses $1004, $1018, and $1024 must be changed to correct the destination addresses. Figure 4-13 lists the source program after these changes have been made. Testing shows that the ASCII to hex conversion program is now correct.

```
TUTOR  1.X > MD 1000 30;DI
001000    0C000030              CMP.B    #48,D0
001004    6D20                  BLT.S    $001026
001006    0C000039              CMP.B    #57,D0
00100A    6E08                  BGT.S    $001014
00100C    02800000000F          AND.L    #15,D0
001012    60FE                  BT.S     $001012
001014    0C000041              CMP.B    #65,D0
001018    6D0C                  BLT.S    $001026
00101A    0C000046              CMP.B    #70,D0
00101E    6E06                  BGT.S    $001026
001020    04000007              SUB.B    #7,D0
001024    60E6                  BT.S     $00100C
001026    203C000000FF          MOVE.L   #255,D0
00102C    4EF81012              JMP      $1012
```

FIGURE 4-13. Corrected Example Program Listing

4.5 SAVING PROGRAMS

After a program has been created and tested, a permanent copy is desired for both documentation purposes and to avoid re-entering it the next time it is to be executed. There are several methods available for saving programs, depending on the optional hardware that is available. Programs can be saved on tape or can be uploaded to a host processor via Port 2 of the MEX68KECB. Once a program has been sent to the host, it can be saved on the host's mass storage media. Uploading to a host requires a program at the host to input the program S-records from the RS-232 port and save them either in RAM or directly onto the mass storage media. Refer to Appendix A for a description of S-records.

4.5.1 Saving Programs on Tape

Whatever the storage media selected, the DUMP (DU) command is used to convert the program to S-record format and send it to the specified port. For a detailed description of the DUMP command, see paragraph 3.5.8. The Port 4 option must be selected to dump to tape.

Before programs can be stored, a cable must be constructed to interface the tape player to the MEX68KECB (paragraph 2.5.3). Hook up the cable and turn the tape player on. Position the tape in the desired location. It is best to leave a relatively large gap between programs, as this makes it easier to reposition the tape before loading a program. The tape can be positioned anywhere within the gap which precedes the desired program. The tape counter, if one is available, is useful in quickly positioning the tape in the correct spot.

The beginning and ending addresses of the program must be known because these parameters are required by the DUMP command. For the example presented in this chapter, the addresses are:

- Beginning $1000
- Ending $102F

Type in the DUMP command line but do not enter a carriage return yet. The command for the example program is:

```
TUTOR 1.X > DU4 1000 102F
```

Press both the play and record buttons on the tape player. After all the leader has gone by and the motor is up to speed, enter a carriage return at the terminal. When the TUTOR 1.X > prompt is again displayed, indicating that the DUMP command has finished, stop the tape player. After repositioning the tape to the beginning of the records just saved, the VERIFY (VE) command should be used to check the tape against the program. Paragraphs 3.5.26 and 4.5.2 describe using the VERIFY command.

4.5.2 Loading and Verifying Programs from Tape

To prepare for loading a program from tape, position the tape before the beginning of the desired program. Do not start the tape player yet, or a portion of the program may be missed by the MEX68KECB. Enter the LOAD command line shown below, including the carriage return.

```
TUTOR  1.X > LO4  cr
```

The MEX68KECB is now looking for information from Port 4. Start the tape player by depressing the play button only. When the prompt is received, stop the tape player. The program should now be in RAM and can be examined with the MEMORY DISPLAY (MD) and MEMORY MODIFY (MM) commands.

The VERIFY command (paragraph 3.5.26) should be used to ensure that the tape has been properly loaded. The VERIFY command reads the tape but instead of putting the information into memory, it makes a comparison between the contents of memory and the records read from tape; the comparison is made after the S-records have been converted back to hex. The sequence of steps for the VERIFY directive is very similar to those required for the LOAD — position the tape, enter the command line including carriage return, start the tape, wait for the prompt, turn the tape player off. The command line is:

```
TUTOR  1.X > VE4  cr
```

If the entire program verifies correctly, only the prompt will be returned. If differences are found between the tape and the RAM, these differences will be listed. For example, in the program example, verify errors can be forced by modifying the RAM after the program has been loaded.

```
TUTOR  1.X > LO4  cr

TUTOR  1.X > MM 1000;W  cr
001000    0C00? cr
001002    0030? 0123.  cr

TUTOR  1.X > VE4  cr
S1131000-.-.0030-.-.-.-.-.-.-.-.-.-.-.-.-.
ERROR

TUTOR  1.X >
```

The contents of locations that did not verify are listed as read from tape. Locations within the same S-record that did verify are represented by dashes. In the example, locations $1002 and $1003 which were changed with the MEMORY MODIFY command did not verify, and their correct (original) contents are shown. After all the errors have been listed, the message ERROR is printed to indicate that one or more errors have been found.

Verification errors can also be generated by errors on the tape. If the output level (volume) from the tape player is incorrect, the MEX68KECB may not be able to read the tape properly. Adjusting the volume control, usually about mid-range, should solve this problem. If different tape players are used, the volume may need to be readjusted.

Errors may also result if the particular tape player used does not invert the information -- which makes no difference with audio tapes but which will affect the MEX68KECB. The MEX68KECB is factory-jumpered to receive inverted data. The incoming data is inverted by the ECB receiver hardware; however, since the receiver firmware expects a non-inverted signal, a second inversion must be provided. The MEX68KECB expects inverted data from the tape player. If the tape player used does not invert the data, an additional inversion can be done on the MEX68KECB with a cut and jump option. See Chapter 6 for the details.

If the S-records are put onto the tape by another computer (i.e., not an MEX68KECB), the tape format used may not be compatible with the Educational Computer. The Educational Computer uses frequency shift keying (FSK) to code the data. A '1' is represented by one period of a 50% duty cycle 2000 Hz square wave. A '0' is represented by one period of a 50% duty cycle 1000 Hz square wave.

When downloading or verifying files from a remote host to the ECB, it is possible that data will be lost for various reasons such as losing an S-record(s) while printing out errors in a previous S-record. To prevent this, the ECB will send characters to the host to stop and start the transferral of the S-records. Various hosts require different characters to do this, and some have no provision for this kind of flow control.

The appropriate start and stop characters should be entered by the user in the first and second bytes of the options variable. The PF command displays the address of the 6-byte variable called OPTIONS.

```
TUTOR  1.X > PF

FORMAT=   15 15
CHAR NULL=00 00
C/R  NULL=00 00
OPTIONS@XXXXXX
```

XXXXXX is the absolute address of the 6-byte variable OPTIONS

The first byte is the transfer on (start) character and the second byte is the transfer off (stop) character. The other four bytes are used by the TM command and when mechanical terminals are used. Refer to paragraphs 3.5.21 and 3.5.23 and Appendix B for a discussion of these bytes.

Both the start and stop characters are initialized to $00 (NUL) on RESET. The bytes can be changed to effect data flow control. With an EXORciser or EXORmacs as host and loading from MDOS or VERSAdos, the flow can be halted with CTRL W ($17) and resumed with any other character (such as a space -- $20). Alternatively, some time-share systems use the device control characters DC1 ($11) and DC3 ($13) as the start and stop characters, respectively.

4.5.3 Upload to a Host

Uploading to a host is another method of saving programs and it also uses the DUMP command. A file is usually uploaded through Port 2. In order to upload successfully, the host must contain a program to input S-records from the RS-232 port and save them. The Educational Computer Board can upload to any host which has an RS-232 port and the required program. Motorola's EXORmacs and EXORciser development systems are both suitable hosts.

4.5.3.1 EXORciser as Host. Before sending S-records to the EXORciser, it must first be conditioned to receive them. To do this, enter the transparent mode (TM, paragraph 3.5.23), which allows the terminal to 'talk' directly to the host/EXORciser. The EXbug LOAD command (refer to the EXORciser User's Guide) will be used to input the S-records from the educational computer, convert them to hex, and store them in RAM. After receiving the EXbug prompt, key in the LOAD directive and select the S option, which loads a single file.

When connected to an EXORciser I, it is necessary to type an "X" following the TM command in order for the EXORciser to respond. For EXORciser II, it should be Control X, as follows:

EXORciser I

```
TUTOR  1.X> TM cr
*TRANSPARENT* EXIT=$01=CTL A
X
EXBUG 1.X LOAD
SGL/CONT S
AC (CTRL A)
TUTOR  1.X> DU2 3000 302F cr
PHYSICAL ADDRESS 00003000 0000302F
TUTOR  1.X>
```

EXORciser II

```
TUTOR  1.X> TM cr
*TRANSPARENT* EXIT=$01=CTL A
XC (CTRL X)
EXBUG 2.X
E* LOAD cr
S/C S
AC (CTRL A)
TUTOR  1.X> DU2 3000 302F cr
PHYSICAL ADDRESS 00003000 0000302F
TUTOR  1.X>
```

The EXORciser is ready to accept the S-records. Exit the transparent mode by entering the exit character -- usually CTRL A. The command line to upload is very similar to the command for dumping to tape. The required parameters are again the beginning and ending addresses and the port option is Port 2.

When the prompt is returned, the entire file has been uploaded. The next step is to transfer the file to disk. Re-enter the transparent mode; several carriage returns may be required before the EXbug prompt is received.

```
TUTOR  1.X > TM cr
 cr cr cr
EXbug X.X
```

```
TUTOR  1.X > TM cr
CTRL X
EXBUG 2.X
```

When the prompt is received, boot MDOS and use the MDOS utility ROLLOUT to create the disk file.

Several types of problems may be encountered in the sequence just described. The EXORciser must contain enough RAM addressed at the same location as the program being transferred. If this is not the case, the program can be moved to a different address within the MEX68KECB, using the BLOCK MOVE (BM), paragraph 3.5.2, before it is uploaded. Also be aware that ROLLOUT cannot roll out memory which is overlaid by either MDOS or the ROLLOUT command itself. These software programs occupy address space $0-$3000. Refer to the ROLLOUT command description in the EXORdisk II/III system user's guide.

A second potential problem involves the formatting of Port 2. The EXORciser processes each S-record after it is read before reading the next record. This requires the MEX68KECB to break between each S-record to allow time for processing by the EXORciser. This is effectively accomplished by inserting a string of nulls after each S-record. Do this with the format Port 2 command.

```
TUTOR  1.X > PF2  cr
FORMAT= 15? cr
CHAR NULL=XX ? cr
C/R  NULL=XX ? 10 cr  (16 nulls after each line)
TUTOR  1.X >
```

The number of nulls required varies with the baud rate but should never be greater than $10 (i.e., 16).

4.5.3.2 EXORmacs as Host. Transferring files to an EXORmacs is accomplished in a slightly different manner than transferral to an EXORciser. Using the VERSAdos utility UPLOADS, the file can be transferred directly to disk, bypassing the intermediate step. In order to use the UPLOADS utility, Port 2 of the Educational Computer must be connected to the MCCM in the EXORmacs.

Before attempting to transfer a file, enter the transparent mode and bring up VERSAdos. Call up the UPLOADS utility, giving the file name. Return to TUTOR. Now dump the file to Port 2 in the same way as before.

```
TUTOR  1.X > DU2 1000 102F  cr
TUTOR  1.X >
```

A disk file has been created. Re-enter the transparent mode to communicate with UPLOADS for any error messages.

4.5.4 Download from a Host

Files are retrieved from the host and checked for load errors with the LOAD and VERIFY commands. The files must be in S-record format. Besides retrieval of programs created using the MEX68KECB assembler/disassembler, the LOAD command is a handy tool for loading MC68000 language programs created using the host's resident or cross assembler. Such assemblers currently exist for the EXORmacs, EXORciser, and other potential hosts.

4.5.4.1 EXORciser as Host. The download sequence from an EXORciser is slightly different, depending on whether the files were originally uploaded or were created within the host. The file to be downloaded must be in S-record format. The ROLLOUT command does not create such a file. In this case, the file can be easily converted to S-record format, using the MDOS utility BINEX.

Once an S-record format is available, the download procedure is the same. MDOS should be loaded under the transparent mode and then control should be returned to TUTOR. LOAD and VERIFY can now be used as described in the section on loading and verifying from tape. The Port 2 option should be selected and a directive must be sent to the EXORciser via the RS-232 port. Any MDOS directive which sends the S-record formatted file to the RS-232 port can be used.

```
TUTOR  1.X > LO2 ;=COPY GETHEX.LX,#CN  cr
TUTOR  1.X >

      or

TUTOR  1.X > LO2 ;=LIST GETHEX.LX  cr
TUTOR  1.X >
```

Care must be taken to assure that any files created at the host are addressed within the user RAM area on the MEX68KECB. Also be aware that when disk files are created, entire sectors are allocated; any extra locations are filled with zeros. Therefore, the file may be slightly longer than the original program.

4.5.4.2 EXORmacs as Host. The download sequence from the EXORmacs is virtually the same as from an EXORciser. After booting VERSAdos in the transparent mode, return to TUTOR. Use the LOAD and VERIFY commands with the VERSAdos directive given in the ASCII string following the LOAD or VERIFY command as before. In this case, however, the S-record files are usually designated by the suffix MX rather than LX.

```
TUTOR  1.X > LO2 ;=LIST GETHEX.MX  cr
TUTOR  1.X >
```

The same cautions apply with regard to program addresses and disk files.