MEX68KECB/D2

MC68000

EDUCATIONAL COMPUTER BOARD

USER'S MANUAL

MOTOROLA

IF YOU HAVE ANY QUESTIONS
REGARDING THIS EDUCATIONAL MANUAL
PLEASE CALL

MOTOROLA SPS
UNIVERSITY SUPPORT
432 NORTH 44TH STREET, SUITE 200
PHOENIX, AZ 85008
PH: (602) 302-8166

MEX68KECB/D2

JULY 1982

MC68000

EDUCATIONAL COMPUTER BOARD

USER'S MANUAL

WARNING

THIS EQUIPMENT GENERATES, USES, AND CAN RADIATE RADIO FREQUENCY ENERGY AND, IF NOT INSTALLED AND USED IN ACCORDANCE WITH THE INSTRUCTION MANUAL, MAY CAUSE INTERFERENCE TO RADIO COMMUNICATIONS. AS TEMPORARILY PERMITTED BY REGULATION, IT HAS NOT BEEN TESTED FOR COMPLIANCE WITH THE LIMITS FOR CLASS A COMPUTING DEVICES PURSUANT TO SUBPART J OF PART 15 OF FCC RULES, WHICH ARE DESIGNED TO PROVIDE REASONABLE PROTECTION AGAINST SUCH INTERFERENCE. OPERATION OF THIS EQUIPMENT IN A RESIDENTIAL AREA IS LIKELY TO CAUSE INTERFERENCE, IN WHICH CASE THE USER, AT HIS OWN EXPENSE, WILL BE REQUIRED TO TAKE WHATEVER MEASURES MAY BE REQUIRED TO CORRECT THE INTERFERENCE.

Second Edition

First Edition January 1982

Dear Customer,

Congratulations on your recent purchase of the 68K Educational Computer Board.

YOUR MOTOROLA WARRANTY

Motorola Inc. warrants this product against defects in material and workmanship for a period of ninety (90) days from the original data of purchase. THIS WARRANTY EXTENDS TO THE ORIGINAL CUSTOMER ONLY AND IS IN LIEU OF ALL OTHER WARRANTIES, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. In no event will the Seller be liable for any incidental or consequential damages.

During the warranty period, Motorola will replace, at no charge, components that fail, provided the product is returned (properly packed and shipped prepaid) to Motorola at the address printed below. Dated proof of purchase (such as a copy of the sales receipt or bill of sale) must be enclosed with shipment to validate warranty. We will return the shipment prepaid via UPS.*

This warranty does not apply if, in the opinion of Motorola, the product has been damaged by accident, misuse, neglect, misapplication, or as a result of service or modification by other than the authorized Motorola Service Center referenced below.

AFTER-WARRANTY SERVICE

After your product is ninety (90) days old or warranty has been voided in any manner, you may return the product to us for repair. Please be sure to ship prepaid, properly pack the product, and include a certified check or money order for $150.00 payable to Motorola Inc. Include any failure information you may have to help expedite repair time. We will return the shipment prepaid via UPS.* We hope you will never need our repair, but it's nice to know you are protected anyway — and that help is nearby.

MOTOROLA C&E INC.
Phoenix Service Depot
3332 E. Broadway Suite 102
Phoenix, Arizona 85040-2830
PH: 602-437-4331
FAX: 602-437-1858

* FOR EXPEDITED RETURN SHIPMENTS, PLEASE INCLUDE AN ADDITIONAL $20 IN YOUR PAYMENT. THIS CHARGE IS TO COVER AIR SHIPMENT. PRICES ARE SUBJECT TO CHANGE WITHOUT PRIOR NOTICE. PLEASE ENCLOSE A COPY OF THE ATTACHED PROBLEM REPORT WITH YOUR BOARD.

P R O B L E M R E P O R T

NAME _____

COMPANY _____

ADDRESS _____

CITY, STATE, ZIP CODE _____

PHONE NUMBER _____

PLEASE FILL IN THE APPROPRIATE AREAS

WARRANTY (90 DAYS FROM RECEIPT OF PRODUCT)

[ ] YES (BE SURE TO INCLUDE A COPY OF YOUR SALES RECEIPT)

[ ] NO (BE SURE TO INCLUDE A CERTIFIED CHECK OR MONEY ORDER FOR $150.00*)

EXPEDITED SHIPPING

[ ] YES (BE SURE TO INCLUDE AN ADDITIONAL PAYMENT OF $20.00*)

[ ] NO

APPLICATION INFORMATION _____

_____

_____

_____

FAILURE INFORMATION _____

_____

_____

_____

* PRICES SUBJECT TO CHANGE WITHOUT NOTICE

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (cont'd)

LIST OF TABLES

CHAPTER 1

INTRODUCTION TO THE MC68000 EDUCATIONAL COMPUTER BOARD

The purpose of this manual is to provide the user with a comprehensive guide for understanding and utilizing the MC68000 Educational Computer Board. The computer board is intended primarily as a self-supporting means for evaluating and learning about the MC68000 16-bit microprocessor. Chapter 1 contains information describing the board and its system configuration.

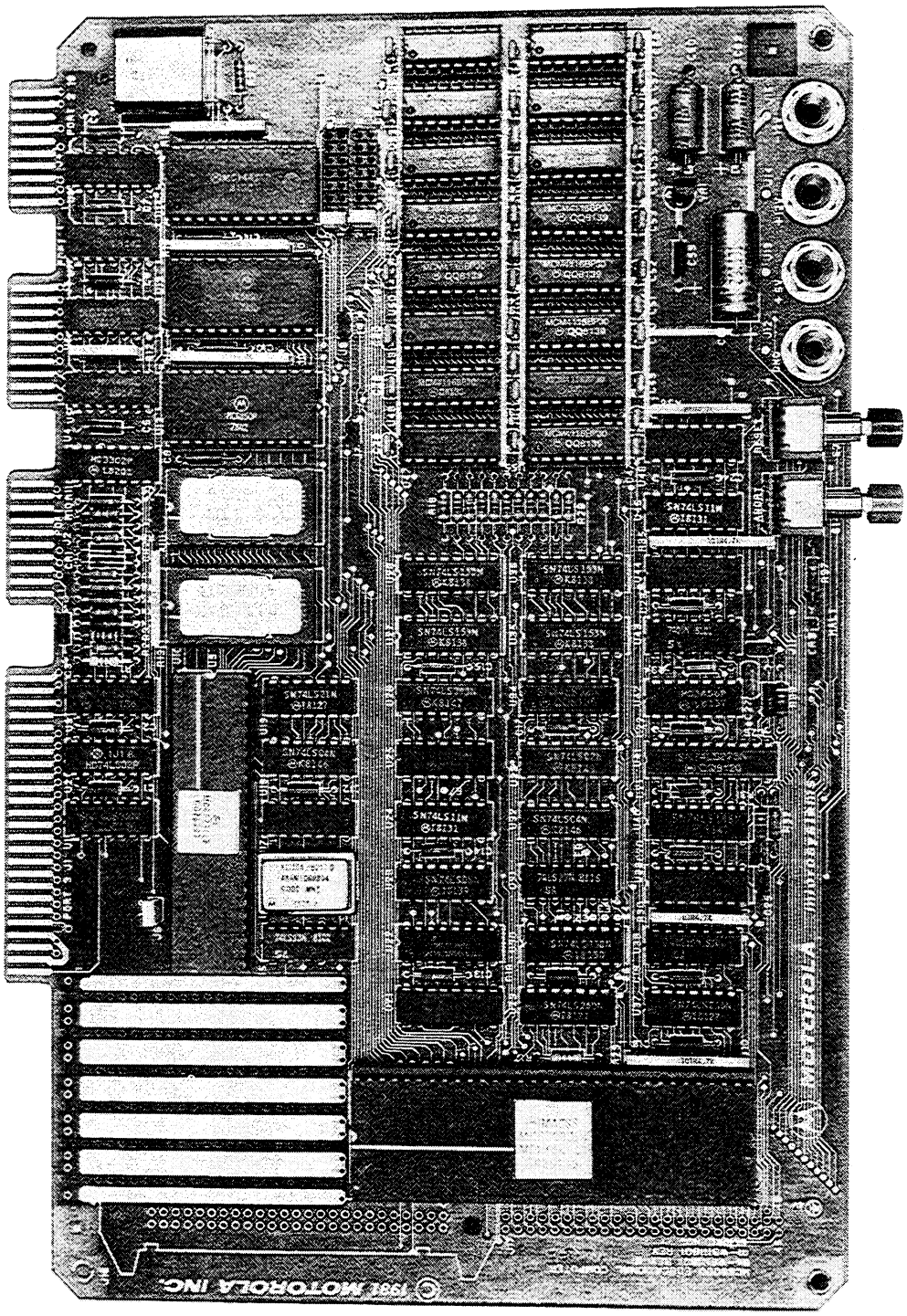


FIGURE 1-1. MC68000 Educational Computer Board

CHAPTER 1

INTRODUCTION TO THE MC68000 EDUCATIONAL COMPUTER BOARD

1.1 WHAT IS THE MC68000 EDUCATIONAL COMPUTER BOARD?

Intended primarily for training and educational use, including college-level courses and industrial in-plant training, the MC68000 Educational Computer Board (ECB) serves as an economical yet comprehensive introduction to systems based on the M68000 family of microcomputer products. Located on a single small printed circuit (PC) card (Figure 1-1), a complete microprocessor system is provided, including an MC68000 16-bit microprocessor, memory, parallel input/output (I/O), and serial communications I/O. The user must only connect an RS-232C-compatible "dumb" terminal and power supplies to have a functional system.

For ease-of-use, the ECB has a resident firmware package that provides a self-contained programming and operating environment. The firmware, aptly named "TUTOR", provides the user with monitor/debug, assembly/disassembly, program entry, and I/O control functions. Utilizing the capabilities provided by the system, the user can investigate and learn the computing power and architectural features of the MC68000. This system also provides a working example of the microprocessor external bus structure and interface to memory and peripheral devices.

The Educational Computer Board's features include:

a. 4-megahertz MC68000 16-bit MPU.

b. 32K bytes of dynamic RAM (DRAM) arranged as 16K x 16.

c. 16K-byte firmware ROM/EPROM monitor addressed as 8K x 16.

d. Two serial communication ports provided for a terminal and a host. Both are RS-232C-compatible and have selectable baud rates.

e. Programs can be downloaded from or uploaded to a host system.

f. A parallel port (16 data lines with handshake) can be used for I/O or for a Centronics-compatible printer interface.

g. Audio tape serial I/O port.

h. Self-contained operating firmware that provides monitor, debug, and disassembly/assembly functions.

i. 24-bit programmable timer.

j. Wire-wrap area provided for custom circuitry.

k. RESET and ABORT function switches.

1.2 GENERAL HARDWARE DESCRIPTION

The MC68000 Educational Computer Board provides the RAM, ROM, timer, and I/O necessary for learning and evaluating the attributes of the MC68000. This microprocessor has a 16-bit data bus and a 23-bit address bus (A1-A23). The address bus is, in effect, 24 bits and provides a memory addressing range of 16 megabytes. The processor also has eight 32-bit data registers, seven 32-bit address registers, two 32-bit stack pointers, a 32-bit program counter, and a 16-bit status register. The MC68000 Data Sheet and User's Manual (MC68000UM), which are included in the ECB's documentation, describe the device in detail.

A 4-MHz MC68000 MPU is used on the educational board (a functional block diagram is shown in Figure 1-2). All the memory and I/O devices communicate with the MPU via a common parallel bus. The various functional areas of the board are described briefly in the following paragraphs.

1.2.1 System Memory

The system memory consists of 32K bytes of dynamic RAM and 16K bytes of ROM or EPROM (two 8-bit bytes = 1 word). The RAM is used both for scratchpad space for the TUTOR firmware and for user programs. Approximately 2K bytes are reserved for the monitor scratchpad; the remaining RAM (approximately 30K bytes) is available to the user. The system firmware occupies the 16K-byte read-only memory.

1.2.2 Serial Communications Ports

Two asynchronous serial communication ports, designated Port 1 for the Terminal and Port 2 for the Host, are provided on the board. Both of these ports are RS-232C-compatible (an E.I.A. standard). The terminal that provides user interface is connected to the educational computer via Port 1, and Port 2 can be connected to a modem or directly to a host computer. The host computer can be used to provide more powerful software capabilities such as program assembly, file management, and editing, and for downloading or uploading programs. Also, an operational condition called transparent mode can be used on the MEX68KECB. This transparent mode effectively bypasses the board and allows the terminal to communicate directly with the host. The terminal and host baud rates must be the same for this mode.

Both serial ports can be jumpered for various data transmission rates (110-9600 baud). Also, if required, either port can be modified to transmit and receive at different baud rates.

1.2.3 Programmable Timer

Contained within the MC68230 PI/T device is a 24-bit general purpose timer. The timer is a synchronous counter that can be used for generating or measuring both time delays and various frequencies. The timer can be clocked by a 5-bit prescaler or directly, and the clock source can be the 4-MHz system clock or an external clock.

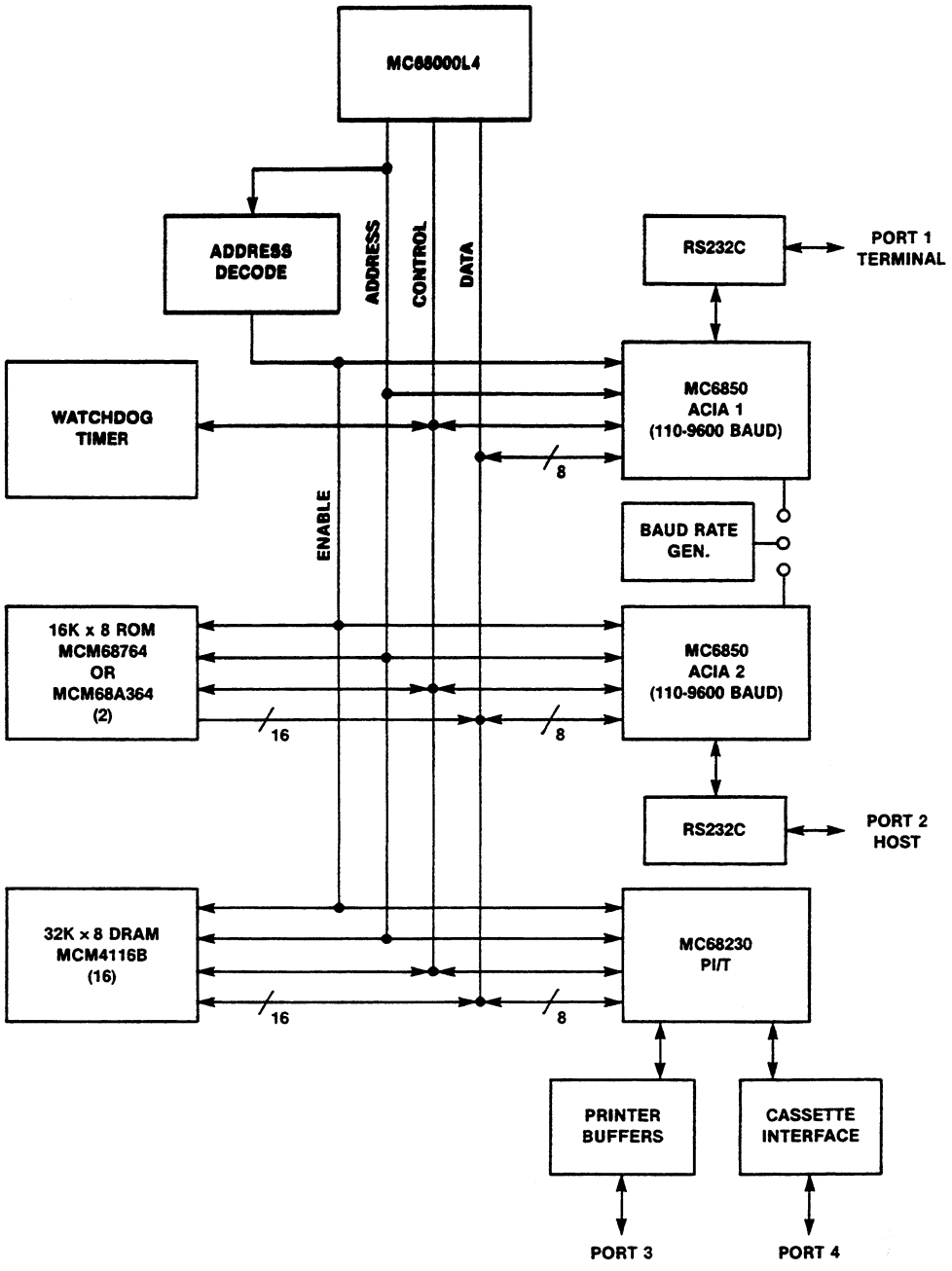


FIGURE 1-2. Functional Block Diagram

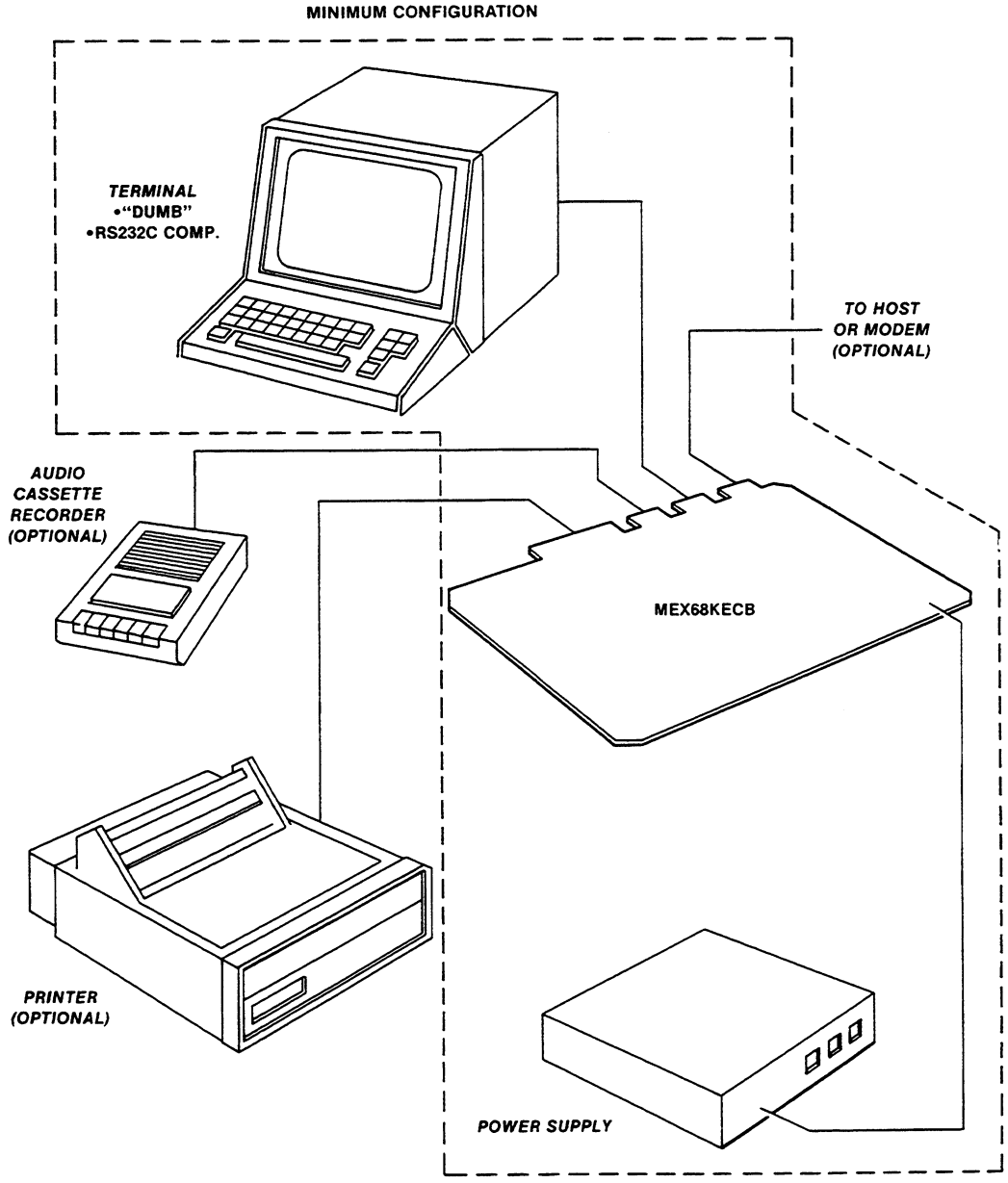


FIGURE 1-3. System Configurations

An assembly listing of the TUTOR firmware, excluding the interactive assembler and disassembler modules, can be purchased from Motorola under the part number M68KTUTOR. Machine-readable source for all the modules can be purchased under part numbers:

M68KTUTORS - VERSAdos 8" floppy diskette
M68KTUTORT - VERSAdos hard disk cartridge

For program development, an interactive assembler/editor function is used in which the source program is not saved. Each instruction is translated into the proper object code and is stored in memory on a line-by-line basis at the time of entry. The assembler source statement is composed of operation and operand fields; line numbers, labels, and comments are not allowed.

In order to display an instruction, the firmware disassembles the object code and displays the instruction mnemonic and operands. Editing is done by re-entering a source statement.

If higher-level assembly capabilities are required, a macro assembler or cross assembler can be run on a host computer. Data can be uploaded and downloaded to the host via serial Port 2.

1.5 SPECIFICATIONS

Table 1-1 lists basic specifications for the MC68000 Educational Computer Board (MEX68KECB).

TABLE 1-1. Specifications

Microprocessor	MC68000 (4 MHz)
Input/Output	
Parallel I/O	MC68230 (16 data lines, 4 control lines) normally configured as Centronics-type printer interface. 24-bit programmable timer included in MC68230.
Cassette Interface	1300 baud serial audio tape.
Serial I/O Ports	Two — one terminal and one host (modem).
Interface	RS-232C interface.
Baud rate	Strap selectable: 110, 150, 300, 600, 1200, 2400 4800, 9600.
System clock	8-MHz crystal providing 4-MHz processor operation.
Memory	32K bytes RAM 16K bytes ROM
Power requirements (Typical)	+5.0 V/750 mA, +12 V/50 mA, -12 V/50 mA
Operating temperature	0 to 50° C
Board Dimensions (Approx.)	
L x W x H	7.5 in. x 10.5 in. x 1.5 in. (19 cm x 27 cm x 4 cm)

CHAPTER 2

INSTALLATION AND POWER-UP INSTRUCTIONS

This chapter provides unpacking, preparation-for-use, installation, and power-up instructions for the MEX68KECB. The board has been designed to require a minimum of hardware modifications; however, the proper serial port baud rates must be selected and the proper cables used to ensure trouble-free start-up. Please read and follow the instructions in this chapter to provide quick start-up and to avoid possible damage to the board.

CHAPTER 2

INSTALLATION AND POWER-UP INSTRUCTIONS

2.1 UNPACKING INSTRUCTIONS

NOTE

If shipping carton is damaged on receipt, request carrier's agent be present during unpacking and inspection of the module.

Unpack the computer board from its shipping carton. Save the packing material for storing or reshipping the board. Refer to the packing list and verify that all items are present. As shipped, the MEX68KECB includes:

a. MEX68KECB Educational Computer Board

b. Four 6-32x1/4" screws

c. Four threaded 6-32x3/4" nylon standoffs

d. Four banana jacks (including hex nuts and solder lugs)

e. Seven plastic cap jumpers

After verifying that items (including any optional parts) are present, inspect the board for damage. Ensure that there are no broken, damaged, or missing parts, and that there is no physical damage to the printed circuit board.

CAUTION

WHEN HANDLING THE BOARD, AVOID TOUCHING AREAS OF MOS CIRCUITRY; STATIC DISCHARGE CAN DAMAGE INTEGRATED CIRCUITS.

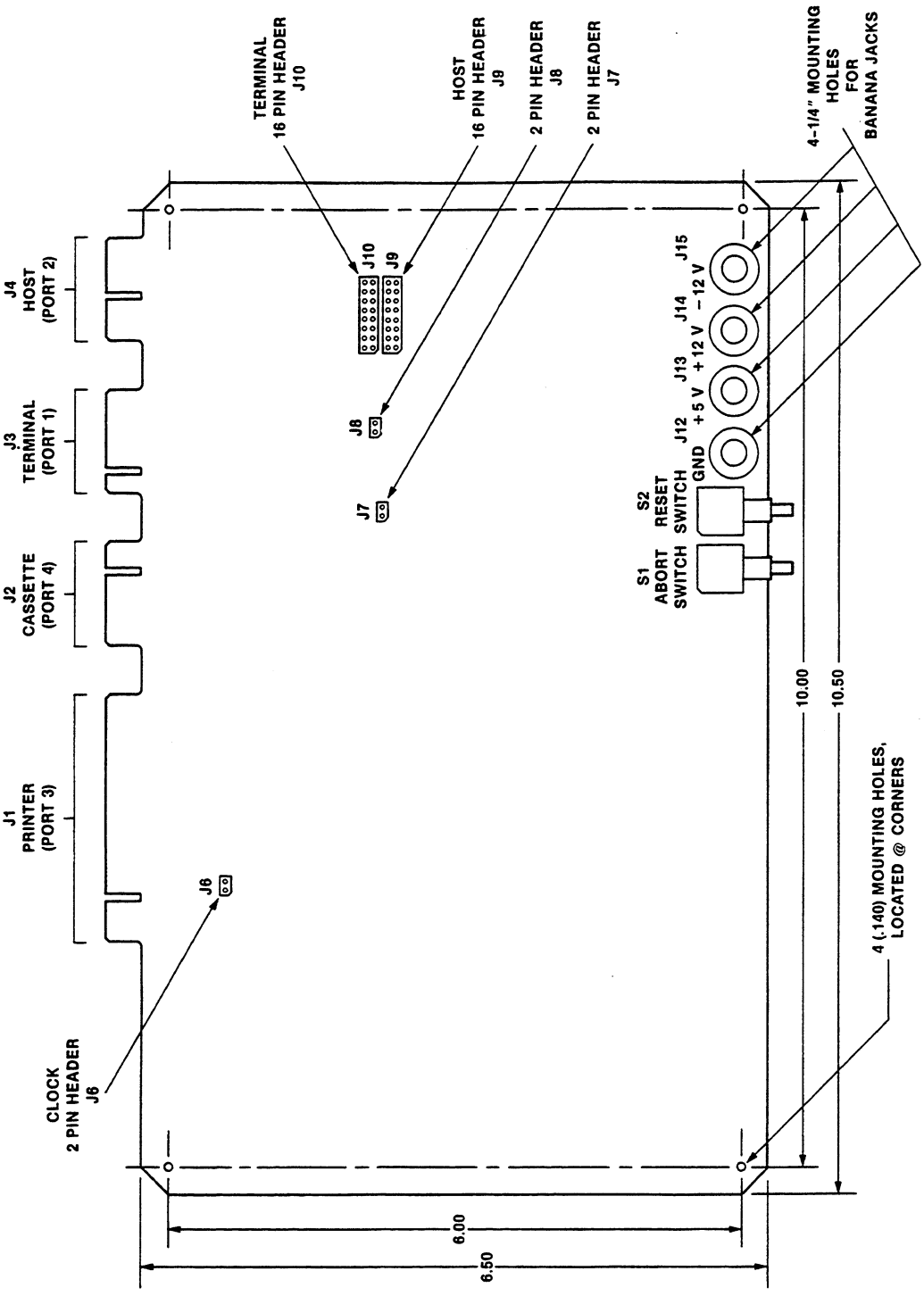


FIGURE 2-1. MEX68KECB Board Layout

2.2 PREPARING THE BOARD FOR USE

The MEX68KECB is intended primarily for training and educational use, including college-level courses and industrial in-plant training. In its most simple configuration, the board requires only an RS-232C compatible terminal (plus cable) and power supplies to function. The preparation instructions are intended to set up the board for this configuration. Use of optional features (audio cassette, host, or printer) requires additional preparation, which is covered in other sections of this chapter.

Figure 2-1 shows a layout of the MEX68KECB. Board preparation concerns the following items:

a. Because the board is intended for laboratory use, standoff legs can be used to allow the board to set on a bench.

b. Power connections must be made. Banana jacks can be used or wires can be soldered to the board.

c. Check that the system clock jumper (J6) is in place.

d. The terminal baud rate must be selected.

2.2.1 Attaching Standoff Legs

Four holes located at the corners of the board are used to mount the nylon standoffs. These are screwed to the back side (opposite of component side) of the card, as shown in Figure 2-2.

NOTE

The user may choose to mount the module on or in an enclosure via these holes. Dimensions are shown in Figure 2-1. The nylon standoffs can be used as spacers to provide clearance of 3/4 inch needed by the banana jacks.

2.2.2 Providing Power to the Board

2.2.2.1 Banana Jacks. Four banana jacks compatible with standard banana plugs are provided for power connectors (+12 Vdc, +5 Vdc, Ground, -12 Vdc). These are mounted in four 1/4-inch holes at a corner of the board, as shown in Figure 2-2.

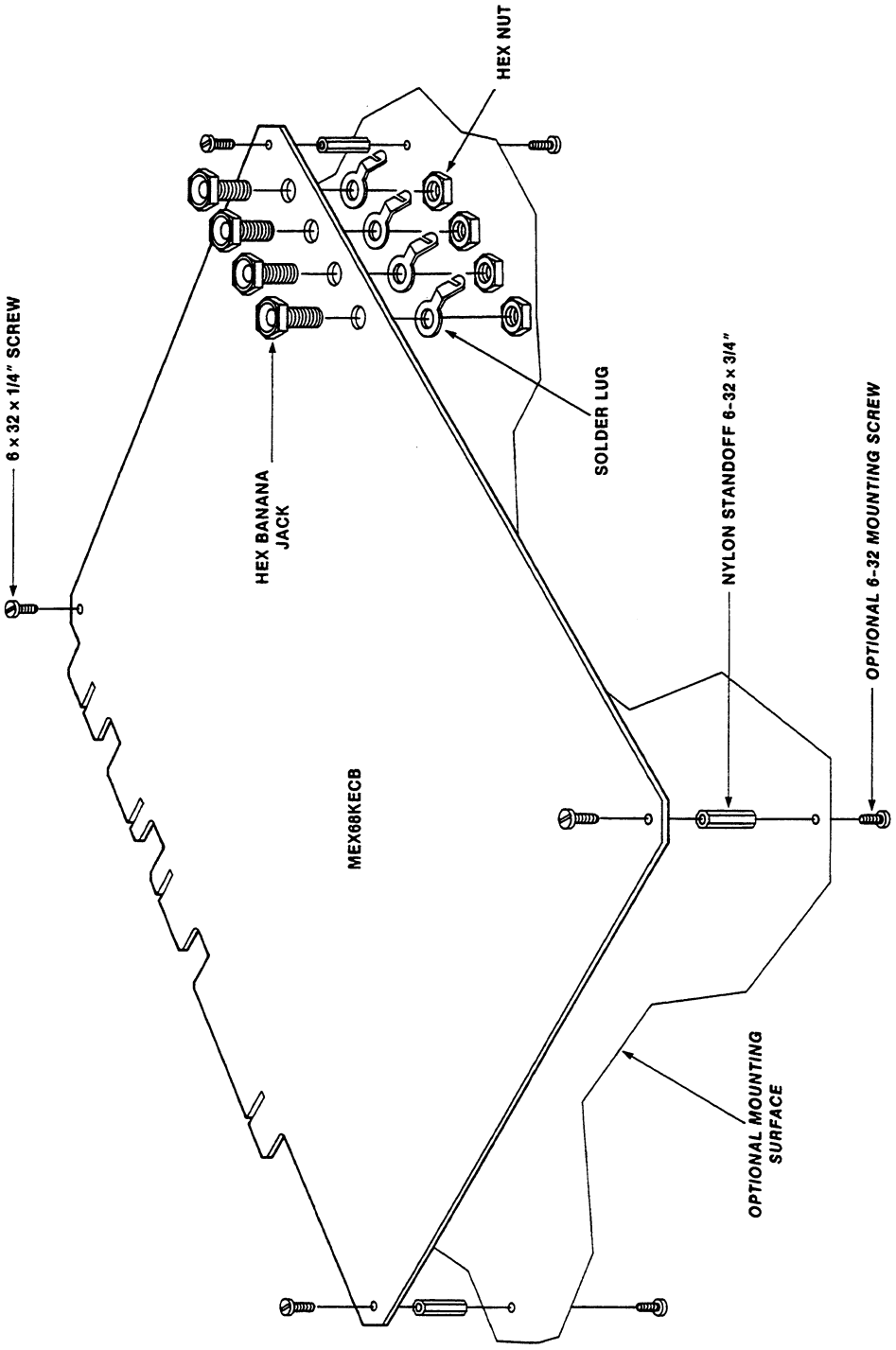


FIGURE 2-2. Hardware Mounting Detail

2.2.2.2 Alternate Method - Discrete Wires. If banana plugs are not desired, discrete wires can be used to supply power to the board. Wires can be soldered to the lugs supplied with the banana jacks (Figure 2-2) or four small holes are provided to solder discrete wires to the board. These holes (designated J12, J13, J14, J15) are shown in Figure 2-3, and interconnect supply voltages as follows:

HOLE DESIGNATION	VOLTAGE
J12	Ground
J13	+5 Vdc
J14	+12 Vdc
J15	-12 Vdc

NOTE

Use of the banana jack solder lugs is recommended because of greater mechanical strength and to prevent possible damage to the board.

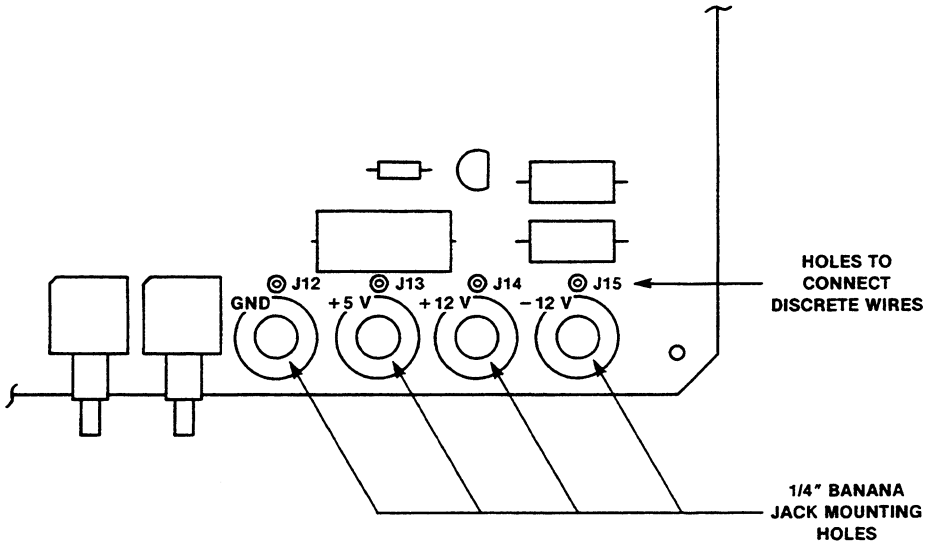


FIGURE 2-3. Detail Showing Location of Holes J12, J13, J14, J15 to Connect Discrete Power Wires

2.2.3 Checking System Clock Jumper

Referring to Figure 2-1, a 2-pin header designated J6 should have a plastic jumper cap in place on it (as shipped). If not, one of the jumpers supplied with the board should be put in place. This jumper connects the system clock source.

2.2.4 Selecting Terminal Baud Rate

2.2.4.1 Normal Operation - Transmitting and Receiving at the Same Baud Rate. Normally, the terminal transmits and receives at the same baud rate. Although the terminal Port 1 can be configured to transmit and receive at different rates, the board as supplied uses a single common baud rate for the port. The host interface Port 2 has similar attributes. The interconnection circuit is shown in Figure 2-4.

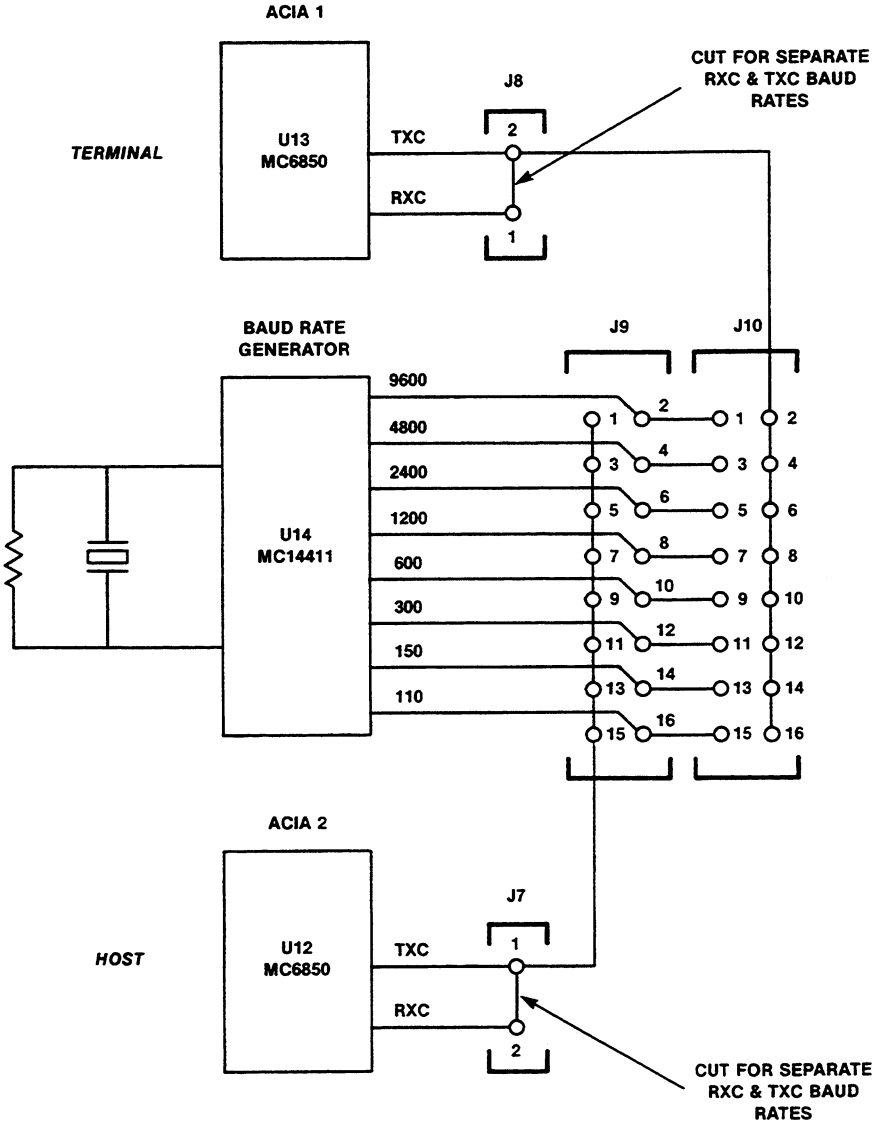


FIGURE 2-4. Interconnection Diagram for Baud Rate Selection

Referring again to Figure 2-1, two headers marked J9 and J10 are shown. Each header consists of a double row of eight pins (16 total), and is used to select the baud rate for a serial port. Header J10 is used to select the terminal baud rate (ACIA 1, serial Port 1).

The pins are jumpered together using a plastic jumper cap (one of seven provided with the board). The cap should be positioned on header J10, as shown in Figure 2-5, to select the desired baud rate. Table 2-1 lists which pins must be jumpered for a given baud rate.

TABLE 2-1. Headers J9 and J10 Jumpers to Select Serial Port Baud Rates

JUMPER PINS	SELECTED BAUD RATE
1-2	9600
3-4	4800
5-6	2400
7-8	1200
9-10	600
11-12	300
13-14	150
15-16	110

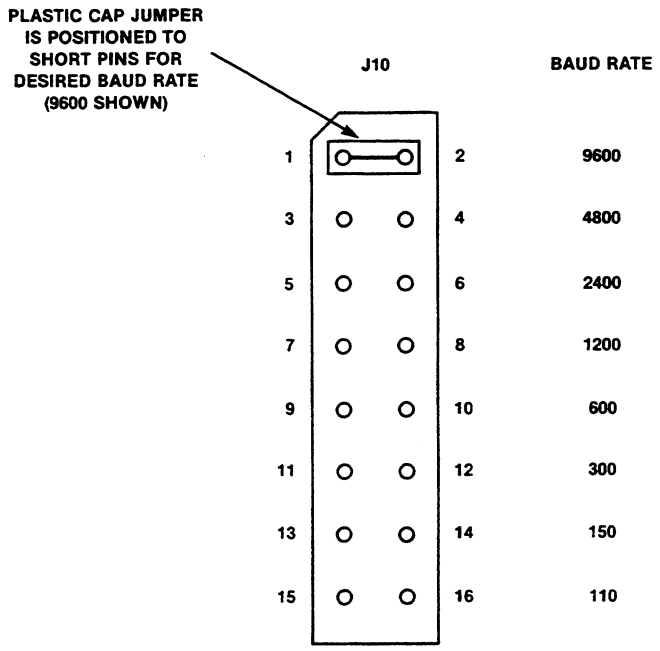


FIGURE 2-5. Terminal Baud Rate Select Jumper (J10)

Figure 6-4 is a functional schematic diagram of the serial communication ports. The RS-232C signal lines required at each port are shown.

Port 1 must receive an active level on DTR (data terminal ready), or data will not be transmitted. The terminal connected to Port 1 must drive DTR. CTS (clear to send), DSR (data set ready), and DCD (data carrier detect) are each asserted when DTR is asserted. Refer to Appendix C for further information.

2.2.4.2 Special Operation - Transmitting and Receiving at Different Baud Rates. The MEX68KECB is wired with the transmit clock (TXC) and Receive clock (RXC) of each ACIA (refer to Figure 2-4) tied together and then jumpered to the selected baud rate. To provide different baud rates, the connection between TXC and RXC must be cut and individual baud rates connected to each. Perform the following steps to select separate transmit and receive baud rates for the terminal:

a. Cut the signal trace located between Pin 1 and Pin 2 of header J8 on the back side of the printed circuit board. BE CAREFUL — be sure to cut the correct trace; it is approximately 1/8 inch long.

b. The transmit baud rate (TXC) is selected by using the plastic jumper cap on header J10 in accordance with Table 2-1.

c. The receiver baud rate (RXC) is selected by wire-wrapping Pin 1 of header J8 to the desired odd numbered pin of header J10. Again, use Table 2-1 to determine the correct pin.

> **NOTE**
>
> The MEX68KECB as now configured is ready to be connected to a terminal and power supplies. If the user wants to utilize options of the printer, tape recorder, and/or host computer (serial Port 2), additional preparation is required. See the appropriate section in this chapter on each option.

2.3 SYSTEM HOOKUP INSTRUCTIONS

As previously stated, the most simple configuration requires only the MEX68KECB, a terminal, and power supplies (Figure 2-6.) This section describes the required interconnections to hook up this configuration.

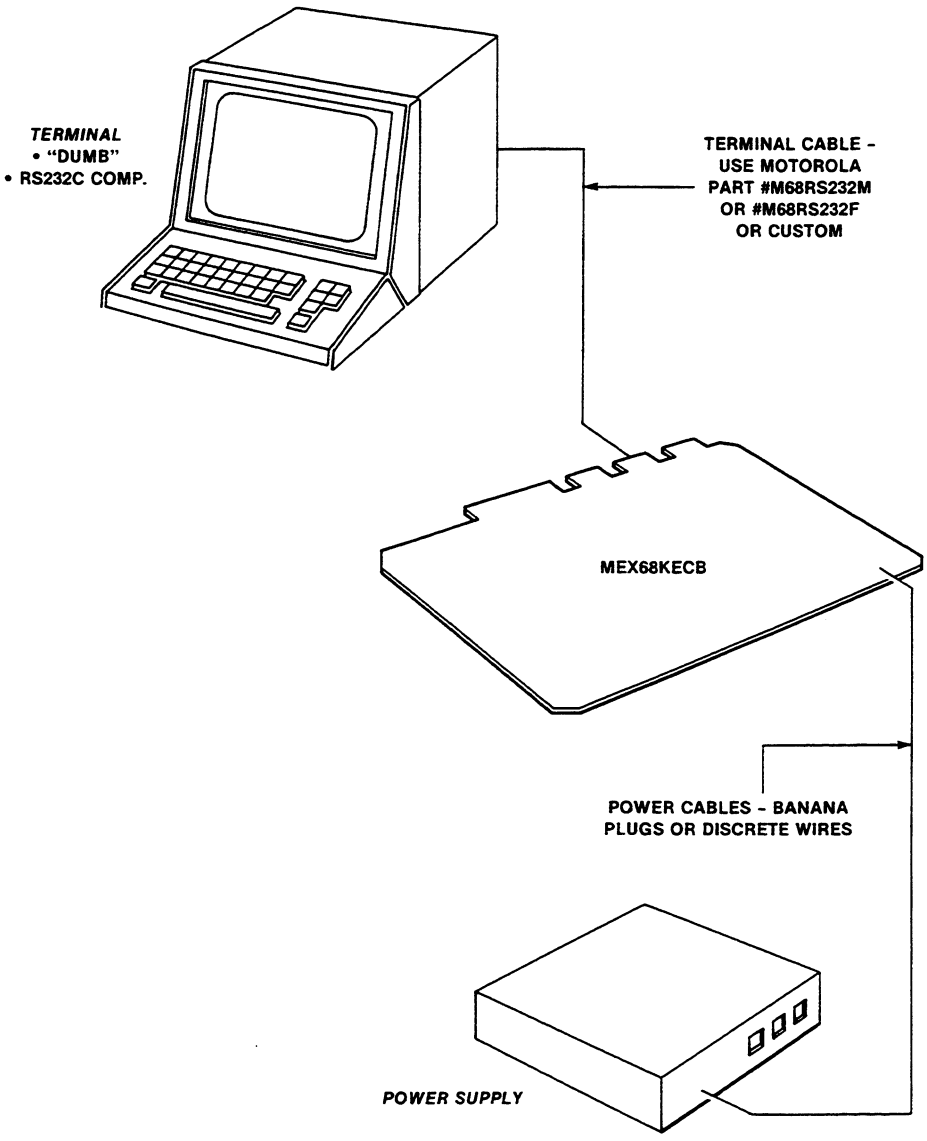


FIGURE 2-6. Minimum System Configuration

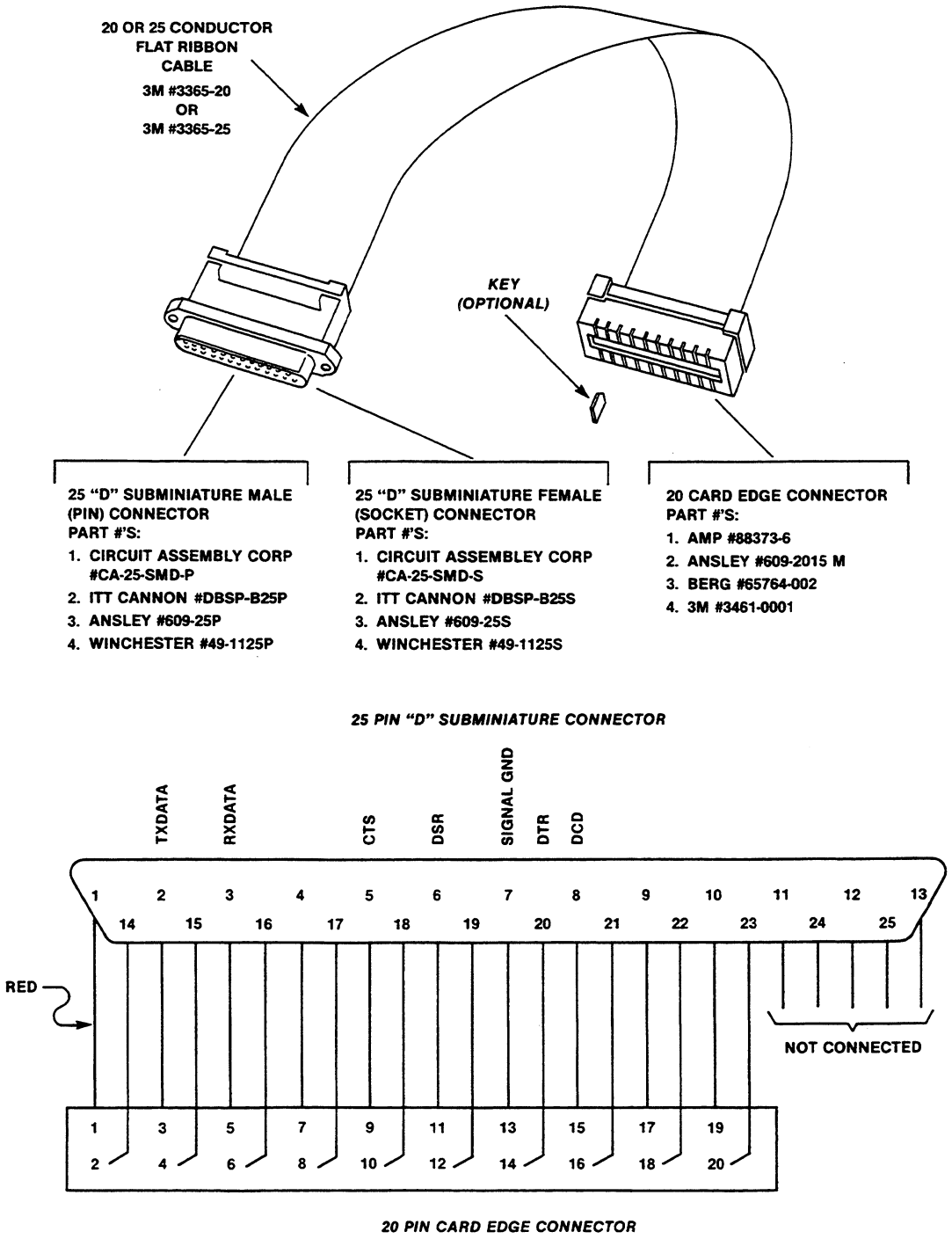


FIGURE 2-7. Terminal Cable Detail and Signal Line Connections

2.3.1 Connecting the Terminal

The terminal is connected to the MEX68KECB with a cable (normally flat ribbon) requiring a 20-contact card edge connector on the MEX68KECB end and a 25-contact "D" subminiature connector on the terminal end. The "D" subminiature connector can be either pin (male) or socket (female), as required by the user terminal. Both of these cable types are available from Motorola:

PART NUMBER	DESCRIPTION
M68RS232M	RS232 CABLE - Card edge connector/Male DB25 connector
M68RS232F	RS232 CABLE - Card edge connector/Female DB25 connector

As an alternative, the user can manufacture his own cable. Figure 2-7 shows a detail of the cable, lists several suitable vendor part numbers (any equivalent part can be used), and shows the conductor line designations. The cable requires a 25-conductor flat ribbon; connectors should be installed according to manufacturer's directions. Also, the card edge connector can be keyed to prevent incorrect cable connection.

The suitable cable is connected to Port 1 (connector J3), as shown in Figure 2-8, with the other end going to the terminal.

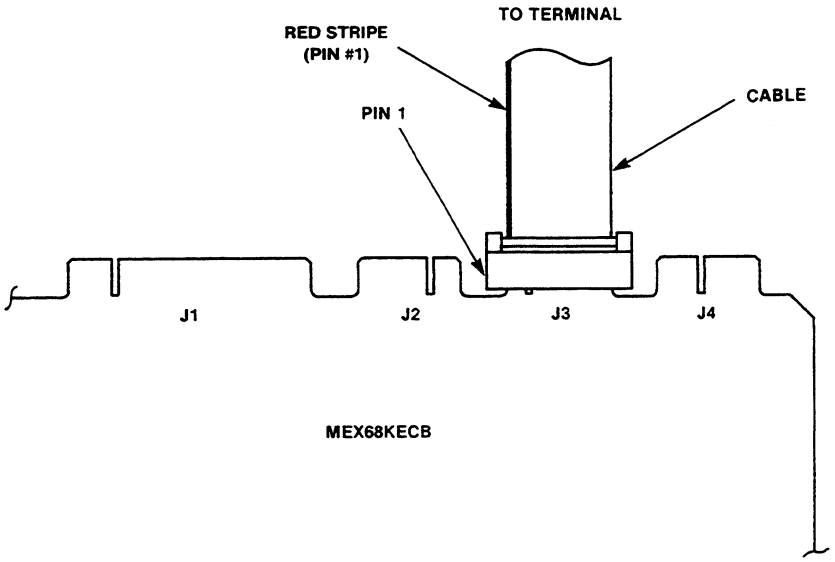


FIGURE 2-8. Terminal Cable Connection to MEX68KECB

2.3.2 Connecting the Power Supplies

Three supply voltages are required for the board — that is, +5.0 Vdc ± 5%, +12.0 Vdc ± 10%, and -12.0 Vdc ± 10%. If the banana jacks are used, cables with standard banana plugs are required to connect the power supplies. With discrete wires, the wires are connected directly to the suitable voltages.

All supply voltages must be referenced to ground, and these connections should be made before turning on power. The voltage turn-on must be done in proper sequence to prevent damage to the RAM devices. Follow system turn-on instructions in the following section.

2.4 SYSTEM TURN-ON AND INITIAL OPERATION

CAUTION

POWER SUPPLY VOLTAGES MUST BE TURNED ON IN PROPER SEQUENCE TO AVOID DAMAGE TO THE DYNAMIC RAM DEVICES. FOLLOW THE TURN-ON INSTRUCTIONS TO PREVENT PROBLEMS.

After the cables are in place, the final step to system turn-on is applying power. The dynamic RAM devices (MCM4116's) require that the negative voltage -12 Vdc be applied first. This is especially important when individual power supplies (such as laboratory supplies) are used. The power-up sequence should be:

a. Ground must be connected common to all power supplies.

b. Turn on -12.0 Vdc.

c. Turn on +12.0 Vdc.

d. Turn on +5.0 Vdc.

If a single multivoltage power supply is used, it is not possible to turn voltages on independently. However, with most power supplies the -12 Vdc and +12 Vdc come up before the +5 Vdc because these are lightly loaded and do not have to charge heavy internal filter capacitance. The user should test the multivoltage supply, simulating typical loading from Table 1-1, to determine if the -12 Vdc comes up first. When powering up with a single multivoltage supply:

a. Be sure all voltages are connected prior to power up.

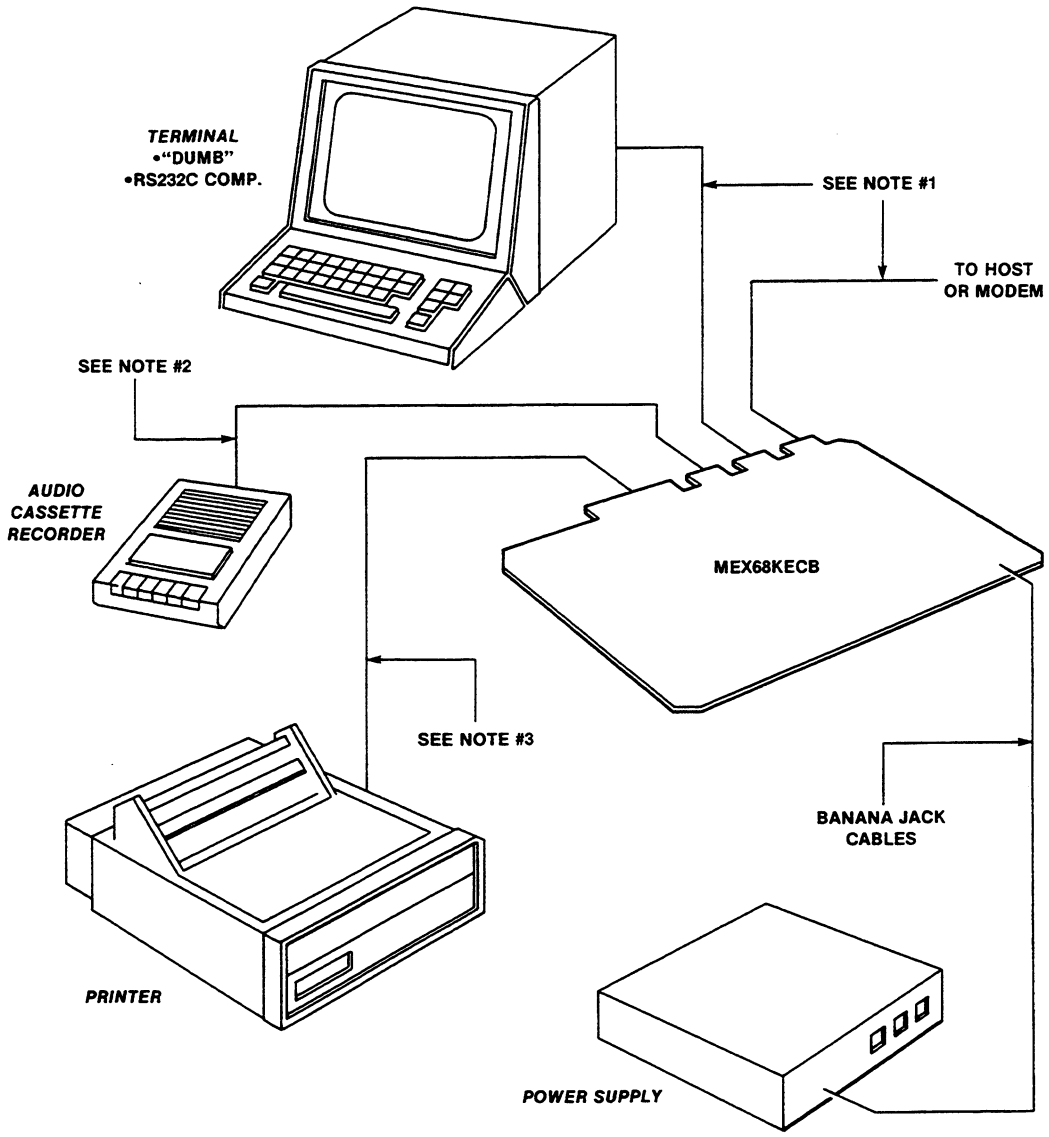
b. Turn power ON to the board.

CAUTION

THE POWER DOWN SEQUENCE IS THE REVERSE OF THE ABOVE POWER UP SEQUENCE AND IS EQUALLY IMPORTANT.

After power on, the system should initialize itself and print on the terminal:

```
TUTOR 1.X >
```



NOTES: 1. USE MOTOROLA CABLE M68RS232M OR M68RS232F OR MAKE CUSTOM.
2. MAKE CUSTOM CABLE.
3. USE MOTOROLA CABLE MEX68PIC OR MAKE CUSTOM.

FIGURE 2-9. Expanded System Configuration with Options

It is now ready for operation under control of the firmware as described in Chapters 3 and 4. If this response does not appear on the terminal, perform the following system checks:

a. Press the black reset button to guarantee that the board has been initialized properly.

b. Check that the terminal and board are set for the same baud rates.

c. If the baud rates are set properly and the terminal is still not reacting properly, the terminal may require special null characters and formatting from the educational computer. The Port Format (PF) command can be used to set the required ACIA format (see Paragraph 3.5.21 and Appendix B).

2.5 PREPARATION FOR USE OF SYSTEM OPTIONS

The MC68000 Educational Computer Board can use options of a Centronics-compatible printer, audio cassette storage, and a link to a host computer. Figure 2-9 shows the expanded system configuration with these options. The following paragraphs describe preparation for use of each option.

2.5.1 Printer Option

The board is properly buffered to directly drive a Centronics-compatible printer. The Port 3 edge connector J1 must be connected via cable to the user-supplied printer. This cable is available from Motorola, Part Number MEX68PIC.

The user may desire to manufacture this cable, although it is a more complex assembly than just flat ribbon cable and connectors. Resistors are used on the cable to help protect unbuffered inputs from damage due to static discharge. Figure 2-10 shows the interconnection diagram of the cable and lists suitable connector part numbers. Note that the cable can be keyed on the card edge connector to prevent incorrect cable connection.

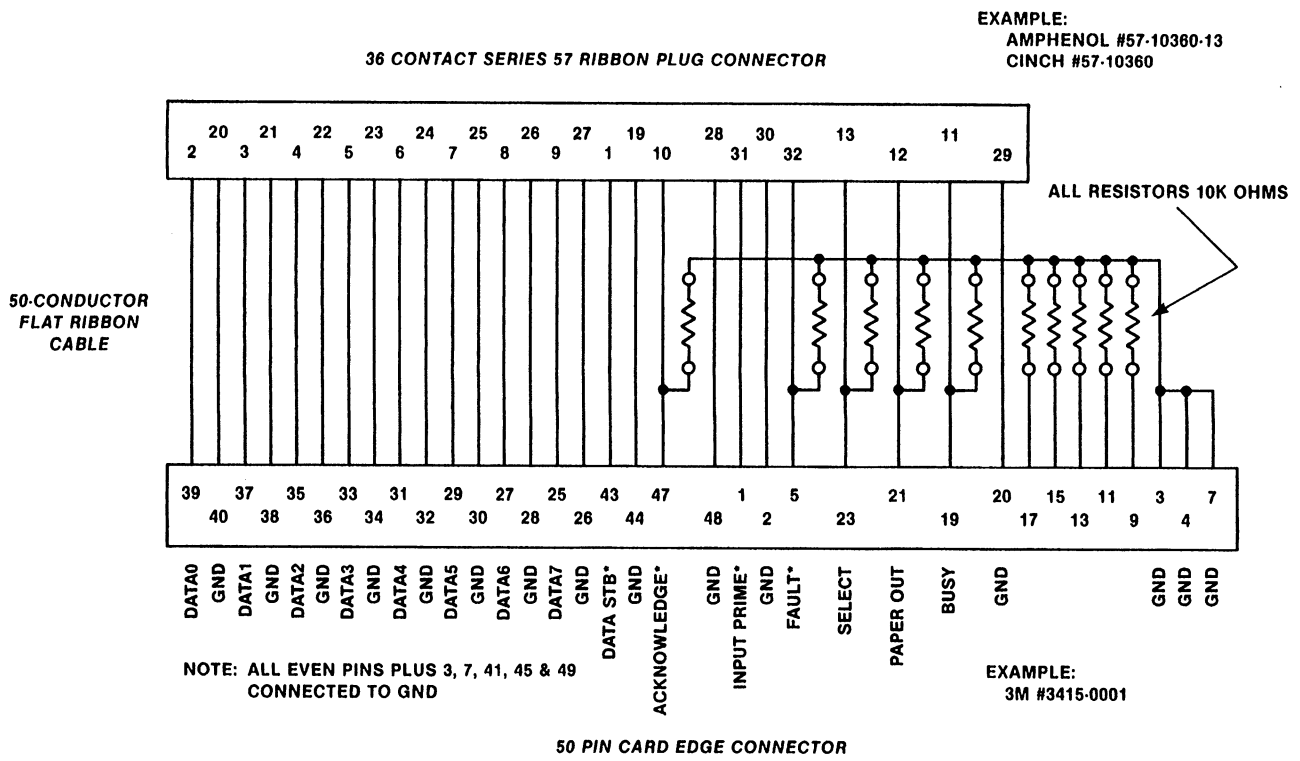


FIGURE 2-10. Printer Cable Interconnection Diagram

The cable is connected to Port 3 (connector J1), as shown in Figure 2-11, with the other end going to the printer.

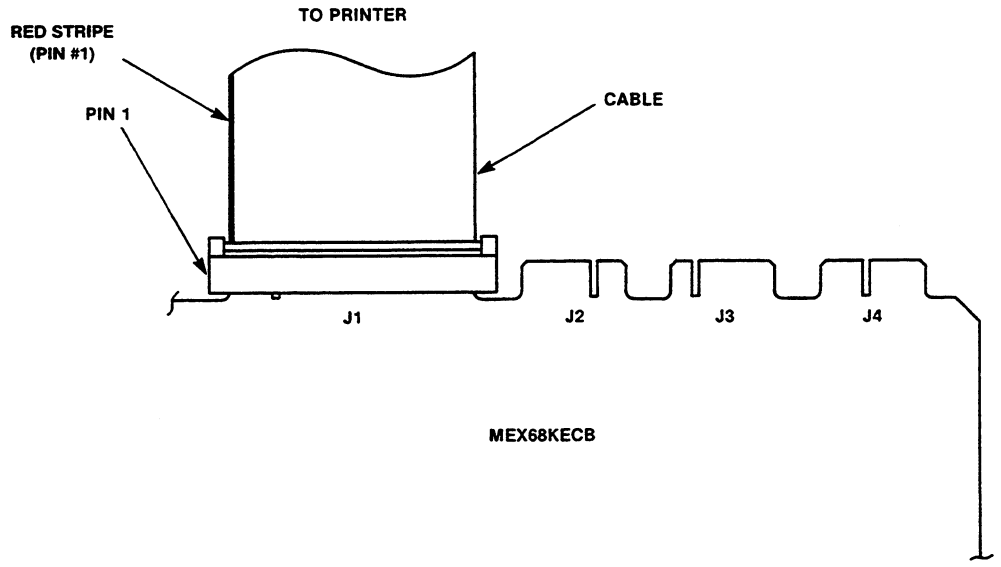


FIGURE 2-11. Printer Cable Connection to MEX68KECB

2.5.2 Host Computer (Modem) Option

A second serial RS-232 port, Port 2 connector J4, is provided to interconnect into a host computer directly or by modem. Preparation similar to serial Port 1 is required — that is, program the baud rate and prepare a cable.

Again referring to Figure 6-4, the modem or host connected to Port 2 must assert CTS (clear to send) before information can be transmitted via Port 2. RTS is asserted by the ECB when power is applied to the board. DTR is asserted as part of the ECB power-up/reset firmware.

2.5.2.1 Selecting Host Baud Rate. As with the terminal baud rate, the host serial port is wired to transmit and receive at the same baud rate. The desired baud rate is selected via a plastic cap jumper positioned on header J9, as shown in Figure 2-12. Also reference Figure 2-4 for the interconnection diagram and Table 2-1 for the selected baud rate.

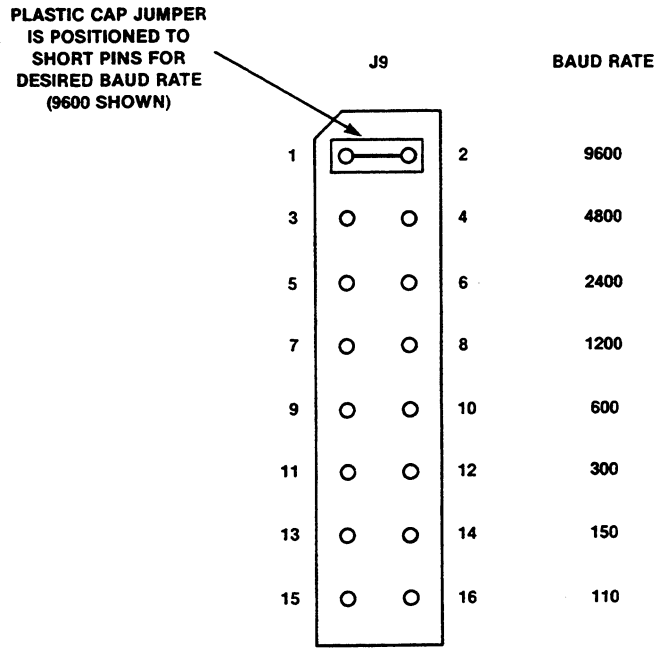


FIGURE 2-12. Host Baud Rate Select Jumper (J9)

Again similar to the terminal port, to provide different baud rates, the connection between TXC and RXC for ACIA2 must be cut (reference Figure 2-4) and individual baud rates connected to each. Perform the following steps to select separate transmit and receive baud rates for the host port:

a. Cut the signal trace located between Pin 1 and Pin 2 of header J7 on the back side of the printed circuit board. BE CAREFUL -- be sure to cut the correct trace; it is approximately 1/8 inch long.

b. The transmit baud rate (TXC) is selected by using the plastic jumper cap on header J9 in accordance with Table 2-1.

c. The receiver baud rate (RXC) is selected by wire-wrapping Pin 2 of header J7 to the desired even numbered pin of header J9. Again, use Table 2-1 to determine the correct pin.

2.5.2.2 Cable Connection. The same cables referenced in paragraph 2.3.1 can be used with the host serial port. As before, the cable can be either purchased or manufactured with the components referenced in Figure 2-7. The card edge connector can also be keyed. Figure 2-13 shows the host computer serial port cable signal line connection.

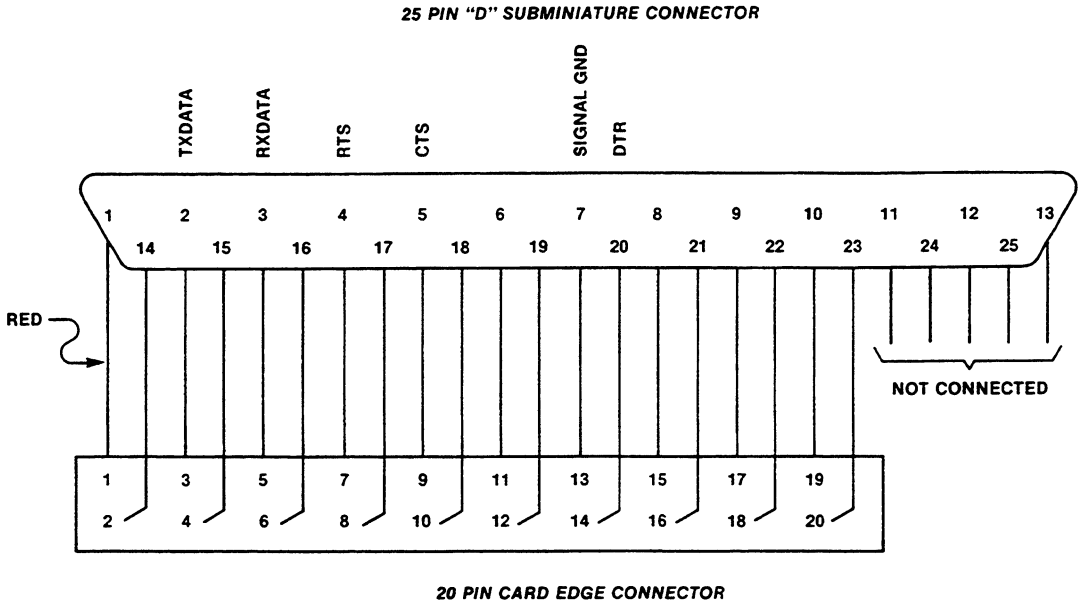


FIGURE 2-13. Host Cable Signal Line Connections

The suitable cable is connected to Port 2 (connector J4), as shown in Figure 2-14, with the other end going to the host computer or modem.

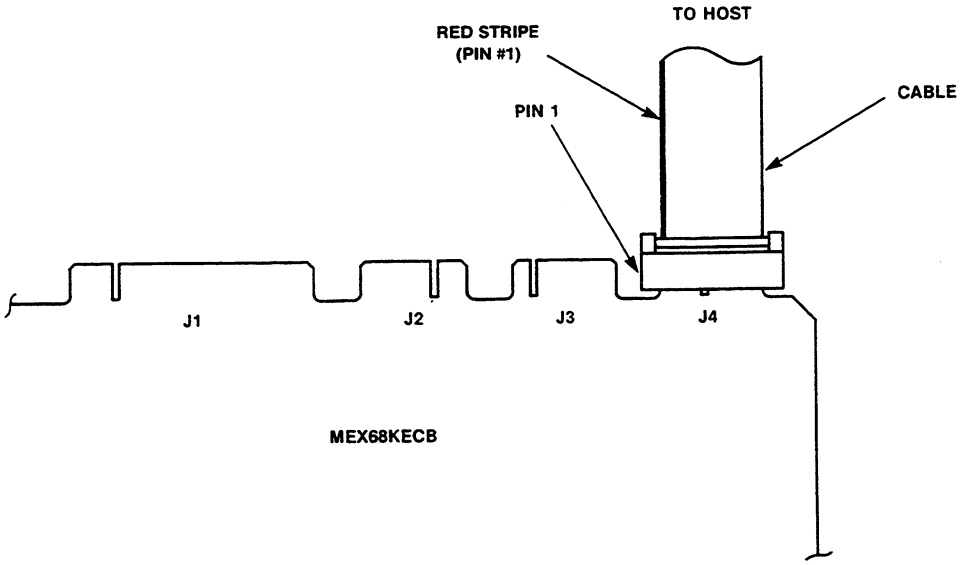
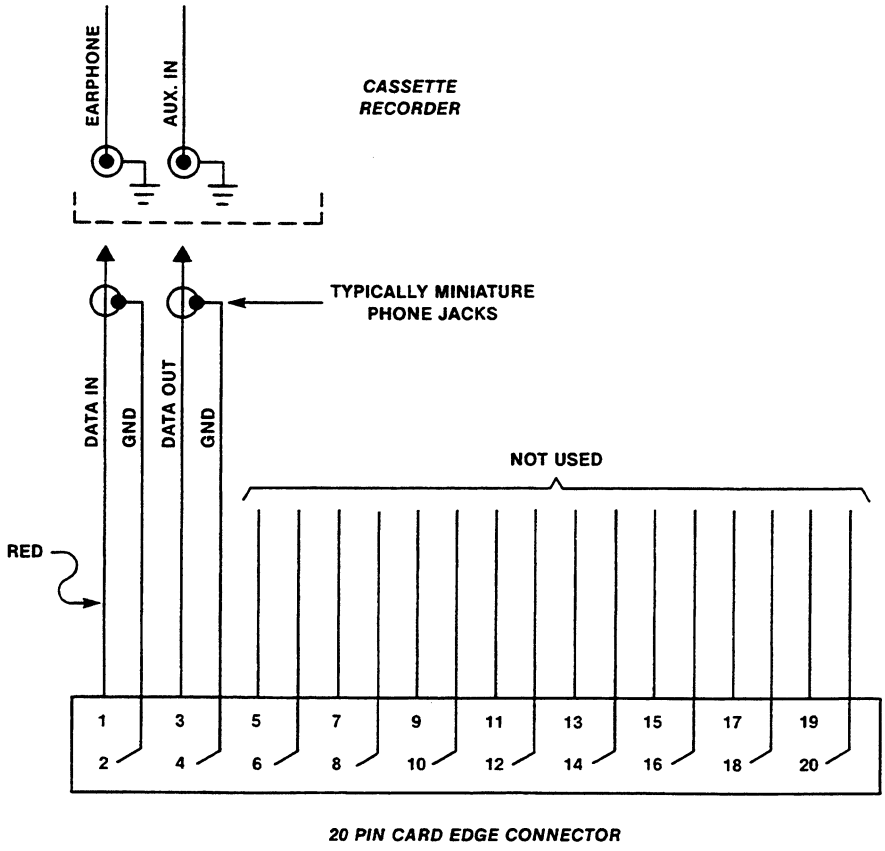


FIGURE 2-14. Host Computer Cable Connection to MEX68KECB

2.5.3 Audio Cassette Option

An audio cassette player can be used for data storage with MEX68KECB. Two signal lines plus ground must be connected to the cassette player from Port 4, connector J2. The user must make a custom cable suitable for the tape player used. Figure 2-15 shows a typical configuration. The educational board requires a 20-pin card edge connector, and a cassette recorder typically uses miniature phone plugs. The "DATA OUT" signal line from the board is connected to the AUXILIARY input to the cassette recorder (or the microphone input if no auxiliary input is available; see Chapter 6 for more details). The "DATA IN" signal line from the board is connected to the EARPHONE output of the recorder. The optional key is located between positions 13 and 15.



NOTE: All even pins are ground.

FIGURE 2-15A. Cassette Recorder Cable Signal Line Connection

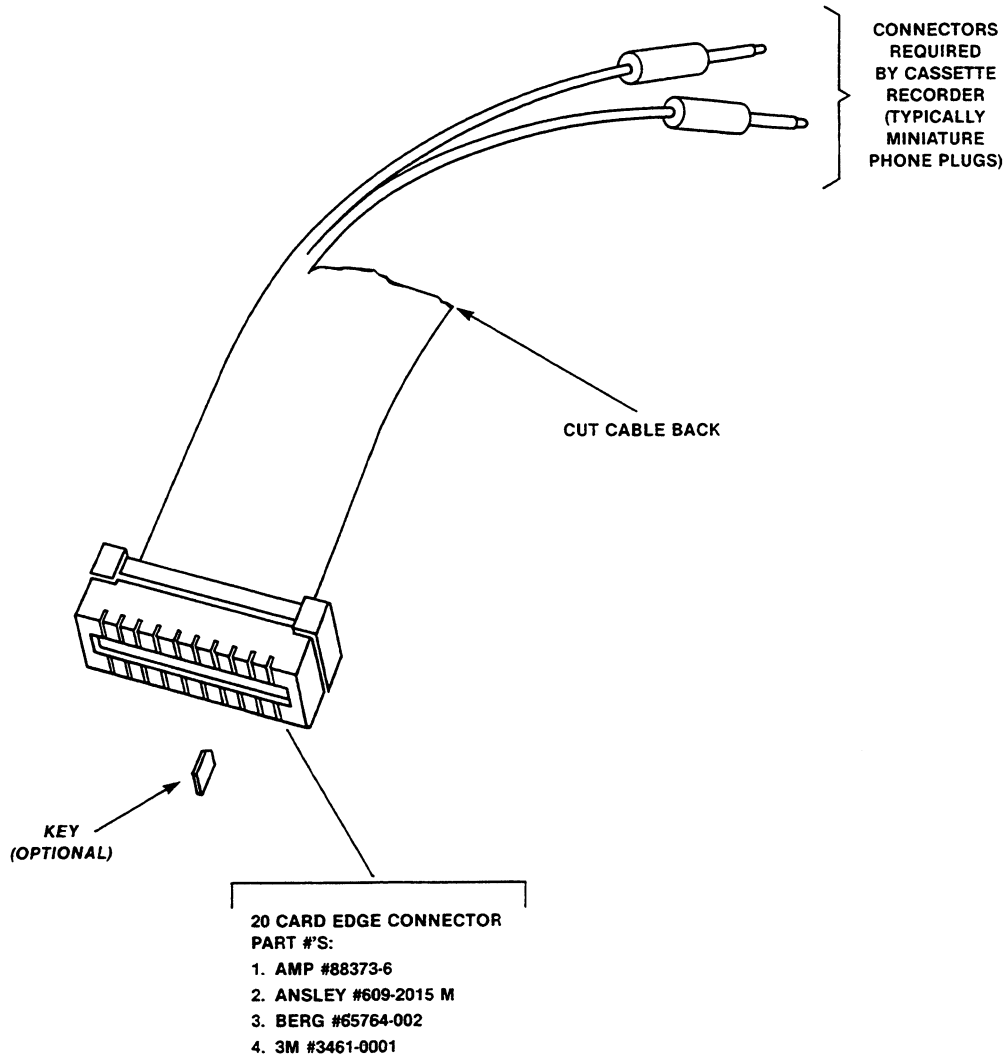


FIGURE 2-15B. Cassette Recorder Cable Detail

CHAPTER 3

USING THE MONITOR/DEBUG FIRMWARE

The MC68000 Educational Computer Board has a resident firmware package that provides a self-contained programming and operating environment. The firmware, aptly named "TUTOR", provides the user with monitor/debug, assembly/disassembly, program entry, and I/O control functions. Chapter 3 is a how-to-use description of the TUTOR package, including user interface and the command structure. Chapter 4 provides a detailed discussion of the assembler/disassembler functions called by the TUTOR firmware.

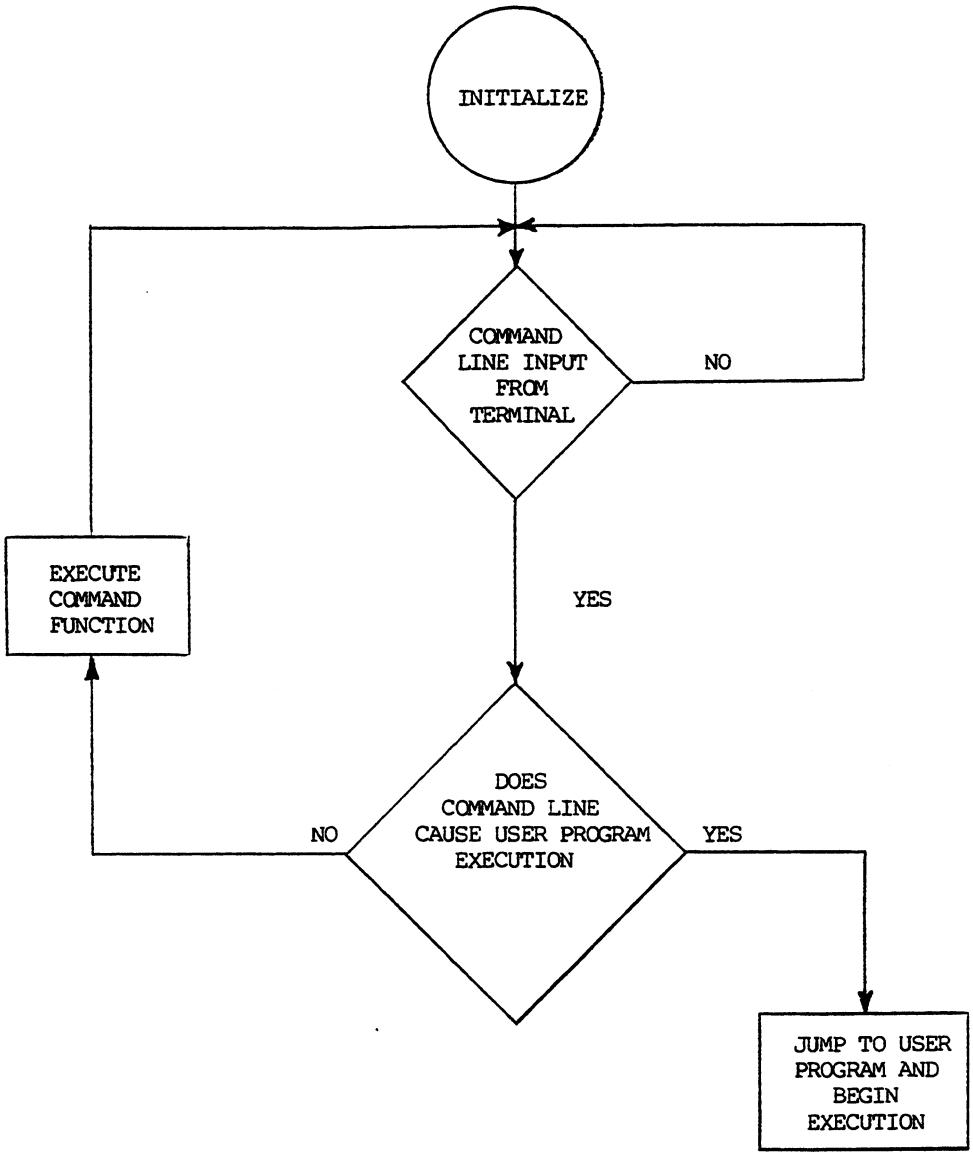


FIGURE 3-1. Flow Diagram of TUTOR Operational Mode

CHAPTER 3

USING THE MONITOR/DEBUG FIRMWARE

3.1 WHAT IS TUTOR?

TUTOR is the resident firmware package for the MC68000 Educational Computer Board. The 16K-byte firmware (stored in two 8Kx8 ROM or EPROM devices) provides a self-contained programming and operating environment. TUTOR interacts with the user through pre-defined commands that are entered via the terminal. The commands fall into four general categories:

a. Commands which allow the user to display or modify memory.

b. Commands which allow the user to display or modify the various internal registers of the MC68000.

c. Commands which allow the user to execute a program under various levels of control.

d. Commands which control access to the various input/output resources on the board.

An additional function called the TRAP 14 handler allows the user program to utilize various routines within TUTOR. The TRAP 14 handler is discussed in Chapter 5.

The operational mode of TUTOR is demonstrated in Figure 3-1. After system initialization, the computer waits for a command line input from the user terminal. When a proper command is entered, the operation continues in one of two basic modes. If the command causes execution of a user program, the TUTOR firmware may or may not be re-entered, depending on the discretion of the user. For the alternate case, the command will be executed under control of the TUTOR firmware, and after command completion, the system returns to a waiting condition. During command execution, additional user input may be required, depending on the command function.

The command format and syntax are similar to other Motorola products based on the MC68000. This is done so that a large relearning effort is not required when the user utilizes these other products.

3.2 OPERATIONAL PROCEDURE

CAUTION

POWER SUPPLIES MUST BE TURNED ON AND OFF IN PROPER SEQUENCE TO AVOID DAMAGE TO THE DYNAMIC RAM DEVICES. FOLLOW THE TURN-ON INSTRUCTIONS TO PREVENT PROBLEMS.

System turn-on and initial operation are described in detail in paragraph 2.4. This information is repeated here for convenience and to prevent possible damage.

3.2.1 System Turn-On

To power-up with individual power supplies:

a. All cables should be connected, making sure ground is connected common to all power supplies.

b. Turn on -12.0 Vdc.

c. Turn on +12.0 Vdc.

d. Turn on +5.0 Vdc.

Alternatively, if a single multivoltage supply is used, then:

e. Be sure all voltages are connected prior to power-up.

f. Turn power ON to the board.

3.2.2 System Initialization

The act of powering up the board will initialize the system. The processor is reset and TUTOR is invoked. After initialization, the terminal will print:

```
TUTOR 1.X >
```

where "X" is the revision number of the software.

NOTE

If this response does not appear, system checks may need to be performed as described in paragraph 2.4. The most common problem is that the terminal and board are not set up for matching baud rates. Also, slower terminals such as T.I. 700 series devices can have special requirements, which are discussed in Appendix B.

Other means can be used to re-initialize the Educational Computer Board firmware. These means are discussed in the following paragraphs.

3.2.2.1 RESET Button. RESET is the black button located on the lower edge of the board. Depressing this button causes all processes to terminate, resets the MC68000 processor and MC68230 PI/T, and restarts the TUTOR firmware. Pressing the RESET button should be the appropriate action if all else fails.

3.2.2.2 ABORT Button. ABORT is the red button located next to the RESET button at the lower edge of the board. The abort function causes an interrupt of the present processing (a level 7 interrupt on the MC68000) and gives control to the TUTOR firmware. This action differs from reset in that no processor register or memory contents are changed, the processor and peripherals are not reset, and TUTOR is not restarted. Also, in response to depressing the ABORT button, the contents of the MC68000 internal registers are displayed.

The abort function is most appropriate when software is being debugged. The user can interrupt the processor without destroying the present state of the system.

3.2.2.3 User Program. The user can return control of the system to the firmware by recalling TUTOR via his program. Instructions can be inserted into the user program to call TUTOR via THE TRAP 14 handler. See Chapter 5.

3.2.3 System Operation

After system initialization or return of control to TUTOR, the terminal will print:

```
TUTOR 1.X >
```

and wait for a response.

The user can call any of the commands supported by the firmware. A standard input routine controls the system while the user types a line of input. Command processing begins only after the line has been entered, followed by a carriage return.

NOTES

1. The user memory is located at addresses $000900-$007FFF. When first learning the system, the user should restrict his activities to this area of the memory map.

2. If a command causes the system to access an unused address (i.e., no memory or peripheral devices are located at that address), a bus trap error will occur. This results in the terminal printing out a trap error message and the contents of all MC68000 registers. Control is returned to the TUTOR monitor. A bus trap error also occurs if the system attempts to write to ROM.

3.3 TERMINAL CONTROL CHARACTERS

Several keys are used as command line edit and control functions. It is best to be familiar with these functions before exercising the system. The functions include:

a. Delete (rubout) key or CTRL H - will delete the last character entered on the terminal.

b. CTRL X - will cancel the entire line.

c. CTRL D - will redisplay the entire line.

d. RETURN (carriage return) - will enter the command line and cause processing to begin.

e. CTRL W - will suspend system output to the terminal. To resume output to the terminal, any other character can be entered.

f. BREAK - will abort commands that do any console I/O and return to the input routine.

For characters requiring the control key (CTRL), the CTRL should be pushed and held down and then the other key (H, X, D, or W) should be pushed.

These control characters are summarized with the command set in Table 3-2.

3.4 COMMAND LINE FORMAT

The command line format is:

TUTOR 1.X > [NO]<command> [<parameters>] [;<options>]

where:

TUTOR 1.X >	Is the prompt from the educational computer generated by TUTOR.
NO	Is the negative form (opposite) of primitive command.
command	Is the primitive command.
parameters	Are separated by spaces and can be of the form <expression> or <address>.
options	Multiple options may be selected.

NOTES

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

   [ ] Enclose optional fields.

   < > Enclose a syntactical variable.

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

2. Fields are separated by one or more spaces used as a delimiter.

The basic command form consists of the primitive command field and the parameters field, although some primitives do not require parameters. The additional command negation and options fields can modify the primitive command.

If an option exists for a command, a semicolon (;) plus <options> field(s) are added to the command. Thus, several extensions can be provided to the user.

3.4.1 Expression as a Parameter

An <expression> can be one or more numeric values separated by the arithmetic operators plus (+) or minus (-). Numbers are assumed hexadecimal except for those preceded by an ampersand (&), which are decimal. In the assembler, numbers are assumed decimal unless preceded by a dollar sign ($).

3.4.2 Address as a Parameter

Many commands use <address> as a parameter. The syntax accepted by TUTOR is the same as that accepted by the assembler, plus a memory indirect mode. Also, contained within TUTOR are eight offset registers designated R0-R7. These registers are software registers only, and are provided for relocatability of code.

3.4.2.1 Address Formats.

FORMAT	EXAMPLE	DESCRIPTION
expression	140	Absolute address (NOTE: offset zero is added)
expression+offset	130+R5	Absolute address plus offset five (not an assembler-accepted syntax)
expression+offset	150+R7	Absolute address (NOTE: offset seven is always zero; not an assembler-accepted syntax)
(A@)	(A5)	Address register indirect
(A@,D@) (A@,A@)	(A6,D4)	Address register indirect with index
expression(A@)	120(A3)	Register indirect with displacement
expression(A@,D@) expression(A@,A@)	110(A2,D1)	Address register indirect with index plus displacement
[expression]	[100]	Memory indirect (not an assembler-accepted syntax)

3.4.2.2. Offset Registers. Eight software registers (not actually hardware configured) are used to modify addresses contained in TUTOR commands. The first seven registers (.R0-.R6) are used as general-purpose offsets, while .R7 (the eighth register) is always zero. The contents of the registers can be displayed by the Offset command (OF), paragraph 3.5.19, and modified by the .RX command, paragraph 3.5.22.

The offset registers are always reset to zero at power-up or by activating the reset button. Thus, if their contents are not changed, the registers will have no effect on the entered address.

Unless another offset is entered, each command that expects an address parameter automatically adds offset R0 to the entered address — that is, if R0 = 1000, then the following commands are the same:

```
BR   10
BR   10+R0
```

The physical address for each of these commands is 1010.

Offset R0 is automatically added to the offset registers any time they are modified. The only exception to this is when another offset register is specifically added. Offset registers are set to zero by adding R7 (always zero) to zero.

Example:

```
.R1   8        R1 = 8              Offset R0 is zero, R1 is set to 8
.R0   100      R0 = 100
.R0   200      R0 = 200+100=300    Offset R0 added
.R3   100+R1   R3 = 100+8=108      Offset R0 not added
.R0   0+R7     R0 = 0              R0 set to zero
```

3.4.3 Command Echo Back

Most commands that require parameters display back to the user the information entered, but in a physical format so that the user sees the expression or address results. Some error checking is done — for example, if an address will cause an obvious error, the message INVALID ADDRESS=XXXXXXXX will result on the terminal connected to serial port 1. Refer to Table 3-3 for the error messages and other messages used in TUTOR.

3.5. TUTOR COMMAND SET

Table 3-1 lists the TUTOR commands by type.

TABLE 3-1. TUTOR Commands

COMMAND MNEMONIC	DESCRIPTION	PAGE
MD	Memory Display	3-24
MM, M	Memory Modify	3-25
MS	Memory Set	3-28
.A0 - .A7	Display/Set Address Register	3-35
.D0 - .D7	Display/Set Data Register	3-35
.PC	Display/Set Program Counter	3-35
.SR	Display/Set Status Register	3-35
.SS	Display/Set Supervisor Stack Pointer	3-35
.US	Display/Set User Stack Pointer	3-35
DF	Display Formatted Registers	3-17
OF	Display Offsets	3-31
.R0 - .R6	Display/Set Relative Offset Register	3-31
BF	Block of Memory Fill	3-11
BM	Block of Memory Move	3-12
BT	Block of Memory Test	3-15
BS	Block of Memory Search	3-14
DC	Data Conversion	3-16
BR	Breakpoint Set	3-13
NOBR	Breakpoint Remove	3-29
GO, G	Go	3-20
GT	Go Until Breakpoint	3-21
GD	Go Direct	3-19
TR, T	Trace	3-38
TT	Temporary Breakpoint Trace	3-39
PA	Printer Attach	3-32
NOPA	Reset Printer Attach	3-30
PF	Port Format	3-33
TM	Transparent Mode	3-36
*	Send Message to Port 2	—
HE	Help	3-22
DU	Dump Memory	3-18
LO	Load	3-23
VE	Verify	3-40

Each of the individual commands is described in the following pages. Figure 3-2 shows the general format of the description.

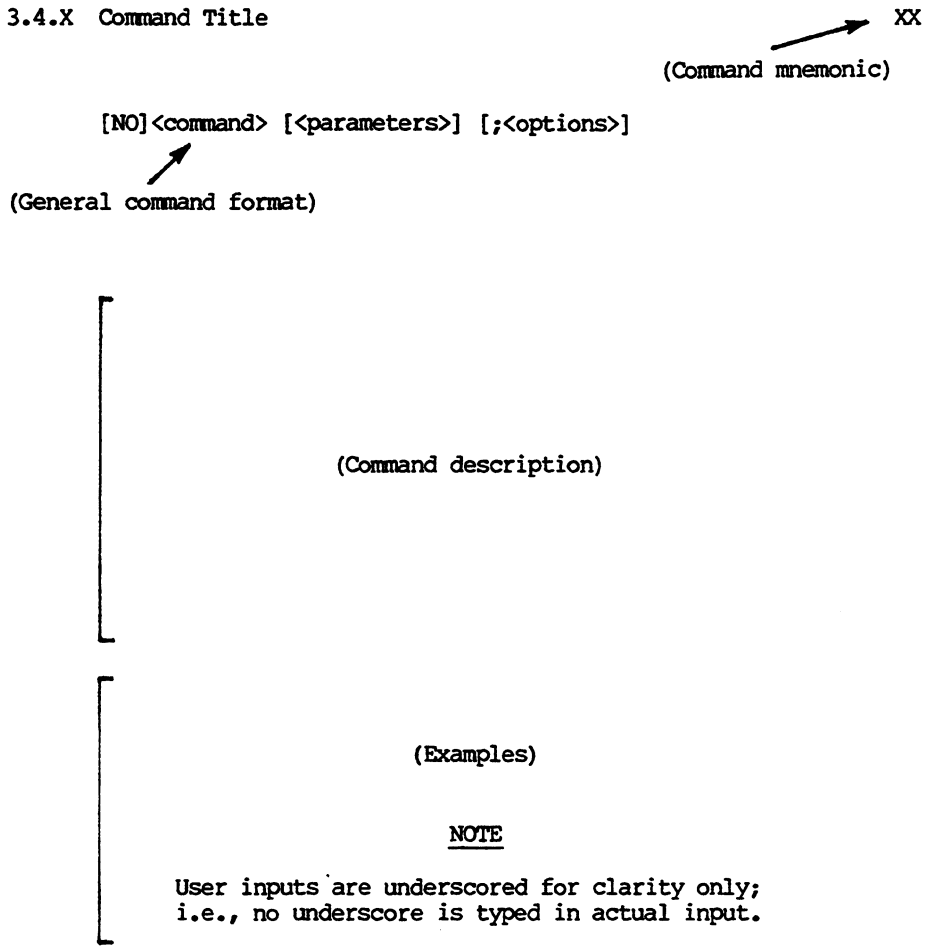


FIGURE 3-2. Command Description Format

3.5.1 Block of Memory Fill BF

BF <address1> <address2> <word>

The BF command fills memory starting with the word boundary (even address) <address1> through <address2>. Both <address1> and <address2> must be even addresses. This command only fills with a word-size (two-byte) data pattern, as specified in hex, octal, decimal, or binary digits. If an entire word size data pattern is not entered, the pattern is right justified and leading zeros are inserted.

EXAMPLE

```
TUTOR  1.X > MD 2004
002004    17 39 2A 33 BF FF 00 9E  41 42 55 CD C4 44 00 98  .9*3?...ABUMDD..

TUTOR  1.X > BF 2004 200A 475A
PHYSICAL ADDRESS=00002004 0000200A

TUTOR  1.X > MD 2004
002004    47 5A 47 5A 47 5A 47 5A  41 42 55 CD C4 44 00 98  GZGZGZGZABUMDD..

TUTOR  1.X > BF 2004 2012 7
PHYSICAL ADDRESS=00002004 00002012

TUTOR  1.X > MD 2004
002004    00 07 00 07 00 07 00 07  00 07 00 07 00 07 00 07  ................

TUTOR  1.X >
```

3.5.2 Block Move BM

BM <address1> <address2> <address3>

The BM command is used to move (duplicate) blocks of memory from one area to another.

<address1> = beginning address of source memory block
<address2> = ending address of source memory block
<address3> = beginning address of destination memory block

EXAMPLE 1

```
TUTOR  1.X > BM 1800 1900 1860          The entire block from $1800 through $1900
PHYSICAL ADDRESS=00001800 00001900      is duplicated, starting at $1860.
PHYSICAL ADDRESS=00001860

TUTOR  1.X >
```

EXAMPLE 2

```
TUTOR  1.X > MD 1800 10
001800    00 11 22 33 44 55 66 77  88 99 AA BB CC DD EE FF  .."3DUfw..*;L]n.

TUTOR  1.X > BM 1806 1809 1804
PHYSICAL ADDRESS=00001806 00001809
PHYSICAL ADDRESS=00001804

TUTOR  1.X > MD 1800 10
001800    00 11 22 33 66 77 88 99  88 99 AA BB CC DD EE FF  .."3fw....*;L]n.

TUTOR  1.X >
```

3.5.3 Breakpoint BR

BR [<address>[;<count>]]...

When encountered, a breakpoint causes program execution to stop and control to be transferred to TUTOR. The BR <address> command sets one or more addresses into the breakpoint address table. This table can hold up to eight breakpoint addresses. Multiple breakpoints (up to eight) may be specified with one call of the breakpoint command. Addresses should be on even word boundaries. The range of <count> is a 32-bit integer.

The breakpoints are inserted into the user program when execution is called via a GO or GT command. The illegal instruction $4AFB is inserted at the addresses specified by the table. During execution of the program, a breakpoint occurs whenever this instruction is encountered. If program control is lost, control can be regained via the RESET or the ABORT button. ABORT is preferred because use of the RESET function may leave breakpoints ($4AFB) in the user program, whereas ABORT will recover properly.

The NOBR command is used to eliminate all breakpoints from the breakpoint table.

While executing a Trace command, the breakpoint addresses are monitored (i.e., the illegal instruction $4AFB is not placed in memory).

COMMAND FORMAT	DESCRIPTION
TUTOR 1.X > BR	Display all breakpoints.
TUTOR 1.X > BR address	Set a breakpoint.
TUTOR 1.X > BR address;count	Set a breakpoint with a count. Count is decremented each time the breakpoint is encountered until count = 0. Execution stops as soon as count is decremented to zero. Thereafter, execution will stop each time the breakpoint is reached.

See also: GT, NOBR, TT

EXAMPLE

```
TUTOR 1.X > .R4 4000
TUTOR 1.X > BR 1010 2000;5 2040 4000
BREAKPOINTS
001010     001010
002000     002000;5
002040     002040
000000+R4  004000

TUTOR 1.X > NOBR 1010 2040
BREAKPOINTS
002000     002000;5
000000+R4  004000

TUTOR 1.X > NOBR
BREAKPOINTS
TUTOR 1.X >
```

```
BS <address1> <address2> 'literal string'
BS <address1> <address2> <data> [<mask>] [;<option>]
```

The BS command has two modes: 1) literal string search, and 2) data search. Both modes scan memory beginning at <address1> through <address2>, looking for a match.

The literal string mode is initiated if a single quote (') follows <address2>. If a single quote does not follow <address2>, data search mode is assumed. In the data search mode, the optional mask, if used, is ANDed to data. The default mask is all one's. The options supported are:

```
;B   byte
;W   word
;L   long word
```

The default is byte.

In both modes of the BS command, if the search finds matching data, the data and the address(es) are displayed. If the search is in data search mode with a mask, and data is found that matches the data after the mask is ANDed, the data from memory before applying the AND mask is displayed.

EXAMPLE

```
TUTOR  1.X > MD 1FF0 15
001FF0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF   ................
002000  43 43 45 45 00 00 00 00  00 00 00 00 00 00 00 00   CCEE............

TUTOR  1.X > BS 1FF0 200F 'CC'
PHYSICAL ADDRESS=00001FF0 0000200F
002000 'CC'

TUTOR  1.X > BS 1FF0 200F 34 ;W
PHYSICAL ADDRESS=00001FF0 0000200F

TUTOR  1.X > BS 1FF0 200F 03 0F
PHYSICAL ADDRESS=00001FF0 0000200F
002000 43
002001 43

TUTOR  1.X > BS 1000 7FFE 4AFB;W
PHYSICAL ADDRESS=00001000 00007FFE
001000 4AFB

TUTOR  1.X >
```

COMMENT

Successful search for literal string 'CC'.

Unsuccessful search for word length data (with default mask).

Successful search for byte length data, with four most significant bits masked.

Successful search for "leftover" breakpoints.

3.5.5 Block of Memory Test BT

BT <address1> <address2>

The BT command is a destructive test of a block of memory beginning at <address1> through <address2> inclusive (both at word boundaries = even addresses). If this test runs to completion without detecting errors, the memory tested will be set to all zeros.

This command may take several seconds to test large blocks of memory.

If memory problems are found, a message is displayed indicating the address, the data stored, and the data read of the failing memory.

EXAMPLE

```
TUTOR  1.X > BT 5000 5FFE
PHYSICAL ADDRESS=00005000 00005FFE

TUTOR  1.X > BT 6000 6040
FAILED AT 6000 WROTE=FFFE READ=FF00
```

DC <expression>

The DC command is used to convert an expression into hexadecimal and decimal. The expression may be entered in hexadecimal, decimal, or mixed format; output will be shown both ways. Default input format is hexadecimal.

Offsets may be used with the DC command. R0 is used if the offset is not specified.

This command is useful in calculating displacements such as destination of relative branch instructions or program counter relative addressing modes.

COMMAND FORMAT	DESCRIPTION
TUTOR 1.X > DC $data	Convert hexadecimal data into hexadecimal and decimal.
TUTOR 1.X > DC &data	Convert decimal data into hexadecimal and decimal.

EXAMPLE

```
TUTOR 1.X > DC &120
$78=&120

TUTOR 1.X > DC &15+$4-$13
$0=&0

TUTOR 1.X > DC -1000
$FFFFF000=-$1000=-&4096

TUTOR 1.X >

TUTOR 1.X > .R0 1000

TUTOR 1.X > OF
R0=00001000 R1=00000000 R2=00000000 R3=00000000
R4=00000000 R5=00000000 R6=00000000 R7=00000000

TUTOR 1.X > DC 10+10+30
$1050=&4176

TUTOR 1.X > DC 10+10+30+R7
$50=&80
```

3.5.7 Display Formatted Registers DF

DF

The DF command is used to display the MC68000 processor registers. The trace display will be displayed whenever the debugger gains control of the program execution -- i.e., at breakpoints and when tracing.

Note that any single register can be displayed with the .Ax, .Dx, etc., commands.

See also: .Rx (contains forms .Ax, .Dx, etc.)

EXAMPLE

```
TUTOR  1.X > DF
PC=00001000 SR=2700=.S7..... US=00002000 SS=00000F00
D0=FFFFFFFF 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
```

NOTE

Any time the registers are displayed, a disassembled line of code is also displayed (see Chapter 4 for format). The instruction located at the address pointed to by the program counter (example, $001000) is disassembled and shown. This is useful for program debugging.

3.5.8 Dump Memory (in S-Record Format) DU

DU[<port number>] <address1> <address2> [<text..>]

The DU command outputs S-records of memory contents from <address1> through <address2> to <port number>. Any optional text is output as part of a header record.

S-records are a standard data format used in transmitting and receiving programs and data. Appendix A discusses them in more detail. As part of the memory dump, any text goes out as an S0 or header record, and transmission ends with an S9 end-of-file record.

The DU command has several options, including dump to Ports 1, 2, 3, and 4. Also a default case of DU dumps to Port 1. The variations include:

COMMAND	PORT #	DESTINATION
DU	Port 1	Terminal
DU1	Port 1	Terminal
DU2	Port 2	Host (modem)
DU3	Port 3	Printer
DU4	Port 4	Audio Cassette

This command does not send control characters to start or stop I/O devices. The offset contained in offset register R0 is added to the starting and ending memory addresses.

See also: LO, VE

EXAMPLE

```
TUTOR  1.X > DU 8800 880F TUTOR 1.X
PHYSICAL ADDRESS=00008800 0000880F
S00C00005455544F5220312E587E
S11388006654BBCE6704610013521E3C004447F813
S9030000FC

TUTOR  1.X > DU1 8800 880F
PHYSICAL ADDRESS=00008800 0000880F
S0030000FC
S11388006654BBCE6704610013521E3C004447F813
S9030000FC

TUTOR  1.X > DU4 7000 70FF                    S-records are dumped to
PHYSICAL ADDRESS=00007000 000070FF            Port 4 (tape recorder)

TUTOR  1.X >
```

3.5.9 Go Direct Execute Program GD

GD [<address>]

The GD command is similar to the GO command, except that GD does not set breakpoints, nor does it start by tracing one instruction. The GD command starts the target program at the location given as address without changing any of the exception vectors (locations $0 through $3FF). If address is not specified, the GD command starts the target program at the address in the PC.

See also: GO, GT

EXAMPLE

```
TUTOR 1.X > GD 2000
PHYSICAL ADDRESS=00002000
```

GO [<address>]
G [<address>]

The GO command causes the target program to execute (free run in real time) until:

a. the target program encounters a breakpoint,

b. abnormal program sequence that causes exception processing (e.g., divide by zero), or

c. operator intervention through the RESET or ABORT pushbutton switch.

NOTE

If breakpoints with count are encountered, real time is not achieved. The breakpoint will not stop processing until Count is diminished to zero, but processing overhead is required.

The GO sequence starts by tracing one instruction, setting any breakpoints, and then free running.

COMMAND FORMAT	DESCRIPTION
TUTOR 1.X > GO	Begin execution at address in PC.
TUTOR 1.X > GO address	Set PC = address and begin execution at that address.

See also: BR, DF, GD, GT, TR, TT

EXAMPLE

```
TUTOR 1.X > MM 2000;L
002000    00002F00 ?3000.

TUTOR 1.X > GO [2000]          Address is memory indirect.
PHYSICAL ADDRESS=00003000
```

3.5.11 Go Until Breakpoint GT

GT <breakpoint address>

The GT command performs the following:

1. sets a temporary breakpoint,
2. sets breakpoints entered by the BR command,
3. sets target program registers as displayed by the DF command,
4. causes the target program to execute from the PC address (free run in real time).

When any breakpoint is encountered, the temporary breakpoint is reset.

If the breakpoint address is in the breakpoint table, the message ERROR and the breakpoint table are displayed.

See also: BR, DF, GD, GO, TR, TT

EXAMPLE

```
TUTOR  1.X > BR 2010 3000

BREAKPOINTS
002010    002010
003000    003000

TUTOR  1.X > DF
PC=00002000 SR=A704=TS7..Z.. US=FFFFFFFF SS=000007BC
D0=FFFF1230 D1=00101200 D2=FED01210 D3=00000000
D4=FFFF0031 D5=FFFFFF2C D6=00000002 D7=00000000
A0=00010040 A1=FFFFFFFF A2=00000454 A7=0000054E
A4=00009F38 A5=0000053A A6=0000053A A7=000007BC
--------------------002000    0C000030            CMP.B   #48,D0

TUTOR  1.X > GT 2006
PHYSICAL ADDRESS=00002006
PHYSICAL ADDRESS=00002000

TUTOR  1.X > GT 2010                  Temporary breakpoint address $2010 is
PHYSICAL ADDRESS=00002010             already in breakpoint table.
ERROR
002010    002010
003000    003000
```

3.5.12 Help HE

HE

The HE command gives the user information as to available commands.

EXAMPLE

```
TUTOR  1.X > HE
.PC .SR .US .SS
.D0 .D1 .D2 .D3. D4 .D5 .D6 .D7
.A0 .A1 .A2 .A3 .A4 .A5 .A6 .A7
.R0 .R1 .R2 .R3 .R4 .R5 .R6

BF    BM    BR    NOBR  BS    BT    DC    DF
DU    G     GD    GO    GT    HE    LO    M
MD    MM    MS    OF    PA    NOPA  PF    T
TM    TR    TT    VE
```

3.5.13 Load (in S-Record Format) LO

LO[<port number>] [;<options>][=text]

The LO command moves object data in S-record format from an external device (Port 1, Port 2, or Port 4) to memory. Appendix A discusses S-records in more detail.

The command has the basic forms:

COMMAND	PORT #	SOURCE
LO1	Port 1	Terminal (i.e., terminal with tape drive)
LO	Port 2	Host (modem) - default port
LO2	Port 2	Host (modem)
LO4	Port 4	Audio cassette

The options include:

;-C Ignore the S-record checksum while loading. A checksum is contained in each S-record. If this option is not selected, the received checksum is compared with the calculated checksum. If they do not agree, the message CHKSUM= and the calculated checksum are sent to Port 1. The data is not loaded into memory if the checksums do not agree.

;X Echo data read from the source port onto the Port 1 terminal.

The optional [=text] is used only with Port 2. The text following the "=" is sent to Port 2. In this manner, a message can be sent to Port 2 to start a download, as an example.

A timeout feature is available for Port 2. If the host connected to Port 2 does not respond within approximately 10 seconds, the message TIMEOUT is sent to Port 1 and the LO command is aborted.

Several characteristics of this command are important to note:

a. The offset contained within register R0 is added to the addresses for the data contained within the S-record.

b. Any record not containing an S0, S1, S2, S8, or S9 string is ignored.

c. If an error occurs, causing the system to print out an error message, one or more lines sent during the error message may be ignored. The system cannot be printing and processing incoming data at the same time. To prevent the loss of information, the ECB can send characters to the host to stop and start the transfer of the S-records. Paragraph 4.5.2 describes this feature.

See also: DU, OF, PF, VE

EXAMPLES	COMMENT
TUTOR 1.X > LO ;X=COPY FILE.MX,#CN	Download from Port 2 with echo option.
TUTOR 1.X > LO;X-C=COPY FILE.MX,#	Download from Port 2 without verifying checksum.

3.5.14 Memory Display MD

MD[<port number>] <address> [<count>][;<options>]

The MD command is used to display a section of memory beginning at <address> and displaying the number of bytes given as <count>. Two modes are used for the data display — that is, hex data (with equivalent ASCII) and disassembled form.

The command has the basic forms:

COMMAND	PORT #	DESTINATION
MD	Port 1	Terminal - default Port
MD1	Port 1	Terminal
MD2	Port 2	Host (modem)
MD3	Port 3	Printer

For <count>, the default condition is 16 bytes when no option is specified and one instruction or directive when the disassemble option is used.

Only one option is specified; therefore, only two output forms are used:

1. No option specified — will display the data in hex and in equivalent ASCII. Data is always displayed in groups of 16 bytes. If the count is not on a 16-byte boundary, the next highest group of 16 will be displayed, unless the count is on a 16-byte boundary plus one, in which case the next highest group of 16 will not be displayed.

   Once the MD command is entered, it will continue with the next 16 lines of output each time a carriage return (CR) is entered. Any other command exits MD and enters the new command.

2. ;DI — invokes the disassembler function. The data is displayed in the disassembled format described in Chapter 4. Included in the instruction display is the address of the opcode, hexadecimal instruction code, instruction mnemonic, and operands. The MD command will display all instructions whose op code is contained within the byte count.

See also: MM, MS

EXAMPLE

```
TUTOR  1.X > MD 1000 12
001000    0C 00 00 30 6D 28 0C 00  00 39 6E 10 02 80 00 00  ...0m(...9n.....
001010    00 0F 11 C0 10 38 1E 3C  00 E4 4E 4E 0C 00 00 41  ...@.8.<.dNN...A

TUTOR  1.X > MD 1000 12 ;DI
001000    0C000030              CMP.B   #48,D0
001004    6D28                  BLT.S   $00102E
001006    0C000039              CMP.B   #57,D0
00100A    6E10                  BGT.S   $00101C
00100C    02800000000F          AND.L   #15,D0
```

```
MM <address> [;<options>]
M <address> [;<options>]
```

The MM command is used to display memory and, as required, modify data or enter new data. The command has two basic forms:

a. Hexadecimal format - The standard form of the MM command displays the address and data at that location. The size option (byte, word, and long word) controls the number of bytes displayed for each address:

OPTION	DESCRIPTION
- (default)	Displays one byte
;W	Displays one word (2 bytes)
;L	Displays one long word (4 bytes)
;O	Displays one byte; accesses only odd address bytes
;V	Displays one byte; accesses only even address bytes
;N	Do not verify; do not read data stored

NOTE: If multiple options are desired, a semicolon (;) must precede each option.

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

[<data>](cr)	Update location and sequence forward.
[<data>]^(cr)	Update location and sequence backward.
[<data>]=(cr)	Update location and reopen same location.
[<data>].(cr)	Update location and terminate.

See also: MD, MS

EXAMPLES

```
TUTOR  1.X > MM 2000;W
002000    2200 ?FFFF
002002    2A07 ?DDDD
002004    60FA ?EEEE^
002002    DDDD ?
002004    EEEE ?
002006    5FFF ?AAAA=
002006    AAAA ?.

TUTOR  1.X > MM 4000;W;N
004000      ?555
004002      ?34.
```

b. ;DI - This option invokes the disassembler/assembler function. The address entered should be the starting address for an instruction (op code) word. The instruction will then be displayed in disassembled form. The disassembled format is described in Chapter 4.

The displayed instruction is followed by a question mark (?) that indicates a new source line may be entered. If a new line is entered, the instruction is immediately assembled, stored, and displayed. To enter a new line, the following format is used:

? <sp> <operation field> <sp> <operand field>(cr)

where:

sp	Is required because no labels are allowed and the format matches the resident assembler.
operation field	Is the MC68000 mnemonic or DC.W directive.
sp	Is a required delimiter.
operand field	Is normally source and destination fields.
cr	Enters new instruction.

The format is discussed in detail in Chapter 4.

Upon entry of the carriage return, the new instruction will overwrite the old instruction and enter the new one. To exit the command, a period (.) is entered immediately after the question mark, followed by a carriage return.

? . cr

NOTE

When inserting new instructions or modifying existing code, the assembler may overwrite the following code. Care must be taken by the programmer to take this factor into account. When moving code, be aware that address vectors may change.

If an error is found in the new instruction, the new line is redisplayed with an "X" immediately under the field suspected of causing a problem in the assembler. The "X" is followed by a question mark to allow re-entry of the corrected souce line.

EXAMPLES

1. TUTOR 1.X > MM 3000;DI
003000 5555 SUBQ.W #2,(A5) ? MOVE.L A0,A1

The assembler overwrites line 3000 and displays:

003000 2248 MOVE.L A0,A1
003002 1211 MOVE.B (A1),D1 ?

2. TUTOR 1.X > MM 3000;DI
```
003000   2248                  MOVE.L   A0,A1 ?
003002   6600E384              BNE.L    $001388 ? MOVE.L A0A1
```

The assembler overwrites line 3002 and displays:

```
003002                         MOVE.L  A0A1
                                       X?
```

An error was found with the operands. The corrected line can now be entered.

3.5.16 Memory Set MS

MS <address> <data...>

The MS command alters memory by setting data into the address specified. The data can take the form of ASCII string or hexadecimal data. Several strings can be entered; however, size is limited to eight characters.

COMMAND FORMAT	DESCRIPTION
TUTOR 1.X > MS 2000 'ABC'	Set memory to ASCII string.
TUTOR 1.X > MS 2003 4445	Set memory to hexadecimal data.
TUTOR 1.X > MS 2005 12345678 12	Size can be up to 8 characters.

See also: MD, MM

EXAMPLE

```
TUTOR 1.X > MD 2000
002000    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................

TUTOR 1.X > MS 2000 'ABC'

TUTOR 1.X > MS 2006 123 123456

TUTOR 1.X > MD 2000
002000    41 42 43 00 00 00 01 23  12 34 56 00 00 00 00 00   ABC....#.4V.....
```

3.5.17 Remove Breakpoint NOBR

NOBR [<address> <address>....]

The NOBR command is used to remove one or more breakpoints from the internal breakpoint table, and functions as the inverse of the BR command.

COMMAND FORMAT	DESCRIPTION
TUTOR 1.X > NOBR	Clear all breakpoints.
TUTOR 1.X > NOBR address	Clear a specific breakpoint.

See also: BR, GT, TT

EXAMPLE

```
TUTOR  1.X > .R3 3000

TUTOR  1.X > OF
R0=00000000 R1=00000000 R2=00000000 R3=00003000
R4=00000000 R5=00000000 R6=00000000 R7=00000000

TUTOR  1.X > BR 2000;5 2030 3000;6 3060

BREAKPOINTS
002000      002000;5
002030      002030
000000+R3   003000;6
000060+R3   003060

TUTOR  1.X > NOBR 3000

BREAKPOINTS
002000     002000;5
002030     002030
000060+R3 003060

TUTOR  1.X > NOBR

BREAKPOINTS

TUTOR  1.X >
```

3.5.18 Reset Printer Attach NOPA

NOPA

The NOPA command allows the user to detach the line printer from the Port 1 terminal.

See also: PA

3.5.19 Offset OF

OF

The OF command displays the offsets contained within registers R0-R7. To help the user with relocatability and position-independent code, seven general-purpose offsets (.R0-.R6) are provided. Offset .R7 is always zero, which provides a convenient way of zeroing other offsets or entering an address without an offset. If no value is assigned to one of the general-purpose offsets, it will have the default value of zero.

Unless another offset is entered, each command that expects an address parameter automatically adds offset R0 to the entered address — that is, if R0 = 1000, the following commands are the same:

```
BR 10
BR 10+R0
```

It should also be noted when setting offsets, R0 is added to the expression being entered into the register. To zero a register, use the form:

```
.RX  0+R7          (X = desired register)
```

The form for setting individual registers is given in the Individual Register Display/Change (.Rx) command.

See also: .Rx

EXAMPLE

```
TUTOR  1.X > .R1 1000
TUTOR  1.X > .R3 3300
TUTOR  1.X > .R5 0+R7

TUTOR  1.X > OF
R0=00000000 R1=00001000 R2=00000000 R3=00003300
R4=00000000 R5=00000000 R6=00000000 R7=00000000

TUTOR  1.X > BR D08 1056

BREAKPOINTS
000D08    000D08
000056+R1 001056

TUTOR  1.X > .R0 2000

TUTOR  1.X > BR 10

BREAKPOINTS
000D08    000D08
000056+R1 001056
000010+R0 002010

TUTOR  1.X > MM 1000+R7
000000+R1 0C ?.

TUTOR  1.X >
```

COMMENT

```
Set offset R1.
Set offset R3.
Reset offset R5.

Set offset R0.

Offset R0 is added to the breakpoint
address. Absolute addresses are on
the right. Addresses relative to
the appropriate offset are displayed
on the left. The appropriate offset
is the nearest offset that is less
than or equal to the absolute address.

To access address $1000 with no offset,
offset R7 (always 0) must be added;
otherwise, offset R0 will automatically
be added.
```

3.5.20 Printer Attach PA

PA

The PA command allows the user to attach the line printer so that information sent to the Port 1 terminal will also be printed. (The printer is physically attached to parallel Port 3 of the board. See the initial setup instructions.)

The printer can also be called by the Dump (DU3) and Memory Display (MD3) commands.

If the printer is deselected or not ready, the message PRINTER NOT READY will be sent to Port 1; TUTOR will wait until the printer is ready or the BREAK key is pushed.

See also: NOPA, DU, MD

3.5.21 Port Format PF

PF[<port number>]

The PF command allows the user to display or assign the characteristics of serial I/O Port 1 and Port 2. Each of these ports may be individually programmed for stop bits, character nulls, and carriage return nulls. The baud rates are selected via jumpers (see Chapter 2).

COMMAND FORMAT	DESCRIPTION
TUTOR 1.X > PF	Display both Port 1 and Port 2 formats.
TUTOR 1.X > PF1	Change Port 1 format.
TUTOR 1.X > PF2	Change Port 2 format.

Parameters include:

a. FORMAT - This two-character (8-bit) parameter determines the number of stop bits after each byte. Transmission used within TUTOR is 8 bits/byte and restricted to one or two stop bits/bytes. Therefore, for stop bits denoted by FORMAT=, entering:

   15 causes 1 stop bit (default)
   11 causes 2 stop bits

   NOTE: This command alters the control register on the MC6850 ACIA. For more information, see the I/O Chapter 6 and the MC6850 data sheet.

b. CHAR NULL - This parameter is the number of nulls sent after each character (default = 00).

c. CR NULL - This parameter is the number of nulls sent after each carriage return/line feed (line of data). (Default = 00). Hard copy terminals usually require four nulls.

d. OPTIONS - This is the address in RAM where the 6-byte options variable is located. The first and second bytes represent the transfer on and transfer off bytes, respectively, which are used to stop and start the transfer of S-records. (Refer to paragraph 4.5.2.) The third and fourth bytes are used with low baud rate or mechanical terminals to control the display. These bytes are discussed in Appendix B. The last two bytes contain the trailing and exit characters used in the transparent mode of operation (paragraph 3.5.23). All bytes are set to their initial (power up) values when the RESET button is pressed.

EXAMPLES

```
TUTOR 1.X > PF

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

```
TUTOR  1.X > PF1
FORMAT=   15?11
CHAR NULL=00?
C/R  NULL=00?

TUTOR  1.X > PF2
FORMAT=   15?
CHAR NULL=00?3
C/R  NULL=00?8

TUTOR  1.X >
```

NOTE

TI 700 series terminals should have the following port characteristics:

BAUD RATE	FORMAT	CHAR NULL	C/R NULL
110	11	0	1
150	15	0	1
300	15	0	4
1200	15	3	17
2400	15	7	2F

See Appendix B.

3.5.22 Individual Register Display/Change .Rx

.A0, .A1, .A2, .A3, .A4, .A5, .A6, .A7
.D0, .D1, .D2, .D3, .D4, .D5, .D6, .D7
.PC, .SR, .SS, .US

The .Rx commands allow the user to display or modify individual registers using the format: .<register> [<expression>]. Commands with a leading period and the registers displayed/altered by these commands are:

.A0 - .A7	address register
.D0 - .D7	data register
.R0 - .R6	relative offset register (software register)
.PC	program counter
.SR	status register (in the MC68000)
.SS	supervisor stack pointer
.US	user stack pointer

EXAMPLE	COMMENT
TUTOR 1.X > .PC .PC=00001010	Display program counter.
TUTOR 1.X > .A7 1300	Set address register seven.
TUTOR 1.X > .R5 5500	Set relative offset register five.

```
TUTOR 1.X > DF
PC=00001010 SR=2704=.S7..Z.. US=FFFFFFFF SS=00001300
D0=0000D0D0 D1=0000D1D1 D2=0000D2D2 D3=00D3D3D3
D4=D4D4D4D4 D5=000000D5 D6=00000000 D7=00000000
A0=00000000 A1=00000000 A2=00000000 A3=00000000
A4=00000000 A5=00000000 A6=00000000 A7=00001300
--------------------001010    FFFF                DC.W    $FFFF
```

See also: DF, OF

3.5.23 Transparent Mode TM

TM [<exit character>] [<trailing character>]

The TM command connects the two serial ports of the board together and ignores all input/output between them until the exit character is entered from the terminal attached to serial port 1. The default exit character is CTRL A ($01).

In the transparent mode, the ECB monitors the data transfer only until it sees the exit character; the exit character, therefore, is also transmitted to the host. The ECB must send another character, called the trailing character, to the host to remove or cancel the exit character from the host's buffer. Otherwise, the exit character will still be in the buffer the next time the transparent mode is entered. The default trailing character is CTRL X ($18). Other trailing characters may be selected.

NOTE

In order to enter a trailing character, an exit character must first be entered. Otherwise, the intended trailing character will be interpreted as the exit character.

Some possible exit or trailing characters such as NUL ($00), space ($20), backspace ($08), end of transmission ($04), cancel ($18), line feed ($0A), and carriage return ($0D) cannot be specified as part of the TM command line. These characters are used as separators or control characters or are ignored by the command interpreter. To use these types of characters as exit and trailing characters, the character must be written to RAM using the MM command. The trailing and exit characters are the fifth and sixth bytes, respectively, of the 6-byte options variable described in paragraph 3.5.21.

An asterisk (*) as the first character of the command line means to transmit the rest of the line to the host (port 2).

The modem connected to port 2 should operate at the same baud rate as the terminal connected to port 1.

See also: LO, PF, VE

EXAMPLE	COMMENTS
TUTOR 1.X >	Startup or reset condition.
TUTOR 1.X > TM	Command to enter transparent mode.
TRANSPARENT EXIT=$01 = CTL A	TUTOR prints this; EXIT=$01=CTL A means that in order to exit this mode, the operator must enter CTRL A.

User talks directly to host, uses editor, assembler, etc.

CTRL A	Ends the transparent mode.
TUTOR 1.X >	TUTOR prints this and system is ready for new command.

NOTE: Other exit and trailing characters can be specified. As examples,

```
TUTOR 1.X > TM CTRL R

*TRANSPARENT* EXIT=$12 = CTL R
```

or

```
TUTOR 1.X > TM 7 *              Trailing character=$2A=*

*TRANSPARENT* EXIT=$37 = 7

TUTOR 1.X > PF

FORMAT=   15 15
CHAR NULL=00 00
C/R  NULL=00 00
OPTIONS@0004E6

TUTOR 1.X > MM 4EA              Enter NULL ($00) trailing character directly
0004EA   2A ?0.                 into option byte.

TUTOR 1.X >
```

TR [<count>]
T [<count>]

The TR command 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, and the instruction pointed to by the program counter is disassembled.

After the trace mode is entered, the prompt includes a colon (i.e., TUTOR 1.X :>). While in this mode, the single character, carriage return, will cause one instruction to be traced. To exit, any command may be entered, followed by a carriage return.

Breakpoints and breakpoint counts are in effect during trace.

Limited tracing can be done within the TUTOR firmware. However, the maximum count is one. Because the stacks are shared by the trace command and the rest of TUTOR, they may become jumbled up when tracing is done in the debugger.

COMMAND FORMAT	DESCRIPTION
TUTOR 1.X > TR	Trace one instruction.
TUTOR 1.X :> T count	Trace "count" (hex) instructions.
TUTOR 1.X :>	Carriage return (CR) executes next instruction.

See also: DF, GO, GT, TT

EXAMPLE

```
TUTOR  1.X > .R6 2000

TUTOR  1.X > .PC 0+R6

TUTOR  1.X > TR 3
PHYSICAL ADDRESS=00002000
PC=00002002 SR=2700=.S7..... US=FFFFFFFF SS=000007BC
D0=0030FF43 D1=0030FF43 D2=0FFFFFFC D3=00000000
D4=FFFFFFFC D5=FFFFFFFC D6=00000002 D7=00000000
A0=00010040 A1=00002004 A2=000007B6 A3=0000053A
A4=00002004 A5=0000053A A6=000007B6 A7=000007BC
--------------------000002+R6 45F82056            LEA.L    $00002056,A2
PC=00002006 SR=2700=.S7..... US=FFFFFFFF SS=000007BC
D0=0030FF43 D1=0030FF43 D2=0FFFFFFC D3=00000000
D4=FFFFFFFC D5=FFFFFFFC D6=00000002 D7=00000000
A0=00010040 A1=00002004 A2=00002056 A3=0000053A
A4=00002004 A5=0000053A A6=000007B6 A7=000007BC
--------------------000006+R6 4EF900008152        JMP      $00008152

.PC within "DEBUGGER".
PC=00008152 SR=2700=.S7..... US=FFFFFFFF SS=000007BC
D0=0030FF43 D1=0030FF43 D2=0FFFFFFC D3=00000000
D4=FFFFFFFC D5=FFFFFFFC D6=00000002 D7=00000000
A0=00010040 A1=00002004 A2=00002056 A3=0000053A
A4=00002004 A5=0000053A A6=000007B6 A7=000007BC
--------------------006152+R6 48B800010406        MOVEM.W  D0,$0406

TUTOR  1.X :>
```

3.5.25 Trace to Temporary Breakpoint TT

TT <breakpoint address>

The TT command performs the following:

a. Sets a temporary breakpoint at the address specified.

b. Starts program execution in the trace mode at the address specified in the program counter (PC) (see TR command).

c. Traces until any breakpoint with a zero count is encountered.

d. Resets the temporary breakpoint.

The temporary breakpoint is not displayed by the BR command.

See also: DF, GO, GT, TR

```
EXAMPLE                                          COMMENT

TUTOR  1.X > .PC 2000

TUTOR  1.X > TT 2004
PHYSICAL ADDRESS=00002004                        Temporary breakpoint address - $2004
PHYSICAL ADDRESS=00002000                        Execution address - $2000
PC=00002002 SR=2708=.S7.N... US=FFFFFFFF SS=000007BC
D0=BDBC4144 D1=BDBC4144 D2=FFFFFFFF D3=FFFFFFFF
D4=FFFFFFFF D5=FFFFFFFF D6=FFFFFFFF D7=FFFFFFFF
A0=FFFFFFFB A1=FFFFFFFF A2=FFFFFFFF A3=FFFFFFFF
A4=FFFF7FFF A5=FFFFFFFF A6=FFFFFFFF A7=000007BC
--------------------002002    2A07                  MOVE.L   D7,D5

AT BREAKPOINT
PC=00002004 SR=2708=.S7.N... US=FFFFFFFF SS=000007BC
D0=BDBC4144 D1=BDBC4144 D2=FFFFFFFF D3=FFFFFFFF
D4=FFFFFFFF D5=FFFFFFFF D6=FFFFFFFF D7=FFFFFFFF
A0=FFFFFFFB A1=FFFFFFFF A2=FFFFFFFF A3=FFFFFFFF
A4=FFFF7FFF A5=FFFFFFFF A6=FFFFFFFF A7=000007BC
--------------------002004    60FA                  BRA.S    $002000
```

3.5.26 Verify (in S-Record Format) VE

VE <port number>[;=text]

The VE command verifies the current contents of memory with the object data in S-record format from a device external to memory. The device can be attached to Port 1, Port 2, or Port 4 — the cassette recorder.

The command has the basic forms:

COMMAND	PORT #	EXTERNAL DEVICE
VE1	Port 1	Terminal (i.e., terminal with tape driver)
VE	Port 2	Host (modem) - default port
VE2	Port 2	Host (modem)
VE4	Port 4	Audio cassette

A timeout feature is present for Port 2. If the host connected to Port 2 does not respond within approximately 10 seconds, the message TIMEOUT will be sent to Port 1 and the VE command will be aborted.

The optional [;=text] is used only with Port 2. The text following the "=" is sent to Port 2. In this manner, the Port 2 device knows what data to send for verification (in S-record format). When a mismatch is found between data in memory and the S-record object data, TUTOR will display the differences. If a display of differences occurs, the record following the record displayed may be lost. To prevent this, the ECB can send characters to the host to stop and start the transfer of the S-records. Paragraph 4.5.2 describes this feature.

Any record not containing an S0, S1, S2, S8, or S9 string is ignored. See Appendix A for information on S-records.

The offset contained in register R0 is added to the addresses of the data contained in each S-record.

See also: DU, LO, PF

EXAMPLE	COMMENT
TUTOR 1.X > VE ;=COPY TEST1.MX,# S1131000-.-.49-.4E-.-.-.-.-.-.-.-.-.-.-.-. ERROR	Verify the file TEST1.MX. The record is an S1 record with $13 byte-count starting at address $001000, and there are differences in the third and fifth bytes.
TUTOR 1.X >	
TUTOR 1.X > VE4 S1132000-.-.-.-.-.-.-.-.-.-.21-.-.-.-.-.-. ERROR	Verify tape file.
TUTOR 1.X >	

3.6 COMMAND SUMMARY AND MESSAGES

TABLE 3-2. TUTOR Commands and Options

COMMAND	DESCRIPTION
BF <address1> <address2> <word>	Block of Memory Fill
BM <address1> <address2> <address3>	Block of Memory Move
BR [<address>[;<count>]]	Breakpoint Set
BS <address1> <address2> <data> [<mask>] [;<option>]	Block of Memory Search; options B, W, L
BT <address1> <address2>	Block of Memory Test
DC <expression>	Data Conversion
DF	Display Formatted Registers
DU[<port number>] <address1> <address2> [<text..>]	Dump Memory (S-records)
GD [<address>]	Go Direct
GO [<address>]	Go
GT <breakpoint address>	Go Until Breakpoint
HE	Help
LO[<port number>] [;<options>] [=text]	Load (S-records); options X, -C
MD[<port number>] <address1> [<count>] [;<option>]	Memory Display; option DI
MM <address> [;<options>]	Memory Modify; options W, L, O, V, N, DI
MS <address> <data...>	Memory Set
NOBR [<address> <address>....]	Breakpoint Remove
NOPA	Reset Printer Attach
OF	Display Offsets
PA	Printer Attach
PF[<port number>]	Port Format
.Rx	Individual Register Display/Change

TABLE 3-2. TUTOR Commands and Options (cont'd)

COMMAND	DESCRIPTION
TM [<exit character>]	Transparent Mode
TR [<count>]	Trace
TT <breakpoint address>	Temporary Breakpoint Trace
VE[<port number>] [=text]	Verify (S-records)
* text....	Send Message to Port 2 (1)
.A0 - .A7 [<expression>]	Display/Set Address Register (2)
.D0 - .D7 [<expression>]	Display/Set Data Register (2)
.R0 - .R6 [<expression>]	Display/Set Relative Offset Register (2)
.PC [<expression>]	Display/Set Program Counter (2)
.SR [<expression>]	Display/Set Status Register (2)
.SS [<expression>]	Display/Set Supervisor Stack Pointer (2)
.US [<expression>]	Display/Set User Stack Pointer (2)
(BREAK)	Abort command
(DEL)	Delete character
(CTRL D)	Redisplay line
(CTRL H)	Delete character
(CTRL W)	Suspend output (3)
(CTRL X)	Cancel command line
(CR)	Process command line

NOTES:

(1) See writeup of TM command.

(2) See writeup of .Rx command.

(3) When CTRL W is used, the output display can be continued by entering any character.

TABLE 3-3. Error Messages and Other Messages

ERROR MESSAGE	MEANING
PRINTER NOT READY	Printer is not properly connected or cannot receive output
SYNTAX ERROR	Error in command line
ERROR	Error
ILLEGAL INSTRUCTION	Instruction used an illegal op-code during program execution
ADDR TRAP ERROR BUS TRAP ERROR	See Traps in MC68000 User's Manual and paragraph 4.3.5.1.
IS NOT A HEX DIGIT	Improper character entered in a field that requires a hexadecimal digit
DATA DID NOT STORE	Data did not go where intended
INVALID ADDRESS=	Too big (1 in bits 24-31) or odd for .W or .L (1 in bit 0)
WHAT	Program does not recognize user's entry
NOT HEX=	Same as IS NOT A HEX DIGIT
FAILED AT.. WROTE=.. READ=..	Read or write command failure output by BT
UNDEFINED TRAP 14	Trap function code is not defined
CHKSUM=	Indicates received checksum is incorrect, correct checksum is given
OTHER MESSAGE	**MEANING**
TUTOR 1.X >	TUTOR prompt
TIMEOUT	Displayed if Port 2 does not respond to LO or VE within 10 seconds
FORMAT=	Displayed by PF command
CHAR NULL=	Displayed by PF command
C/R NULL=	Displayed by PF command

TABLE 3-3. Error Messages and Other Messages (cont'd)

ERROR MESSAGE	MEANING
OTHER MESSAGE	**MEANING**
OPTIONS@XXXXXX	Displayed by PF command
TRANSPARENT EXIT=$01=CTL A	Displayed by TM command
SOFTWARE ABORT	Displayed when abort button is pressed
BREAK	BREAK key has been used
AT BREAKPOINT	Indicates program has stopped at breakpoint
BREAKPOINTS	Displayed by BR command
PHYSICAL ADDRESS=	Actual address by command
PC within "DEBUGGER"	Displayed by trace commands

CHAPTER 4

USING THE ASSEMBLER/DISASSEMBLER

Integrated into the MC68000 Educational Computer firmware is an assembler/disassembler function. The disassemble function is called as an option to the Memory Display (MD ;DI) and Memory Modify (MM ;DI) commands, and also is used during execution of system trace and the display register command. The assemble function allows code entry and editing and is invoked by the Memory Modify (MM ;DI) command. Chapter 4 is a detailed discussion of the assembler/disassembler.

CHAPTER 4

USING THE ASSEMBLER/DISASSEMBLER

4.1 INTRODUCTION

Included as part of the MC68000 Educational Computer firmware is an assembler/disassembler function. The assembler/disassembler is an interactive assembler/editor in which the source program is not saved. Each source line is translated into the proper MC68000 machine language code and is stored in memory 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 MC68000 instructions are translated. The mnemonic ILLEGAL, described in Appendix B of the MC68000 User's Manual, is not recognized by the educational computer assembler. Also, refer to paragraph 4.2.2.4 for restrictions on the use of the mnemonic CCR.

The educational board assembler is effectively a subset of the MC68000 Resident Structured Assembler. It has more limitations than the resident assembler, such as not allowing line numbers and labels; however, it is a powerful tool for creating, modifying, and debugging MC68000 code.

4.1.1 M68000 Assembly Language

The symbolic language used to code source programs for processing by the assembler is called M68000 assembly language. This language is a collection of mnemonics representing:

- Operations
  - MC68000 machine-instruction operation codes
  - Directive (pseudo-op)
- Operators
- Special symbols

4.1.1.1 Machine-Instruction Operation Codes. That part of the assembly language that provides mnemonic machine-instruction operation codes for the MC68000 machine instructions is described in the MC68000 16-Bit Microprocessor User's Manual, MC68000UM. The user should reference this manual.

4.1.1.2 Directives. The assembly language can contain mnemonic directives which specify auxiliary actions to be performed by the assembler. Directives are not always translated to machine language.

Assembler directives assist the programmer:

- In controlling the assembler output
- In defining data and symbols
- In allocating storage

The educational board assembler recognizes only one directive called define constant (DC.W). This directive is used to define data within the program. Refer to paragraph 4.2.4 for a description of this directive.

4.1.2 Comparison with MC68000 Resident Structured Assembler

There are several major differences between the MEX68KECB assembler and the MC68000 Resident Structured Assembler. The resident assembler is a two-pass assembler that processes an entire program as a unit, while the educational board assembler processes each line of a program as an individual unit. Due mainly to this basic functional difference, the capabilities of the TUTOR assembler are more restricted:

a. Label and line numbers are not used. - Labels are used to reference other lines and locations in a program. The one-line assembler has no knowledge of other program lines and, therefore, cannot make the required association between a label and the label definition located on a separate line.

b. Source lines are not saved. - In order to read back a program after it has been entered, the machine code is disassembled and then displayed as mnemonic and operands.

c. Limited error indication. - The one-line assembler will show a question mark (?) under the portion of the source statement where an error probably occurred, or will display the word "ERROR" or other short message. In contrast, the resident assembler generates specific error messages for over 60 different types of errors.

d. Only one directive (DC.W) is accepted.

e. No macro operation capability is included.

f. No conditional assembly is used.

g. Several symbols recognized by the resident assembler are not included in the MEX68KECB assembler character set. These symbols include !, >, and <. Two other symbols, * and /, each have multiple meanings to the resident assembler, depending on the context, but only one meaning to the MEX68KECB assembler. Finally, the ampersand character (&) specifies a decimal number when used with the ECB assembler (although numbers with no prefix are assumed to be decimal) while this symbol represents a logical AND function to the resident assembler. Paragraph 4.2.1.5 describes the MEX68KECB assembler character set.

Although functional differences exist between the two assemblers, the one-line assembler is a true subset of the resident assembler. The format and syntax used with the TUTOR assembler are acceptable to the resident assembler except as described in g. above.

4.2 SOURCE PROGRAM CODING

A source program is a sequence of source statements arranged in a logical way to perform a predetermined task. Each source statement occupies a line and must be either an executable instruction or a DC.W assembler directive. Each source statement follows a consistent source line format.

4.2.1 Source Line Format

Each source statement is a combination of operation and, as required, operand fields; line numbers, labels, and comments are not used. The general format is:

sp <operation field> sp [<operand field>]

The space (sp) must be the first character of each line. This is to be consistent with the resident assembler, which expects the first field of each line to be either a space or a label. Because the TUTOR assembler never allows a label, the first character must always be a space.

4.2.1.1 Operation Field. The operation field must follow at least one space (more can be used) and entries can consist of one of two categories:

a. Operation codes - which correspond to the MC68000 instruction set, or

b. Define constant directive - the DC.W is recognized to define a constant in a word location. This is the only directive recognized by the assembler.

The size of the data field affected by an instruction is determined by the data size code. Some instructions and directives can operate on more than one data size. For these operations, the data size code must be specified or a default size applicable to that instruction will be assumed. The size code need not be specified if only one data size is permitted by the operation. The data size code is specified by a period (.), appended to the operation field, and followed by B, W, or L, where:

B = Byte (8-bit data)
W = Word (the usual default size; 16-bit data).
L = Long word (32-bit data)

The data size code is not permitted, however, when the instruction or directive does not have a data size attribute.

Examples (legal):

LEA	2(A0),A1	Long word size is assumed (.B,.W not allowed); this instruction loads effective address of first operand into A1.
ADD.B	(A0),D0	This instruction adds the byte whose address is (A0) to lowest order byte in D0.
ADD	D1,D2	This instruction adds low order word of D1 to low order word of D2. (W is the default size code.)
ADD.L	A3,D3	This instruction adds entire 32-bit (long word) contents of A3 to D3.

Example (illegal):

SUBA.B	#5,A1	Illegal size specification (.B not allowed on SUBA). This instruction would have subtracted the value 5 from the low order byte of A1; byte operations on address registers are not allowed.

4.2.1.2 Operand Field. If present, the operand field follows the operation field and is separated from the operation field by at least one space. When two or more operand subfields appear within a statement, they must be separated by a comma. In an instruction like ' ADD D1,D2' the first subfield (D1) is generally applied to the second subfield (D2) and the results placed in the second subfield. Thus, the contents of D1 are added to the contents of D2 and the result is saved in register D2. In the instruction ' MOVE D1,D2' the first subfield (D1) is the sending field and the second subfield (D2) is the receiving field. In other words, for most two-operand instructions, the general format ' opcode source,destination' applies.

4.2.1.3 Disassembled Source Line. The disassembled source line may not look identical to the source line entered. The disassembler makes a decision on how to represent a numerical value based on how it interprets the number's use. If the number is determined to be an address or a "would-be" address, it is displayed in hexadecimal; everything else is decimal. For example,

```
MOVE.L  #$1234, $5678
```

disassembles to

```
005000    21FC000012345678    MOVE.L  #4660,$00005678
```

Also, for some instructions, there are two valid mnemonics for the same op code, or there is more than one assembly language equivalent. The disassembler may choose a form different from the one originally entered. As examples:

a. BRA is returned for BT
b. DBF is returned for DBRA

NOTE

The assembler recognizes two forms of mnemonics for two branch instructions. The BT form (branch conditionally true) has the same op code as the BRA instruction. Also, DBRA (decrement and branch always) and DBF (never true, decrement, and branch) mnemonics are different forms for the same instruction. In each case, the assembler will accept both forms.

4.2.1.4 Mnemonics and Delimiters. The assembler recognizes all MC68000 instruction mnemonics except ILLEGAL. Numbers are recognized as both decimal and hexadecimal, with decimal the default case (note that this is reverse to the TUTOR commands):

a. Decimal - is a string of decimal digits (0-9) without a prefix (default) or preceded by an optional ampersand (&). Examples are:

```
1234
&1234
```

b. Hexadecimal - is a string of hexadecimal digits (0-9, A-F) preceded by a dollar sign ($). An example is:

```
$AFE5
```

One or more ASCII characters enclosed by apostrophes (') constitute an ASCII string. ASCII strings are left-justified and zero-filled (if necessary), whether stored or used as immediate operands. This left justification will be to a word boundary if one or two characters are specified, or to a long word boundary if the string contains more than two characters.

```
005000    5300              DC.W     'S'
005002    223C41424344      MOVE.L   #'ABCD',D1
005008    3536              DC.W     '56'
```

NOTE

The MC68000 has seventeen 32-bit registers (D0-D7, A0-A6, SSP, USP) in addition to a 32-bit program counter (24 bits available) and a 16-bit status register. Registers D0-D7 are used as data registers for byte, word, and long word operations. Registers A0-A6 and SSP and USP are used as software stack pointers and base address registers; they may also be used for word and long word data operations. All 17 registers may be used as index registers. Register A7 is a pseudo register, used as the system stack pointer corresponding to either SSP or USP, depending on the operating state.

The following register mnemonics are recognized by the assembler:

D0-D7	Data Registers
A0-A7	Address Registers Address register seven represents the system stack pointer of the active system state.
USP	User stack pointer. Used only in privileged instructions which are restricted to supervisory state.
CCR	Condition code register (low 8 bits of SR)
SR	Status register. All 16 bits may be modified in the supervisor state. Only low 8 bits (CCR) may be modified in user state.
PC	Program Counter. Used only in forcing program counter-relative addressing

4.2.1.5 Character Set. The character set recognized by the MEX68KECB assembler is a subset of ASCII, and these are listed below:

a. The uppercase letters A through Z

b. The integers 0 through 9

c. Arithmetic operators: + -

d. Parentheses ( )

e. Characters used as special prefixes:

(pound sign) specifies the immediate form of addressing

$ (dollar sign) specifies a hexadecimal number

& (ampersand) specifies a decimal number

@ (commercial at sign) specifies an octal number

% (percent sign) specifies a binary number

' (apostrophe) specifies an ASCII literal character

f. Five separating characters:

Space

, (comma)

. (period)

/ (slash)

- (dash)

g. The character * (asterisk) indicates current location.

4.2.2 Instruction Summary

The following paragraphs summarize the types of MC68000 instructions, their variations, and addressing modes. The MC68000 User's Manual describes the MC68000 instructions and addressing modes in greater detail.

4.2.2.1 Arithmetic Operations. The MC68000 instruction set includes the operations of add, subtract, multiply, and divide. Add and subtract are available for all data operand sizes, including extended, and also for address operands.

Multiply and divide may be signed or unsigned. Operations on decimal data (BCD) include add, subtract, and negate. The general form is:

```
OPERATION.SIZE    SOURCE,DESTINATION
```

Example:

```
ADD.W   D1,D2     Adds low order word of D1 to low order word of D2.

SUB.B   #5,(A1)   Subtracts 5 from the byte whose address is contained in A1.
```

4.2.2.2 MOVE Instruction. The MOVE instruction is used to move data between registers and/or memory. These moves include register-to-register, memory-to-memory, memory-to-register, and register-to-memory transfers. The general form is:

```
MOVE.SIZE     SOURCE,DESTINATION
```

Examples:

MOVE	D1,D2	Moves low order word of D1 into low order word of D2.
MOVE.L	$02000,$03000	Moves long word addressed by $02000 into long word addressed by $03000.
MOVE.W	#'A',1000	Moves word with value of 'A'00 into byte addressed by &1000.
MOVE	$2000,A3	Moves word addressed by $2000 into low order word of A3.

The MOVEQ mode always specifies a 32-bit destination operand and is, therefore, used only for a MOVE.L operation. The source data is eight bits and is sign-extended to a 32-bit value.

4.2.2.3 Compare Instructions. The general formats of the compare and check instructions are:

```
CMP.SIZE      OPERAND1,OPERAND2
CHK           BOUNDS,REGISTER
```

where $operand_1$ is compared to $operand_2$ by the non-destructive subtraction of $operand_1$ from $operand_2$ without altering $operand_1$ or $operand_2$.

Condition codes resulting from the subtraction include: N set for negative result, Z set for zero result, V set for overflow, and C set for a generated borrow.

The CHK instruction will cause a system trap if the register contents are less than zero or greater than the value specified by "bounds".

Examples:

CMP.L	$2000,D1	Compares long word at location $2000 with contents of D1, setting condition codes accordingly.
CHK	(A0),D3	Compares word whose address is in A0 with lower order word of D3; if check fails (see the MC68000 User's Manual), a system trap is initiated.

4.2.2.4 Logical Operations. Logical operations include AND, OR, EXCLUSIVE OR, NOT, and two logical test operations. These functions may be done between registers, between registers and memory, or with immediate source operands. The general form is:

```
OPERATION.SIZE     SOURCE,DESTINATION
```

Example:

```
AND   D1,D2        Low order word of D2 receives logical 'and' of low
                   order words in D1 and D2.
```

The destination may also be the status register (SR). When in the user state, only the lower eight bits of the status register may be modified. The byte size extension must be used. Note, however, that the mnemonic CCR is not accepted by the assembler for logical operations. Instead, the mnemonic SR must be used with a size extension of .B. CCR is used only with the MOVE instruction.

EXAMPLE: ANDI.B #5,SR instead of ANDI.B #5,CCR

4.2.2.5 Shift Operations. Shift operations include arithmetic and logical shifts, as well as rotate and rotate with extend. All shift operations may be either fixed with the shift count in an immediate field or variable with the count in a register. Shifts in memory of a single bit position left or right may also be done. The general form is:

```
OPERATION.SIZE     COUNT,OPERAND
```

Example:

```
LSL.W    #5,D3     Performs a left, logical shift of low order word of D3
                   by 5 bits; .W is optional (default).

ASR      #1,(A2)   Performs a right, arithmetic shift of the word whose
                   address is contained in A2; since this is a memory
                   operand, the shift is only 1 bit.

ROXL.B   D3,D2     Performs a right rotation with extend bit of low order
                   byte of D2; shift count is contained in D3.
```

4.2.2.6 Bit Operations. Bit operations allow test and modify combinations for single bits in either an 8-bit operand for memory destinations or a 32-bit operand for data register destinations. The bit number may be fixed or variable. The general form is:

```
OPERATION      BITNO,OPERAND
```

Example:

```
BCLR   #3,$44(A3)  Tests bit number 3 in byte whose address is given by
                   address in A3 plus displacement of $44, sets or clears
                   the Z condition code, and clears the specified bit in
                   the destination.

BCHG   D1,D2       Tests a bit in D2, reflects its value in condition code
                   Z, and then changes value of that bit; bit number is
                   specified in D1.
```

4.2.2.7 Conditional Operations. Condition codes can be used to set and clear data bytes. The general form is:

```
OPERATION     LOCATION
```

Example:

```
SNE   (A5)+        If condition code 'NE' (not equal) is true, then set
                   byte whose address is in A5 to 1's; otherwise, set that
                   byte to Ø's; increment A5 by 1.
```

4.2.2.8 Branch Operations. Branch operations include a branch to subroutine, an unconditional branch, and 14 conditional branch instructions. The general form is:

```
OPERATION.EXTENT     LOCATION
```

Examples:

```
003058   61A6       BSR    $3000    Branch to subroutine at location $3000.

003FF0   670E       BEQ.S  $4000    Short branch to $4000, on condition "EQ".

003FF0   6600000E   BNE.L  $4000    Long branch to $4000, on condition "NE".

003FF0              BPL.S  $3000    Short branch not allowed; displacement >
                                    8 bits.
```

All conditional branch instructions are PC-relative addressing only, and may be either one- or two-word instructions. The corresponding displacement ranges are:

```
one-word     -128...+127 bytes          (8-bit displacement)

two-word     -32768...+32767 bytes      (16-bit displacement)
```

By default, the assembler will resolve all references, both relative and absolute, by using the shorter form of the effective address in the operand reference, if possible; otherwise, the longer form will be chosen. The user can force the long form of the instruction by using the .L suffix.

In a short branch instruction, the operand must not reference the statement which immediately follows it. This would result in a displacement value of 0, which is recognized by the assembler as an error condition.

4.2.2.9 Jump Operations. Jump operations include a jump to subroutine and an unconditional jump. The general form is:

```
OPERATION.EXTENT      LOCATION
```

Example:

JMP	4(A7)	Unconditional jump to the location 4 bytes beyond the address in A7.
JMP.L	$2000	Long (absolute) jump to the address $2000.
JSR	$3000	Jump to subroutine at address $3000.

Jumps may specify any control addressing mode as the destination location. All references will use the shorter absolute address format, if possible; otherwise, the longer format will be used. The default extent may be overridden on a single jump operation to a label by appending "S" or "L" as an extent code for the instruction.

4.2.2.10 DBcc Instruction. This instruction is a looping primitive of three parameters: condition, data register, and address. The instruction first tests the condition to determine if the termination condition for the loop has been met and, if so, no operation is performed. If the termination condition is not true, the data register is decremented by one. If the result is -1, execution continues with the next instruction. If the result is not equal to -1, execution continues at the indicated location. The address must be within 16-bit displacement. The general format of the instruction is:

```
DBcc      DATA REGISTER,ADDRESS
```

4.2.2.11 Load/Store Multiple. This instruction allows the loading and storing of multiple registers. Its general format is:

```
MOVEM.SIZE  REGISTERS,LOCATION (register to memory)
MOVEM.SIZE  LOCATION,REGISTERS (memory to register)
```

where size may be either W (default) or L.

The "registers" operand may assume any combination of the following:

```
R1/R3/R6, etc., means R1 and R3 and R6
R1-R3, etc., means R1 through R3
```

The order in which the registers are processed is independent of the order in which they are specified in the source line; rather, the order of register processing is fixed by the instruction format. See MOVEM instruction in Appendix B of the MC68000 User's Manual for further details.

NOTE

Registers discussed here include data registers zero through seven and address registers zero through seven but not the software offset registers (R0 through R7) used by TUTOR.

Examples:

`MOVEM (A6)+,D1/D5/D7`	Load registers D1, D5, and D7 from three consecutive (sign-extended) words in memory, the first of which is given by the address in A6; A6 is incremented by 2 after each transfer.
`MOVEM.L A2-A6,-(A7)`	Store registers A2 through A6 in 5 consecutive long words in memory; A7 is decremented by 4 (because of .L); A6 is stored at A7; A7 is decremented by 4; A5 is stored at A7, etc.
`MOVEM (A7)+,A1-A3/D1-D3`	Loads registers D1, D2, D3, A1, A2, A3 in order from the six consecutive (sign-extended) words in memory, starting with address in A7 and incrementing A7 by 2 at each step.
`MOVEM.L A1/A2/A3,$2000`	Store registers A1, A2, A3 in three consecutive long words starting with location $2000.

4.2.2.12 Load Effective Address. This instruction allows computation and loading of an effective address into an address register. The general format is:

```
LEA   OPERAND,REGISTER
```

Example:

```
LEA   (A2,D5),A1          Load A1 with effective address specified by
                          first operand; see later explanation of
                          addressing mode "address register indirect
                          with index" (paragraph 4.2.3.2).
```

4.2.2.13 Variants on Instruction Types

Certain instructions allow a "quick" form when immediate data within a restricted size range appears as an operand. It is necessary for the programmer to "force" such a form by appending a "Q" to the mnemonic op code (to indicate "quick") on instructions for which such a form exists. If the specified quick form does not exist, or if the immediate data does not conform to the size requirements of the abbreviated form, an error will be generated.

Some instructions also have "address" variant forms (which refer to address registers as destinations); these variants append an "A" to the instruction mnemonic (e.g., ADDA, CMPA). This variant will be chosen by the assembler without programmer specification, when appropriate to do so; the programmer need specify only the general instruction mnemonic. However, the programmer may "force" or specify such a variant form by appending the "A". If the specified variant does not exist or is not appropriate with the given operands, an error will be generated.

The CMP instruction also has a memory variant form (CMPM) in which both operands are a special class of memory references. The CMPM instruction requires postincrement addressing of both operands. The CMPM instruction will be selected by the assembler, or it may be specified by the programmer.

The variations — A, Q, and M — must conform to the following restrictions:

A Must specify an address register as a destination, and cannot specify a byte size code (.B).

Q Requires immediate operand be in a certain size range. MOVEQ also requires longword data size.

M Both operands must be postincrement addresses.

For example, the instruction

```
ADDQ  #9,D0       Attempts to add value 9 to D0
```

will cause an assembly error, because the immediate operand is not in the valid size range (1 through 8).

4.2.3 Addressing Modes

Effective address modes, combined with operation codes, define the particular function to be performed by a given instruction. Effective addressing and data organization are described in detail in Section 2, "Data Organization and Addressing Capabilities", of the MC68000 User's Manual.

References to data addresses may be odd only if a byte is referenced. Data references involving words or long words must be even. Likewise, instructions must begin on an even word boundary.

Individual bits within a byte (operand for memory destinations) or long word (operand for data register destinations) may be addressed with the bit manipulation instructions (paragraph 4.2.2.6). Bits for a byte are numbered 7 to 0, with 7 being the most significant bit position and 0 the least significant. Bits for a long word are numbered from 31 to 0, with 31 being the most significant bit position and 0 the least significant bit position.

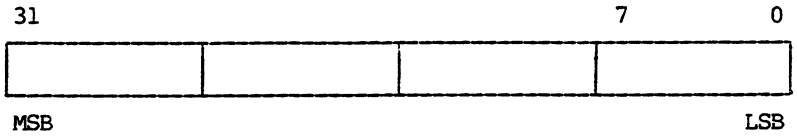


Table 4-1 summarizes the addressing modes defined for the MC68000, their syntax, and significant constraints.

TABLE 4-1. Address Modes

MODE	SYNTAX	COMMENTS
1) Register Direct		
a) Data register direct	Dn	
b) Address register direct	An	
2) Memory Address		
a) Address register indirect	(An)	
b) Address register indirect with predecrement	-(An)	
c) Address register indirect with postincrement	(An)+	
d) Address register indirect with displacement	<expr>(An)	<expr> must be absolute; displacement: 16 bits, sign-extended; register size is 32 bits (no options)
e) Address register indirect with index	<expr>(An,Am) <expr>(An,Dm)	<expr> must be absolute; displacement: 8 bits, sign-extended; note that A or D register may be used as index register (16 bits, sign-extended unless .L used); address register: 32 bits

TABLE 4-1. Address Modes (cont'd)

MODE	SYNTAX	COMMENTS
3) Special Address		
a) Absolute	<expr>	<expr> must specify an absolute address; there are two formats: absolute short is 16 bits, sign-extended; absolute long is 32 bits
b) PC with displacement	<expr>(PC)	forced PC with displacement; <expr> must be absolute; displacement: 16 bits, sign-extended
	<expr>(PC,An) <expr>(PC,Dn)	forced PC with index and displacement; <expr> must be absolute; displacement: 8 bits, sign-extended
c) Immediate data	#<expr>	<expr> must be absolute; may be used with .B, .W, .L
4) Implicit PC reference	—	Invoked by conditional branch (Bcc) or DBcc instruction; the effective address is a displacement from the PC; the displacement is either 8 or 16 bits, depending on the destination address and whether the default size is overridden on the current instructions.
5) Other implicit references	—	Some instructions make implicit reference to the system stack pointer (SP), the supervisor stack pointer (SSP), the user stack pointer (USP), or the status register (SR).

4.2.3.1 Register Direct Modes. These effective addressing modes specify that the operand is in one of the 17 multifunction registers (eight data and nine address registers). The operation is performed directly on the actual contents of the register. When an instruction is executed, references to A7 specify the supervisor stack pointer if the supervisor state status bit is set in the status register and the user stack pointer otherwise.

Notations:	An Dn	where n is between 0 and 7	Address register direct Data register direct

Examples:	CLR.L	D1	Clear all 32 bits of D1
	ADD	A1,A2	Add low order word of A1 to low order word of A2

4.2.3.2 Memory Address Modes. The following effective addressing modes specify that the operand is in memory and provide the specific address of the operand.

Address Register Indirect

The address of the operand is in the address register specified by the register field.

Notation: (An)

Examples:	MOVE	#5,(A5)	Move the value 5 to word whose address is contained in A5.
	SUB.L	(A1),D0	Subtract the value in the long word whose address is contained in A1 from D0.

Address Register Indirect with Predecrement

The address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four, depending upon whether the operand size is byte (.B), word (.W), or long (.L).

Notation: -(An)

Examples:	CLR	-(A2)	Subtract 2 from A2; clear word whose address is now in A2.
	CMP.L	-(A0),D0	Subtract 4 from A0; compare long word whose address is now in A0 with contents of D0.

Address Register Indirect with Postincrement

The address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two, or four, depending upon whether the size of the operand is byte (.B), word (.W), or long (.L).

Notation: (An)+

Examples:		
MOVE.B	(A2)+,D2	Move byte whose address is in A2 to D2; increment A2 by 1.
MOVE.L	(A4)+,D3	Move long word whose address is in A4 to D3; increment A4 by 4.

Address Register Indirect with Displacement

The address of the operand is the sum of the address in the address register and the 16-bit sign-extended displacement.

Notation: <expression>(An)

Examples:		
CLR.B	5(A0)	Clear byte whose address is given by adding 5 to contents of A0.
MOVE	#2,10(A2)	Move 2 to word whose address is given by adding 10 to contents of A2.

Address Register Indirect with Index

The address of the operand is the sum of the address in the address register, the 8-bit sign-extended displacement, and the contents of the index (A or D) register.

Notations:		
<expression>(An,Rn.W)		Specifies sign-extended low order word of index register.
<expression>(An,Rn.L)		Specifies entire contents of index register.

Examples:		
ADD	5(A1,D2),D5	Add to low order word of D5 the word whose address is given by addition of contents of A1, the sign-extended low order word of index register D2, and the displacement 5.
MOVE.L	D5,$20(A2,A3.L)	Move entire contents of D5 to long word whose address is given by addition of contents of A2, contents of entire index register A3, and the displacement $20.

4.2.3.3 Special Address Modes. Special address modes use the effective address register field to specify the special addressing mode instead of a register number. The following table provides the ranges for absolute short and long addresses.

32-bit address	16-bit representation of 32-bit address	
00000000 . . 00007FFF	0000 . . 7FFF	Absolute Short
00008000 . . FFFF7FFF	No representation in 16 bits	Absolute Long
FFFF8000 . . FFFFFFFF	8000 . . FFFF	Absolute Short

Absolute Short Address

The 16-bit address of the operand is sign extended before it is used. Therefore, the useful address range is 0 through $7FFF and $FFFF8000 through $FFFFFFFF.

Notation: XXX

Example:

```
002500    4EF80400        JMP     $400    Jump to hex address 400 specified
                                          as a 16-bit sign-extended address.
```

Absolute Long Address

The address of the operand is the 32-bit value specified.

Notation: XXX

Examples:

```
007800    4EF900012000    JMP     $12000  Jump to hex address 12000 specified
                                          as a 32-bit address.

002500    4EF900000400    JMP.L   $400    Jump to hex address 400 specified
                                          as a 32-bit address. The long
                                          address is forced by the .L option.
```

Program Counter with Displacement

The address of the operand is the sum of the address in the program counter (current instruction location plus two) and the sign-extended 16-bit displacement integer. The assembler calculates this sign-extended displacement by subtracting the address of the displacement word (i.e., current instruction address plus two) from the value in the operand field.

Notation:	<expression>(PC)	Forced program counter-relative; cannot be used for branch instructions

The branch instructions (BRA, BSR, Bcc, DBcc) are a special case of the program counter with displacement address mode. These instructions always use program counter relative addressing; the displacement integer, however, can be either 16 or 8 bits long for the BRA, BSR, and Bcc instructions. An 8- or 16-bit displacement is specified by an .S or .L, respectively, following the instruction mnemonic. However, since these instructions allow only one address mode, the program counter with displacement mode is not explicitly selected in the source line. Instead, only the destination address of the branch is specified as shown in the following examples. For all other instructions, the program counter with displacement mode must be explicitly selected.

Examples:

001050	6700FFAE	BEQ.L	$1000	Branch if EQ condition code to $1000. Displacement integer is 16 bits.
001050	6E0E	BGT	*+$10	Branch if GT condition code to 16 bytes past this instruction.
001050	4EFAF8AE	JMP	$900(PC)	Force the evaluation of $900 to be program counter-relative. Displacement = $0900-$(1050+2)=$F8AE.

Program Counter with Index

The address is the sum of the address in the program counter, the sign-extended 8-bit displacement value, and the contents of the index (A or D) register. The displacement is calculated in the same manner as above.

Notations:	<expression>(PC,Rn.W) <expression>(PC,Rn.L)	Forced program counter-relative with index using word (default) or long word index.

Examples:

005000	4EFAC0FE	JMP	$1100(PC)	Force evaluation of $1100 to be program counter-relative. Displacement value is 16 bits.
005000		JMP	$1100(PC,A2)	Destination address is out of range; displacement value is only 8 bits for program counter with index address mode.
001150	4EFBA0AE	JMP	$1100(PC,A2)	Force evaluation of $1100 to be program counter-relative with index. Lower 16 bits of A2 are used as the index.
002030	323B58CE	MOVE	$2000(PC,D5.L),D1	Force evaluation of $2000 to be program counter-relative with index. All 32 bits of D5 are used as the index.

NOTE: In the program counter with displacement and program counter with index address modes, the expression represents the actual memory address. For example, to jump to address $1050, the instruction JMP $1050(PC) might be used. The assembler calculates the required displacement to reach address $1050 from the current location. In the address register indirect with displacement and the address register indirect with index address modes, however, the expression represents the displacement rather than the memory address — hence, the instruction JMP $1050(A0) will jump to the memory address given by the contents of register A0 plus $1050.

Immediate Data

An absolute number may be specified as an operand by immediately preceding a number or expression with a '#' character. The immediate character (#) is used to designate an absolute number other than a displacement or an absolute address.

Notation: #XXX

Examples:			
	MOVE	#1,D0	Move value 1 to low order word of D0.
	SUB.L	#1,D0	Subtract value 1 from the entire contents of D0.

4.2.3.4 Notes on Addressing Options. By default, the assembler will resolve all references, both PC relative and absolute, by using the shorter form of the effective address in the operand reference, if possible; otherwise, the longer form will be chosen.

On an instruction which does not allow a size code, the reference default format may be overridden (for that instruction only) by appending .S (short) or .L (long) to the instruction mnemonic.

The shorter form of the effective address for relative branch instructions is an 8-bit displacement; the longer format is a 16-bit displacement. For absolute jumps, the shorter effective address is the 16-bit absolute short; the longer format is the 32-bit absolute long mode. In either relative branches or absolute jumps, if the shorter format is directed and the longer format is found necessary, an error will occur.

A long form may be forced by following the instruction mnemonic with .L

Example:

```
BEQ.L   $3050       If condition code 'EQ' (equal) is true, then branch to
                    $3050 (using the long form of the instruction).
```

In this case, the instruction size is forced to two words. An error will be printed if the operand field is not in the range of an 16-bit displacement.

Default actions of the assembler have been chosen to minimize two common address mode errors:

a. Displacement range violations

   Relative branch instructions (Bcc, BRA, BSR) allow either 8-bit or 16-bit displacements from the PC. On references in such instructions, the default action is to use the 8-bit displacement if the destination address is within that range; otherwise, the 16-bit displacement is used.

b. Inappropriate absolute short address

   Absolute addresses may be short (16-bit) or long (32-bit). On references with absolute effective address, the default action is to use the absolute short form if the address can be represented in 16 bits with sign extension; otherwise, the absolute long form is used.

4.2.4 DC.W Define Constant Directive

The format for the DC.W directive is:

sp DC.W <operand>

The function of the directive is to define a constant in memory. The DC.W directive can have only one operand (16-bit value) which can contain the actual value (decimal, hexadecimal, or ASCII). Alternatively, the operand can be an expression which can be assigned a numeric value by the assembler. The constant is aligned on a word boundary as word (.W) size is specified.

An ASCII string is recognized when characters are enclosed inside single quotes ('). Each character (7 bits) is assigned to a byte of memory, with the eighth bit (MSB) always equal to zero. If only one byte is entered, the byte is left justified. A maximum of two ASCII characters may be entered for each DC.W directive.

Examples are:

```
001022   04D2   DC.W   1234      Decimal number
001024   AAFE   DC.W   $AAFE     Hexadecimal number
001026   4142   DC.W   'AB'      ASCII string
001028   5443   DC.W   'TB'+1    Expression
00102A   4300   DC.W   'C'       ASCII character is left justified
```

4.3 ENTERING AND MODIFYING SOURCE PROGRAMS

User programs are entered into the Educational Computer RAM using the one-line assembler/disassembler. The program is entered in assembly language statements on a line-by-line base. The source code is not saved as it is converted immediately to machine code upon entry. This imposes several restrictions on the type of source line that can be entered.

Symbols and labels, other than the defined instruction mnemonics, are not allowed. The assembler has no means to store the associated values of the symbols and labels in lookup tables. This forces the programmer to use memory addresses and to enter data directly rather than use labels.

Also, editing is accomplished by retyping the entire new source line. Lines can be added or deleted by moving a block of memory data to free up or delete the appropriate number of locations.

In order to more clearly describe the procedures used to enter, modify, and execute a program, a specific example will be described. Figure 4-1 lists a program that converts an ASCII coded number into its hexadecimal equivalent. An ASCII character is in the lowest 8 bits of register D0 when the program is entered. Upon exiting, D0 contains the equivalent hexadecimal digit (0 to F), or an FF if the ASCII character does not correspond to a proper hex number.

```
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 / Bits b5						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	Column / Hex	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    0C0C0041            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.

CHAPTER 5

TRAP 14 HANDLER

An additional function contained within the MC68000 Educational Computer firmware is a function called the TRAP 14 handler. This function can be called by the user program to utilize various routines within TUTOR, to call special routines, and to return control to TUTOR. This chapter describes the TRAP 14 handler and how it is used.

CHAPTER 5

TRAP 14 HANDLER

5.1 WHAT IS THE TRAP 14 HANDLER?

The TRAP 14 handler is a function contained within TUTOR that allows system calls from user programs. The system calls can be used to access selected functional routines contained within the firmware, including ASCII/hex conversion, input routines, output routines, etc. The user is then not required to reproduce these functions in his own program.

5.1.1 Types of Exceptions

Exceptions are inputs to the MC68000 which change the "normal" flow of a program. They can be generated by either internal or external causes. There are three basic kinds of exceptions recognized by the MC68000:

a. Exceptions which cause the instruction currently being executed to be aborted. These consist of reset, bus error, and address error exceptions.

b. Exceptions which allow the current instruction to be completed before processing the exception. These are trace, interrupt, illegal instruction, and privilege violation exceptions. Illegal instructions and privilege violations cause the current instruction to be executed as an NOP instruction.

c. Exceptions which occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, DIVS, and DIVU (when dividing by zero) instructions are included in this group.

5.1.2 MC68000 Exception Processing

Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the Supervisor mode (S) bit is asserted, putting the processor into the supervisor privilege state. Also, the Trace mode (T) bit is negated. For the reset and interrupt exceptions, the interrupt priority mask is also changed to match the level of the interrupt causing the exception.

In the second step, the vector number of the exception is determined. For interrupts other than auto-vectored interrupts, the vector number is obtained by a processor fetch from an external device, classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

The third step is to save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked, using the supervisor stack pointer. The program counter value stacked usually points to the next unexecuted instruction; however, for

bus error and address error, the value stacked for the program counter is unpredictable and may be incremented from the address of the instruction which caused the error. Additional information defining the instruction/operation causing the error is stacked for the bus error and address error exceptions. The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The processor then resumes instruction execution. The instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution are started.

5.1.3 Trap 14 Handler

Traps are instructions which generate exceptions. The TRAP instruction can generate one of 16 exception vectors. Traps are useful for implementing system calls from user programs. The TRAP 14 handler within TUTOR serves this purpose. The TRAP 14 handler permits selected routines from TUTOR to be accessed by the user's target programs. In addition, it allows the user to append his own routines to the TRAP 14 handler and redefine the functions provided.

Up to 255 different functions can be accessed via the TRAP 14 handler. When using the TRAP 14 handler in TUTOR, the number of the desired function is passed to the TRAP 14 handler in the least significant byte of register D7. The handler uses this function number to find the address of the selected routine in a lookup table, and transfers control to that address. Most of the defined functions return to the user's program upon completion; the exceptions are function numbers 229 and 228, which return to TUTOR.

Of the 255 available functions, 127 are reserved by Motorola (numbered 128 through 254). It is suggested, therefore, that the user assign numbers 0 through 127 to user routines.

5.2 TRAP 14 CALLING SEQUENCE

The calling sequence is:

```
MOVE.B   #<function number>,D7
TRAP     #14
```

where <function number> is a number from 0 through 254, which represents the selected function. Calls to functions not defined result in the message 'UNDEFINED TRAP 14'; program control is passed to TUTOR.

The appropriate function number is placed in the least significant byte of register D7 before executing the TRAP instruction. A summary of the defined functions and the corresponding function numbers is shown in Table 5-1.

TABLE 5-1. TRAP 14 Function Summary

FUNCTION	FUNCTION NAME	FUNCTION DESCRIPTION
255	—	Reserved function - end of table indicator.
254	—	Reserved funciton - used to link tables.
253	LINKIT	Append user table to TRAP 14 table.
252	FIXDADD	Append string to buffer.
251	FIXBUF	Initialize A5 and A6 to 'BUFFER'.
250	FIXDATA	Initialize A6 to 'BUFFER' and append string to buffer.
249	FIXDCRLF	Move 'CR', 'LF', string to buffer.
248	OUTCH	Output single character to Port 1.
247	INCHE	Input single character from Port 1.
246	—	Reserved function.
245	—	Reserved function.
244	CHRPRINT	Output single character to Port 3.
243	OUTPUT	Output string to Port 1.
242	OUTPUT21	Output string to Port 2.
241	PORTIN1	Input string from Port 1.
240	PORTIN20	Input string from Port 2.
239	TAPEOUT	Output string to Port 4.
238	TAPEIN	Input string from Port 4.
237	PRCRLF	Output string to Port 3.
236	HEX2DEC	Convert hex value to ASCII encoded decimal.
235	GETHEX	Convert ASCII character to hex.
234	PUTHEX	Convert 1 hex digit to ASCII.
233	PNT2HX	Convert 2 hex digits to ASCII.
232	PNT4HX	Convert 4 hex digits to ASCII.
231	PNT6HX	Convert 6 hex digits to ASCII.
230	PNT8HX	Convert 8 hex digits to ASCII.
229	START	Restart TUTOR; perform initialization.
228	TUTOR	Go to TUTOR; print prompt.
227	OUT1CR	Output string plus 'CR', 'LF' to Port 1.
226	GETNUMA	Convert ASCII encoded hex to hex.
225	GETNUMD	Convert ASCII encoded decimal to hex.
224	PORTIN1N	Input string from Port 1; no automatic line feed.
223-128	—	Reserved.
127-0	—	User-defined functions.

5.3 TRAP 14 FUNCTIONS

There are five groups of functions defined by the TRAP 14 handler. These are:

a. Input/Output single character or character strings to or from I/O ports.

b. Conversion routines:

   Hex to decimal (ASCII format)
   Hex to ASCII - 1, 2, 4, 6, or 8 digits
   ASCII (one digit) to hex
   ASCII formatted hex to hex
   ASCII formatted decimal to hex

c. Buffer control routines.

d. Transfer control to TUTOR with/without performing initialization.

e. Routines to insert additional user functions into TRAP 14 lookup table.

NOTE: The expected convention when using the TRAP 14 handler is independent user and supervisor stacks.

5.3.1 Input/Output Functions

The Input/Output group of TRAP 14 functions includes routines to move information from/to the four available I/O ports to/from memory. They are useful for receiving commands from the terminal and displaying responses at the console and at the printer. Communication with a host is also possible.

There are five input routines which can be called via the TRAP 14 handler. A buffer string can be received from Port 1, 2, or 4. Input is not received from Port 3 because this port is typically connected to a printer. Single character input can also be received from Port 1. The four string input routines — PORTIN1, PORTIN1N, PORTIN20, and TAPEIN — receive input from Port 1, Port 1, Port 2, and Port 4, respectively. ASCII coded strings are typically used although this is not necessary.

The first three routines accept input until the ASCII code for a carriage return ($0D) is received signifying the end of the string. The last routine, TAPEIN, recognizes a line feed ($0A) as the end of string indicator. Because it is used exclusively with S-records, TAPEIN expects the first character of each string to be an ASCII 'S' ($53); characters prior to the 'S' are ignored.

All of the string input routines move characters from the appropriate port to a buffer pointed to by register A6. Before using a TRAP 14 call, the user must, in some cases, initialize parameters other than register D7. For the string input calls, A6 must be initialized to point to the next free location in the buffer where the characters will be stored. Register A5 must point to the start address of the buffer. A comparison is made betwen A5 and A6 each time a character is received, and the buffer size is not allowed to grow larger than 127 bytes (characters).

Upon completion, the routines PORTIN20 and TAPEIN leave A6 pointing to the last character in the buffer. PORTIN1 and PORTIN1N leave A6 pointing to the last character plus one (i.e., the next free location). All incoming bytes are masked to seven bits by ANDing them with the value $7F. Control characters (ASCII codes less than $20) are ignored by PORTIN20 and TAPEIN, while PORTIN1 and PORTIN1N ignore nulls ($00).

Both PORTIN1 and PORTIN1N will echo the characters back out to Port 1. The only difference between these two routines is that PORTIN1 will send both a carriage return and a line feed back to Port 1 upon receipt of only a carriage return. PORTIN1N sends only a carriage return. Table 5-2 summarizes the input functions.

Single character input is available only for Port 1. Using the routine INCHE, a single character is input from Port 1 and transferred to the lowest byte of register D0 (the remainder of D0 is unchanged). No additional parameters are required for this function other than the function number which is passed to the TRAP 14 handler in register D7. Upon completion, D0 contains the received character and A0 has the base address of the serial interface device, the MC6850, associated with Port 1. In addition, register D1 is used by INCHE and is not restored.

There are seven output routines corresponding to the five input functions with two additions. Both string and single character output is provided for Port 3; there are no corresponding input functions. The five string functions are OUTPUT, OUT1CR, OUTPUT21, PRCRLF, and TAPEOUT corresponding to Ports 1, 1, 2, 3, and 4, respectively;. Functions which send single character output to Ports 1 and 3 are named OUTCH and CHRPRINT, respectively.

In all of the string output routines, register A5 is used to point to the beginning of the string, and register A6 points to the byte immediately following the last byte in the string. The string that is outputted consists of all the bytes between the address contained in A5 and the one in A6, including the byte pointed to by A5 but not including the byte to which A6 points. In each case, with one exception, A5 is pointing to the last byte in the output string plus one when the output function is complete. The exception, PRCRLF, leaves register A5 unchanged.

Strings are generally collections of ASCII encoded characters. In the case of Port 4 (tape interface), the string are also generally in S-record format. OUTPUT, OUT1CR, and OUTPUT21 will echo the character being sent to their respective ports to Port 3 (printer) if it is attached (using the PA command). Any character sent to Port 3, by these functions or by PRCRLF, must be a printable ASCII character or a carriage return or line feed; all other characters are replaced by the ASCII code for a period ($2E) before they are sent to the printer. Printable characters include all ASCII codes between and including $20 through $7F.

The major difference between OUTPUT and OUT1CR is that OUT1CR appends a carriage return and line feed to the end of the string. None of the other output string functions append the CRLF to the end of the string. OUT1CR also destroys the contents of registers D0 and D1 and moves the base address of the Port 1 serial device into register A0. Additionally, the contents of register D0 are destroyed by PRCRLF. Refer to Table 5-3 for a summary of parameters required and registers used by these functions.

Two single character output functions can be called; OUTCH sends a character to Port 1 echoing it to Port 3 if it is attached, and CHRPRINT sends a character directly to Port 3. The character is generally ASCII coded and must be put in the lowest byte of register D0 before the output function is called. As with the string functions, any character sent to Port 3 must be a printable character or a carriage return or line feed.

OUTCH does not restore the contents of either D0 or D1. It initializes A0 to the base address of the Port 1 serial I/O device.

TABLE 5-2. Input Functions

FUNCTION NAME	FUNCTION NUMBER	PORT NUMBER	OUTPUT TYPE	TERMINATION CHARACTER	REGISTER A6 INITIAL	REGISTER A6 FINAL	REGISTER A5 INITIAL VALUE	OTHER REGISTERS FINAL VALUE
PORTIN1	241	1	String	CR = $0D	Next Free Loc.	Buf. End + 1	Buffer Start	---
PORTIN1N	224	1	String	CR = $0D	Next Free Loc.	Buf. End + 1	Buffer Start	---
PORTIN20	240	2	String	CR = $0D	Next Free Loc.	Buf. End	Buffer Start	---
TAPEIN	238	4	String	LF = $0A	Next Free Loc.	Buf. End	Buffer Start	---
INCHE	247	1	Single Char.	---	---	---	---	D0 = input char. A0 = ACIA #1 base address D1 destroyed

TABLE 5-3. Output Functions

FUNCTION NAME	FUNCTION NUMBER	PORT NUMBER	OUTPUT TYPE	REGISTER A6 INITIAL VALUE	REGISTER A5 INITIAL	REGISTER A5 FINAL	OTHER REGISTERS
OUTPUT	243	1	String	Buf. End + 1	Buf. Start	Buf. End + 1	---
OUT1CR	227	1	String	Buf. End + 1	Buf. Start	Buf. End + 1	D0, D1 destroyed A0-ACIA #1 base address
OUTPUT21	242	2	String	Buf. End + 1	Buf. Start	Buf. End + 1	---
PRCRLF	237	3	String	Buf. End + 1	Buf. Start	Buf. Start	D0 destroyed
TAPEOUT	239	4	String	Buf. End + 1	Buf. Start	Buf. End + 1	---
OUTCH	248	1	Single Char.	---	---	---	D0, D1 destroyed A0-ACIA #1 base address
CHRPRINT	244	3	Single Char.	---	---	---	---

5.3.2 Conversion Functions

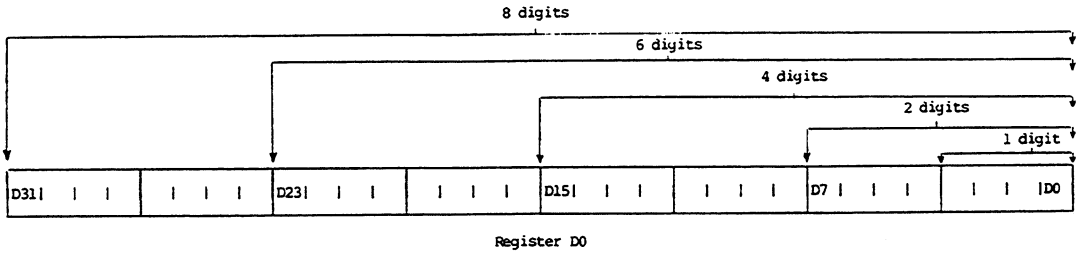
The conversion functions can be divided into two groups:

- Hex conversion routines which convert hexadecimal numbers to ASCII encoded decimal or ASCII encoded hex.
- ASCII conversion routines which convert ASCII encoded numbers to either decimal or hexadecimal.

Both groups are needed to interface with the various peripherals connected to the I/O ports. Most require all characters to be ASCII encoded.

There are five different routines to convert hex numbers to ASCII encoded hex digits, which are then transferred to a buffer. One, two, four, six, or eight hex digits can be converted to ASCII, depending on which of the five hex to ASCII conversion functions — PUTHEX, PNT2HX, PNT4HX, PNT6HX, PNT8HX — is used. Register A6 must point to the buffer where the character(s) will be stored. A6 is incremented by 1, 2, 4, 6, or 8, depending on the number of digits converted.

The hex number to be converted is passed to the conversion routine in register D0 (right-justified using four bits per digit); anywhere from four to 32 bits of D0 may be required, depending on the number of digits to be converted.



Each digit requires four bits; up to eight digits can be represented. The most significant digit is converted and placed in the buffer first, followed by the other digits in descending order. All five routines destroy the contents of register D0. The contents of registers D1 and D2 may or may not be destroyed, depending on the function (see Table 5-4 for more details).

HEX2DEC converts the 8-digit hex number found in register D0 to the equivalent decimal number. This decimal number is converted to ASCII-encoded digits and placed in the buffer. All leading zeros are suppressed when the number is transferred to the buffer. The final value of register A6 (the buffer pointer) is therefore unpredictable; its value depends on the number of non-zero digits in the number. Specific information on this function is shown in Table 5-4.

The six conversion functions discussed thus far convert hex numbers to ASCII. The last three conversion functions are used to convert ASCII characters to hex or decimal digits. GETNUMA and GETNUMD take ASCII encoded numbers from a buffer pointed to by register A5 and convert them to hex and decimal digits, respectively. If the number of digits is too large to be represented in 32 bits or if an inappropriate digit is received (i.e., a digit which is too large for the base or is not a digit at all), the conversion will be aborted and an error message displayed at the terminal. Otherwise, the conversion will continue until all digits in the buffer have been processed. Register A6 should point one byte past the last digit to indicate the end of the buffer. The resultant value is returned in register D0; all 32 bits are used. A5 is left pointing one byte past the last digit.

TABLE 5-4. Hex Conversion Routines

FUNCTION NAME	FUNCTION NUMBER	NUMBER OF HEX DIGITS CONVERTED	PORTION OF D0 USED FOR DIGIT(S)	FINAL VALUE OF A6	REGISTERS USED AND NOT RESTORED
PUTHEX	234	one	bits 3-0	A6 init + 1	D0
PNT2HX	233	two	bits 7-0	A6 init + 2	D0,D2
PNT4HX	232	four	bits 15-0	A6 init + 4	D0,D1,D2
PNT6HX	231	six	bits 23-0	A6 init + 6	D0,D1,D2
PNT8HX	230	eight	bits 31-0	A6 init + 8	D0,D1,D2
HEX2DEC	236	eight	bits 31-0	*	D0

* unpredictable — depends on original hex value

The final conversion function, GETHEX, converts the ASCII character in the lowest byte of register D0 to hex and returns the hex number in the same byte. If the ASCII code does not represent a valid hex digit, an error message is generated. The ASCII conversion routines are summarized in Table 5-5; all registers not included in the table are unchanged.

5.3.3 Buffer Control Functions

The buffer control functions are useful in conjunction with the various input and output functions which are available. The control functions are used primarily to initialize buffer pointers and to move ASCII strings into an output buffer.

FIXBUF initializes registers A5 and A6 to point to 'BUFFER'. This is a buffer within the system RAM and is used by TUTOR. No other registers are altered and no parameters are required. FIXBUF can be used to initialize A5 and A6 prior to calling the string input functions. Usage of this buffer while tracing or with breakpoints will produce erroneous results.

FIXDADD, FIXDATA, and FIXDCRLF are all used to transfer an ASCII string to a buffer and are summarized in Table 5-6; all registers not included in the table are unchanged. The first two, FIXDADD and FIXDATA, move a string pointed to by register A5 into the buffer. FIXDADD requires that register A6 point to the buffer, while FIXDATA always moves the string to the location in system RAM called 'BUFFER'. The ASCII end of transmission character (EOT = $04) is used to indicate the end of the string. It must be the last character in the string, but is not moved to the buffer. Registers A5 and A6 are left pointing to 'BUFFER' and the buffer end + 1, respectively.

FIXDCRLF is quite similar to FIXDATA — a string pointed to by register A5 is moved to 'BUFFER'. The difference is that FIXDCRLF transfers the ASCII code for a carriage return and line feed to the beginning of the buffer before transferring the string. The termination character ($04) and parameters returned are the same.

TABLE 5-5. ASCII Conversion Routines

FUNCTION NAME	FUNCTION NUMBER	REGISTER A5		REGISTER A6		REGISTER D0	
		INITIAL	FINAL	INITIAL	FINAL	INITIAL	FINAL
GETNUMA	226	buffer start	buffer end + 1	buffer end + 1	---	---	hex value 32 bits
GETNUMD	225	buffer start	buffer end + 1	buffer end + 1	---	---	hex value 32 bits
GETHEX	235	---	---	---	---	ASCII char 8 bits	hex value 4 bits

TABLE 5-6. Buffer Control Functions

FUNCTION NAME	FUNCTION NUMBER	FUNCTION DESCRIPTION	PARAMETERS REQUIRED	PARAMETERS RETURNED
FIXDADD	252	Append a string to buffer; $04 signifies end of string	A5 - String start A6 - Next location in buffer	A5 - 'BUFFER' A6 - Buffer end + 1
FIXDATA	250	Same as FIXDADD but initializes A6 to 'BUFFER' before moving data	A5 - string start	A5 - 'BUFFER' A6 - Buffer end + 1
FIXBUF	251	Initializes A5 and A6 to 'BUFFER'	None	A5 - 'BUFFER' A6 - 'BUFFER'
FIXDCRLF	249	Move CR, LF, string to 'BUFFER'; $04 signifies end of string; initializes A6 to 'BUFFER' before moving data	A5 - string start	A5 - 'BUFFER' A6 - Buffer end + 1

5.3.4 Transfer Control to TUTOR

Using TRAP 14, control can be transferred from a target program to TUTOR in two ways. One path uses the TRAP 14 function START. After control is passed to TUTOR, the restart initialization procedure is executed. The stack pointer, register A7, is initialized to 'SYSSTACK'. The exception vectors, located in the lower portion of the system RAM, are initialized. A portion of the upper system RAM is zeroed. The status register is set to $2700 — interrupt mask level 7 and supervisor state. The prompt is then sent to Port 1.

Using the TRAP 14 function TUTOR, the other path performs only a portion of the initialization described above. The stack pointer and status register are initialized and the prompt is displayed. See Table 5-7.

TABLE 5-7. Transfer Control to TUTOR

FUNCTION NAME	FUNCTION NUMBER	DESCRIPTION	REGISTERS AFFECTED
START	229	Restart TUTOR Perform Initialization	Status Register=$2700 A7='SYSSTACK', Program Counter=0
TUTOR	228	Go to TUTOR Print Prompt	Status Register=$2700 A7='SYSSTACK'

NOTE: Calling START or TUTOR from the user state will result in a privilege violation because they write to the status register.

5.3.5 Inserting Additional Functions

User-defined functions can also be called using the TRAP 14 handler. The TRAP 14 handler uses the function number and a lookup table to determine the address of the selected function. User functions are included by inserting additional lookup tables. The format for entries in these tables is $UUSSSSSS, where UU is the function number in hex — $0 through $7F for user-defined functions — and SSSSSS is the starting address of the function in hex. Each entry in the table requires one long word (32 bits).

The table is inserted quite simply; the TRAP 14 function LINKIT, function number 253, performs the insertion. Register A0 must point to the lookup table to be inserted when the TRAP instruction is executed. The new table is, in effect, placed in front of the old lookup table. However, since the new table is not addressed immediately in front of the old table (i.e., old table is in ROM and new table is in RAM), a link must be provided in the new table to connect it with the old table. This is accomplished by making the starting address of the old table preceded by $FE the last entry in the new table. The format is $FETTTTTT, where TTTTTT is the pointer to the old table. This is easily accomplished because LINKIT returns the required long word ($FETTTTTT) in register A0. The target program should store this value at the end of the new lookup table immediately after executing the TRAP 14 instruction. The following is an example of the procedure used to link a user-defined table. Location NEWTBL is the beginning of the user table and location ENDTBL is the end of the user table. After executing the TRAP instruction, the pointer to the old table must be saved at the end of the new table.

NOTE: Labels NEWTBL and ENDTBL are used here for clarity but will not, of course, be accepted by the interactive assembler.

```
        LEA     NEWTBL,A0       Register A0 points to new table
        MOVE.B  #253,D7         LINKIT
        TRAP    #14
        MOVE.L  A0,ENDTBL       A0 contains $FETTTTTT
          .                     where TTTTTT points to old table
          .
          .

NEWTBL  DC.W    $UUSS
        DC.W    $SSSS
        DC.W    $UUSS
        DC.W    $SSSS
          .
ENDTBL  DC.W    $XXXX           Link to old table will be stored here
        DC.W    $XXXX
```

LINKIT is summarized in Table 5-8. Additional tables can be inserted in the same way.

Each user-defined function requires a user-written software routine located somewhere within the user RAM. These routines should use an RTS (return from subroutine) instruction, not an RTE (return from exception) instruction, to return to the calling program. The TRAP 14 handler jumps to the user software routine, leaving only the program counter on the stack. The RTE instruction expects both the status register and the program counter to be on the stack. NOTE: The user must make sure that the stack pointers are pointing to locations within the user RAM and that the two stacks do not overlap before executing a TRAP 14 instruction.

TABLE 5-8. Inserting Additional Functions

FUNCTION NAME	FUNCTION NUMBER	DESCRIPTION	REQUIRED PARAMETERS	REGISTERS AFFECTED
—	255	Reserved - End of Table Indicator $FFXXXXXX - must be last entry in last table	—	—
—	254	Reserved - Use to link one table to another	—	—
LINKIT	253	Insert User Table	A0 - start add of table to be inserted	D0 - destroyed A0 - start add of old table

Because function numbers are compared with entries in the user table(s) first, it is possible to redefine function numbers which have been defined by TUTOR (i.e., 128 through 253; function numbers 254 and 255 cannot be redefined). For example, if a user does not have a host system and has nothing else tied to Port 2, that user may not require the Port 2 I/O functions numbered 242 and 240. The user can redefine the functions associated with these numbers by placing the appropriate function number and the address of the new routine ($UUSSSSSS) in the user function table. The user table will be searched first and the entry in the other table will be ignored.

Function number 255 is described as a reserved function in Table 5-1. Although this function is not called by target programs, it nevertheless performs a useful function as an end-of-table indicator. All 255 available function numbers may not be assigned to a function and, therefore, will not be included in the lookup table(s). Thus, it is possible to attempt to call a nonexistent function by using an unassigned function number. In such a situation, a match will not be found in the table(s). Whatever follows the table(s) in memory — program, data, random bytes — will automatically be used to extend the table in an attempt to find a match. A bus or address trap error is the usual end point of this sequence, although significant damage may occur prior to the trap error.

To avoid these problems, the end-of-table function number should always be the last entry in the last lookup table. Usually this is the table provided by TUTOR. An address is not required for the end-of-table entry; only the function number is needed. The format is $FFXXXXXX, where the X's represent 'don't care' characters. This entry is included as the last long word of the TRAP 14 lookup table provided by TUTOR. When the end-of-table entry is reached, the message 'UNDEFINED TRAP 14' is sent to Port 1 and control is transferred to TUTOR.

CHAPTER 6

SYSTEM INPUT/OUTPUT

For system I/O, the MC68000 Educational Computer Board provides two serial communications ports, a printer compatible parallel port, an audio tape interface, and programmable timer. Chapter 6 provides a detailed discussion of each of these areas.

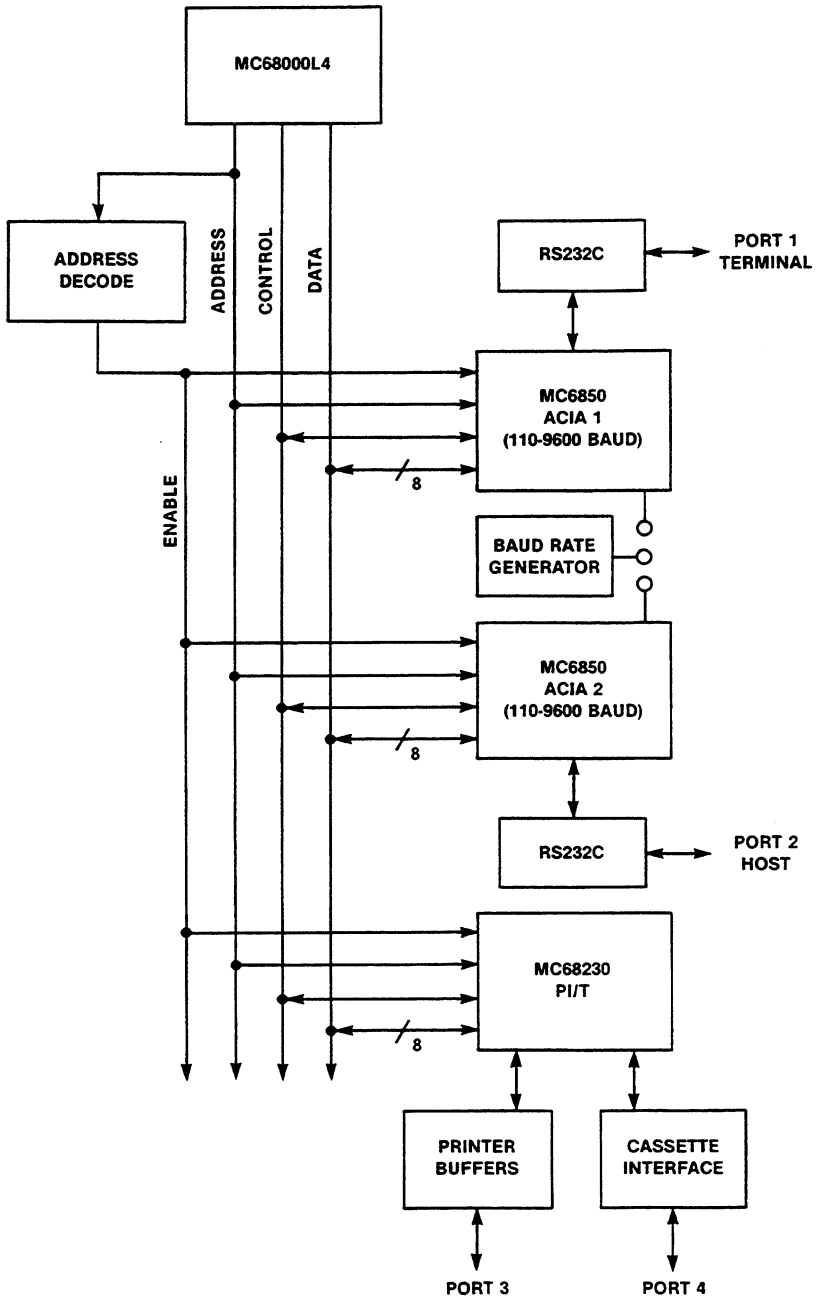


FIGURE 6-1. System I/O Block Diagram

CHAPTER 6

SYSTEM INPUT/OUTPUT

6.1 INTRODUCTION - INPUT/OUTPUT LSI DEVICES

Figure 6-1 shows the block diagram of the I/O structure for the MC68000 Educational Computer Board. Only two types of MOS LSI drives are used to interface various peripherals to the MC68000. The first device is the MC6850 Asynchronous Communications Interface Adapter (ACIA) which is a member of the M6800 family of peripheral parts. Two MC6850's are used to provide the serial communications channels of Port 1 and Port 2.

The second device type is the MC68230 Parallel Interface and Timer (PI/T) which is an M68000 family part. The Port 3 printer (parallel I/O) interface, the Port 4 cassette tape interface, and a programmable timer are all provided by this device.

6.1.1 MC6850 Asynchronous Communications Interface Adapter

The MC6850 Asynchronous Communications Interface Adapter (ACIA) provides the data formatting and control to interface serial asynchronous data to an M6800 synchronous parallel bus. Parallel data received from the system bus is transmitted asynchronously along with start and stop bits and parity information. Going the other way, the received serial bit stream is stripped of the start, stop, and parity bits before it is transferred to the parallel bus. Figure 6-2 shows the MC6850 block diagram. Active low signals are specified by an * following the signal name.

Information transfer and control is accomplished via four 8-bit registers within the ACIA. Two of the registers are read only registers and two are write only registers. Only one register select input is required to address all four registers; the read/write (R/W) input provides the other select control line. These registers allow programming of variable word lengths, clock division ratios, transmit control, receive control, and interrupt control.

Two bytes within the address map are needed for each of the two ACIA's present on the board. One register can be loaded and another read at each byte address. Each device has an 8-bit data interface, and the system is configured with one ACIA tied to the lower eight bits of the MC68000 data bus (D0 through D7) and the second ACIA tied to the upper eight bits (D8 through D15) of the bus. In this manner, both odd and even address locations are utilized within the address space.

As was noted, the MC6850 is a synchronous bus interface which requires that a read or write to any of its registers must by synchronized to its E clock. The E clock for the educational board is supplied by the MC68000L4 and is one-tenth (1/10) the clock frequency of the MC68000L4; that is, E is a 400 KHz clock. To interface with a synchronous device, the MC68000 can modify its normally asynchronous bus cycle to meet a synchronous cycle requirement.

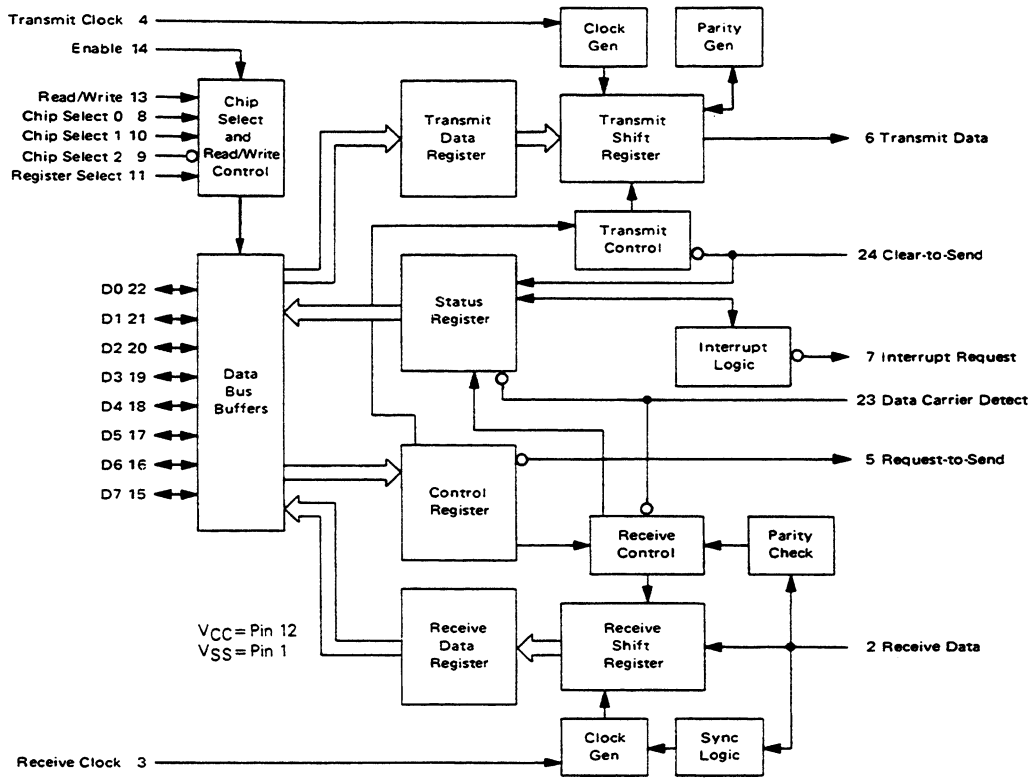


FIGURE 6-2. MC6850 ACIA Block Diagram

Hardware external to the MC68000 asserts the Valid Peripheral Address (VPA*) line whenever the MC6850's address space is accessed. The MC68000 then synchronizes itself to the E clock and asserts Valid Memory Address (VMA*). VMA* goes to the ACIA chip select logic and enables the ACIA's. When E clock is high, data is transferred. When E goes low again, the processor negates VMA* and ends the cycle.

The MC6850's are the only devices that use the MC68000's on-board synchronous interface. For more information on the MC6850, consult its data sheet.

6.1.2 MC68230 Parallel Interface and Timer

The MC68230 Parallel Interface and Timer (PI/T) provides two double buffered parallel interface ports, eight general purpose I/O pins, and a 24-bit programmable timer. The ports and the timer compose two independent sections within the PI/T. The port section consists of two 8-bit ports, Port A and Port B, four handshake lines, H1, H2, H3, and H4, and a third 8-bit port, Port C. Port C, however, is a dual function port. Six of the eight pins which make up Port C have a second function associated with the timer, interrupts, or direct memory access (DMA) requests. The MC68230 block diagram is shown in Figure 6-3.

Ports A and B can be operated individually or combined as one 16-bit wide parallel port. The parallel ports will operate in unidirectional or bidirectional modes. In the unidirectional modes, data direction registers within the PI/T determine whether the port pins are inputs or outputs. In the bidirectional modes, the data direction registers are ignored and the direction is determined dynamically by the state of the four handshake pins. These programmable handshake pins provide an interface flexible enough for connection to a wide variety of low, medium, or high speed peripherals.

The other independent section within the PI/T is the timer. The timer consists of a 24-bit synchronous presettable down counter and a 5-bit prescaler. Use of the prescaler is optional. The down counter is clocked either by the output of the prescaler or by an external timer input pin (one of the Port C dual function pins). The prescaler, in turn, is clocked by either the system clock (CLK pin) or the external timer input pin. The external timer input is brought out on the J2 edge connector. The PI/T must be programmed by the user to utilize the external clock. The MC68230 timer can generate periodic interrupts, a square wave, or a single interrupt after a programmed time period. It can also be used for elapsed time measurement.

The MC68230 has 23 registers that can be addressed from the system bus. The data bus interface is eight bits wide and is tied to the lower eight bits (D0 through D7) of the system bus. Because of this, byte operations are valid only on <u>odd</u> addresses and accesses to upper bytes; <u>even</u> addresses are invalid and result in a bus trap error. The MC68230 occupies a 64-byte address space (32 words) although only 23 odd addresses are used. Note that the DTACK* will be returned anytime the other nine odd locations are accessed. These locations read as zeros, and writes to them are ignored.

The MC68230 PI/T is a complex device. This description is very short and the reader is encouraged to become more knowledgeable by referring to the MC68230 Data Sheet.

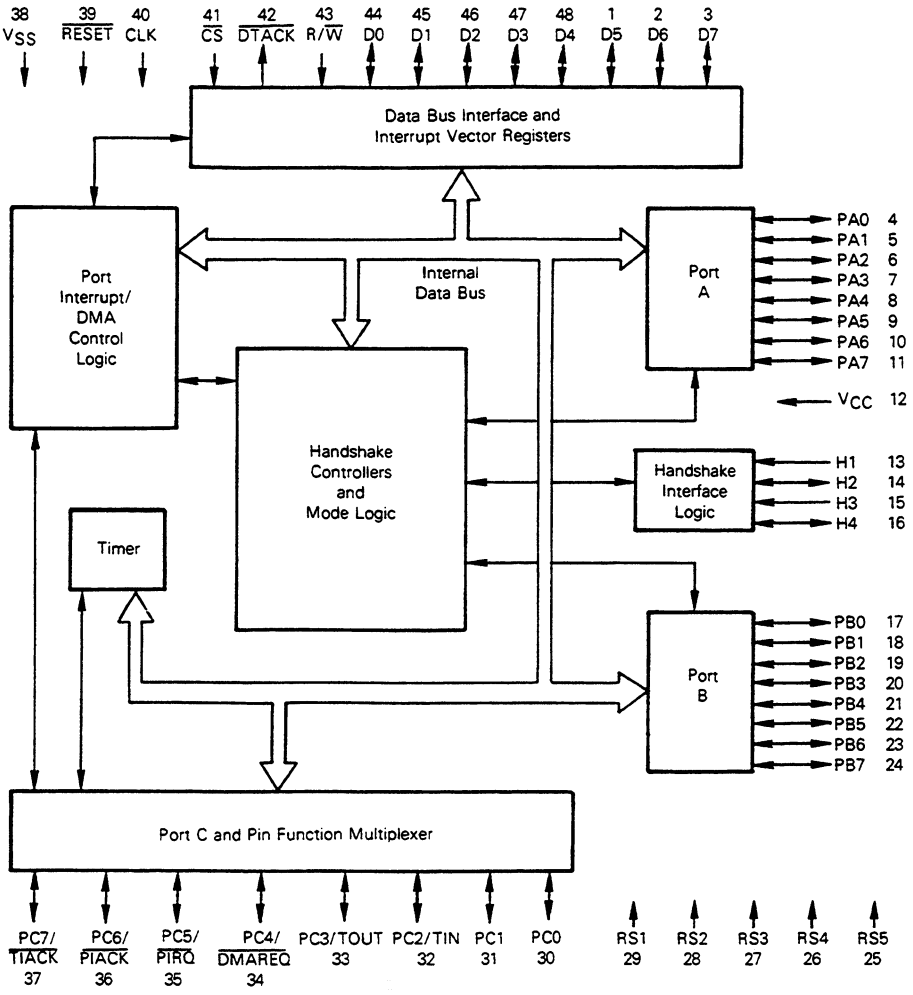


FIGURE 6-3. MC68230 PI/T Block Diagram

6.1.3 I/O Device Address Map

Page $010000-$01FFFF within the educational computer address map is reserved for the input/output devices. The registers contained within the PI/T and the ACIA's are all accessed within this page. The device addresses are not fully decoded within the page; therefore, each device can be accessed at several different areas. Refer to paragraph 7.2.2 and Table 7-1 for the address map of the entire system.

Table 6-1 shows the address map for the MC68230 PI/T, which contains 23 register locations. Within page $010000, the PI/T can be accessed any time address line A6 = 0. The MC68230 is enabled by a signal that is decoded from the upper eight address bits, A6 and LDS*. In turn, the MC68230 registers are selected via address lines A1 through A5. The PI/T responds to the following bit pattern:

```
0000 0001 XXXX XXXX X0AA AAAUDS*
                       54 321
```

The device is connected to the lower eight data lines and can be accessed only at odd byte addresses.

Table 6-2 shows the address map for the MC6850 ACIA's which each contain four registers (two are read only and two are write only). Within page $010000, the ACIA's can be accessed any time address line A6 = 1. The ACIA's are enabled by a signal that is decoded from the upper eight address bits. After being enabled, the ACIA's then use address lines A6 and A1 as well as the data strobes (LDS* and UDS*) to select registers. The ACIA's respond to the following bit pattern:

```
                                LDS*
0000 0001 XXXX XXXX X1XX XXAUDS*
                               1
```

One ACIA is accessed on the lower eight bits of the data bus, and the other is accessed on the upper eight bits of the data bus. The data strobes determine which device is selected.

TABLE 6-1. MC68230 PI/T Address Map

ADDRESS($)	PI/T REGISTER
010001	Port General Control Register (PGCR)
010003	Port Service Request Register (PSRR)
010005	Port A Data Direction Register (PADDR)
010007	Port B Data Direction Register (PBDDR)
010009	Port C Data Direction Register (PCDDR)
01000B	Port Interrupt Vector Register (PIVR)
01000D	Port A Control Register (PACR)
01000F	Port B Control Register (PBCR)
010011	Port A Data Register (PADR)
010013	Port B Data Register (PBDR)
010015	Port A Alternate Register (PAAR)
010017	Port B Alternate Register (PBAR)
010019	Port C Data Register (PCDR)
01001B	Port Status Register (PSR)
010021	Timer Control Register (TCR)
010023	Timer Interrupt Vector Register (TIVR)
010027	Counter Preload Register High (CPRH)
010029	Counter Preload Register Middle (CPRM)
01002B	Counter Preload Register Low (CPRL)
01002F	Count Register High (CNTRH)
010031	Count Register Middle (CNTRM)
010033	Count Register Low (CNTRL)
010035	Timer Status Register (TSR)

NOTE:

The PI/T address decode is redundant within page $010000. The PI/T can be accessed any time address line A6 = 0 within the page.

TABLE 6-2. MC6850 ACIA Address Map

ADDRESS ($)	ACIA REGISTER
010040	ACIA 1 - Control Register (Write Only) Status Register (Read Only)
010041	ACIA 2 - Control Register (Write Only) Status Register (Read Only)
010042	ACIA 1 - Transmit Data Register (Write Only) Receive Data Register (Read Only)
010043	ACIA 2 - Transmit Data Register (Write Only) Receive Data Register (Read Only)

NOTE:

The ACIA address decode is redundant within page $010000. The ACIA's can be accessed any time address line A6 = 1 within the page.

6.2 SERIAL COMMUNICATIONS - PORT 1 AND PORT 2

Two serial ports are provided on the MC68000 Educational Computer Board. Port 1 is normally connected to a terminal, and Port 2 provides a link to a host computer. Figure 6-4 shows a functional schematic diagram of the serial port logic. Two MC6850 ACIA's provide the interface between the computer's parallel data bus structure and the serial communications lines. Data taken from the system bus is serially transmitted and serial data received is read to the bus. The MC6850's also provide data formatting and error checking.

The serial interface is a subset of the E.I.A. RS-232C standard. Buffers provide the proper drive and levels to interface the MOS and TTL devices to RS-232C. Appendix C discusses RS-232C in more detail.

The functional mode of each ACIA is programmed via its control register (one of four on-board registers). The programmable control register sets variable word lengths, clock division ratios, transmit control, receive control, and interrupt control. The control registers are programmed at system initialization for the selected operational modes.

6.2.1 ACIA Control Register

The ACIA control register is eight bits wide and determines the operational mode of the device. Table 6-3 lists the control bits and their corresponding states. In the normal operational mode of the educational computer, the control register is programmed with $15 which corresponds to:

- receive and transmit interrupts disabled
- 8-bit words with one stop bit, no parity
- divide-by-16 clock ratio

Some comments can be made concerning this operational mode:

a. The educational computer hardware supports ACIA interrupts (see paragraph 6.6) but the TUTOR firmware does not contain service routines for these interrupts. If the user desires to use these interrupts, he must provide the service routines.

b. The ACIA can be programmed to transmit parity bits. However, the TUTOR firmware does not check received data for parity errors. The parity error bit in the status registers is ignored.

c. Request to Send ($\overline{RTS}$) is a control line on the ACIA which allows the MPU to control a peripheral or modem by writing to the control register. $\overline{RTS}$ must be low when the educational computer is operating in its normal mode. Bringing $\overline{RTS}$ high on the Port 1 ACIA causes the board to operate in the Transparent Mode (see paragraph 6.2.6).

d. The divide-by-16 clock ratio is selected because of considerations discussed in paragraph 6.2.2.

To change the control register characters, the Port Format (PF) command (paragraph 3.5.21) can be used. The PF1 command is used to alter Port 1 (ACIA1) and the PF2 command is used to alter Port 2 (ACIA2). The user must be aware of the requirements and restrictions when altering these registers.

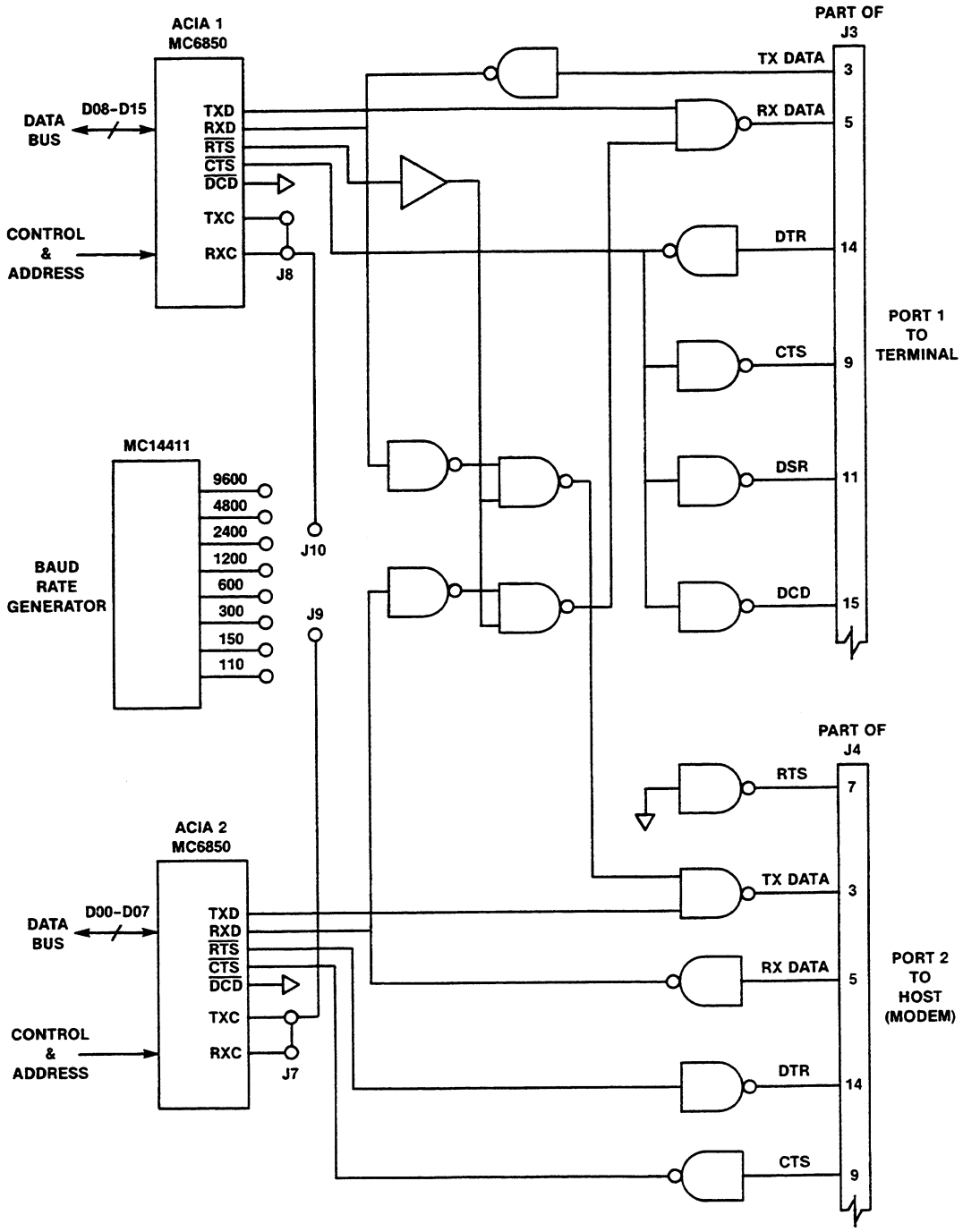


FIGURE 6-4. Serial Communications Ports Functional Schematic Diagram

TABLE 6-3. ACIA Control Register Bits

BIT 7 RECEIVE INTERRUPT ENABLE	BIT 6 & BIT 5 TRANSMIT CONTROL	BIT 4, BIT 3, & BIT 2 WORD SELECT	BIT 1 & BIT 0 COUNTER DIVIDE SELECT
0-Disabled 1-Enabled	00-$\overline{\text{RTS}}$=low Transmit Interrupt Disabled	000 - 7 bits, even parity, 2 stop bits	00 - ÷ 1 01 - ÷ 16
	01-$\overline{\text{RTS}}$=low Transmit Interrupt Enabled	001 - 7 bits, odd parity, 2 stop bits	10 - ÷ 64 11 - Master Reset
	10-$\overline{\text{RTS}}$=high Transmit Interrupt Disabled	010 - 7 bits, even parity, 1 stop bit	
	11-$\overline{\text{RTS}}$=low Transmit Interrupt Disabled Transmits a Break Level on Transmit Data Output	011 - 7 bits, odd parity, 1 stop bit	
		100 - 8 bits, no parity, 2 stop bits	
		101 - 8 bits, no parity, 1 stop bit	
		110 - 8 bits, even parity, 1 stop bit	
		111 - 8 bits, odd parity, 1 stop bit	

6.2.2 Baud Rates

The serial baud rates are jumper selectable as discussed in paragraphs 2.2.4 and 2.5.2.1. However, the clock divide ratio as selected by the control register of the ACIA affects the serial baud rate. The MC14411 Baud Rate Generator generates clocks which are actually 16 times higher in frequency than the desired serial baud rates. Under normal operation with the ACIA programmed for divide-by-16 clock ratio, the serial baud rates specified for the board result. Other non-standard clock rates could be generated.

Using a clock which is 16 times the serial bit rate allows the ACIA to synchronize the clock with the incoming serial data stream. If the ACIA clock frequency were equal to the serial bit rate, the ACIA could not synchronize the clock and the data.

6.2.3 TUTOR Firmware I/O Drivers

As previously stated, the ACIA's are programmed at system initialization. The I/O firmware drivers can then be used to control communication using the serial ports. The drivers for the terminal and host ports are very similar. Characters to be transmitted are converted to ASCII-coded bytes and are placed in a RAM buffer. Pointers to the beginning and end of the buffer are saved. The buffer is transmitted one byte at a time by writing each byte into the appropriate ACIA Transmit Data Register. The ACIA inserts the previously selected start, stop, and parity bits and performs the parallel to serial conversion.

The Transmit Data Register is double-buffered; that is, it can accept a second byte to be transmitted while it is actually transmitting a first byte. Bit 1 of the ACIA Status Register (TDRE) is polled by the processor to determine when the data has been transferred from the Transmit Data Register so that new data may be entered while the old data is being transmitted serially. An interrupt can also be generated if the transmitting interrupt is enabled.

The maximum rate at which the processor can enter data is a function of the serial baud rate and the number of bits transmitted for each byte (including start, stop, and parity bits):

$$\text{Update rate (bytes/second)} = \frac{\text{baud rate (bits/second)}}{\text{number of bits/byte (bits/byte)}}$$

As an example, for 9600 baud and eight data bits with one stop bit and one start bit, the maximum update rate is 960 bytes/second or approximately one byte every millisecond.

The input driver routines monitor bit 0 (RDRF) of the ACIA Status Register to determine when a new byte of data has been received. As above, this bit is polled by the processor; however, an interrupt can be generated. The ACIA removes the parity, start, and stop bits, sets the appropriated status bits, and performs the serial-to-parallel conversion. The received data is typically encoded ASCII, and a carriage return signifies the end of the input string. As the incoming data is received, the processor stores it in a buffer. After the carriage return is received, processing begins on the data stored in the buffer.

6.2.4 Port 1 Terminal Interface

ACIA1 (U13) is used as the terminal interface and is connected to the upper half (D08-D15) of the system bus. As previously described in Table 6-2, its registers reside at even addresses $010040 and $010042 within the memory map.

The RS-232C interface of Port 1 (J3) appears as a modem to the terminal connector. Referring again to Figure 6-4, six signal lines are supported through RS-232C buffers. The ACIA interfaces to three of these including transmit data (TX DATA), receive data (RX DATA), and data terminal ready (DTR). Some terminals require other lines to be present including clear to send (CTS), data set ready (DSR), and data carrier detect (DCD). These signals are driven back to the terminal; however, they are merely a connection to DTR which shows that the terminal is attached and ready.

As part of the power-up and reset firmware routines, the ACIA is reset by setting control register bits 0 and 1. This ensures that the device is master reset before selecting the operating configuration. This bus-programmed master reset also releases the internal reset that occurs at power-on. The programmed reset must be applied prior to operating the ACIA.

To initialize the ACIA, a $15 is written to the control register. Information can now be sent or received. As previously described, the driver routines interface with the device to transmit and receive data.

One special feature of Port 1 is that the processor also monitors the framing error status bit (FE) of ACIA1. Detection of this error form shows activation of the BREAK key, which causes the processor to abort the present activity and return to TUTOR.

6.2.5 Port 2 Host Interface

ACIA2 (U12) forms the host interface and is connected to the lower half (D00-D07) of the system bus. Its registers reside at odd addresses $010041 and $010043 within the memory map.

The RS-232C interface of Port 2 (J4) appears as a "terminal" to the host or modem connector. Five signal lines are supported including TX DATA, RX DATA, RTS, DTR, and CTS.

ACIA2 is reset and initialized similar to ACIA1. Operation for transmitting and receiving data occurs in a similar manner.

While transmitting or receiving data via Port 2, TUTOR also monitors Port 1 for the BREAK key. This allows the user to abort a function which uses Port 2 communications.

6.2.6 Transparent Mode

As described in paragraph 3.5.23, the Transparent Mode (TM) command within TUTOR connects serial Ports 1 and 2 together. In this manner, the user terminal appears connected directly to the host computer, bypassing the MEX68KECB.

To enter this mode, the Request To Send ($\overline{RTS}$) pin of the ACIA1 is brought high by writing a 10 into bits 6 and 5 of the control register. The RTS line gates the transmit and receive data paths so that the serial ports are tied together; that is, the incoming data line of Port 1 is gated through to the outgoing line of Port 2, and the incoming data line of Port 2 is gated through to the outgoing data line of Port 1. ACIA1 continues to receive incoming data from Port 1.

In the transparent mode, the educational board is not directly involved in data transmission between the terminal and the host. The ECB monitors the data transmission from the terminal and restores normal operation upon detecting a predetermined exit character. The character is selected with the TM command.

6.3 PARALLEL I/O PORT3 - PRINTER INTERFACE

Port 3 of the MC68000 Educational Computer is a parallel I/O interface and is configured to support a Centronics type printer. The MC68230 PI/T is used to provide the register and bus interface required for this I/O. The MC68230 supports a wide range of operating modes through programming of its 23 internal registers, however, this discussion focuses on this particular application. Refer to the MC68230 Data Sheet for a description of other modes.

6.3.1 Signal Line Configuration

Figure 6-5 shows the MC68230 and the buffer devices used on Port 3. The parallel interface consists of 8 data lines buffered as outputs (PA0 through PA7) which are associated with handshake lines H1 and H2, and eight unbuffered data lines (PB0 through PB7) which can be used as bidirectional lines and are associated with handshake lines H3 and H4.

The signal line designations used for connector J1 are those corresponding to the Centronics interface. The 8 buffered output data lines are PD0 through PD7. Handshake line H2 is also buffered as an output strobe and is called DATA STROBE*. Handshake line H1 is an unbuffered input strobe and is designated ACKNOWLEDGE*.

Only three of the unbuffered data lines (PB0 through PB2) are used in the printer interface. Signals from the printer indicate:

a. SELECT - when the printer is selected.

b. PAPER OUT - when the printer is out of paper.

c. BUSY - when the printer is busy.

Any of these conditions is valid when the signal is high. The printer is unable to receive information when PAPER OUT or BUSY is activated and when SELECT is not activated.

The additional handshake lines H3 and H4 are also used. H3 is the FAULT* input from the printer which indicates a printer fault condition such as out of paper or deselect. This input is not monitored by the TUTOR firmware because the individual conditions are used as shown above. Finally, output handshake line H4 is buffered and drives signal INPUT PRIME*. When INPUT PRIME* is driven low, the input buffer within the printer is cleared and the printer logic is initialized.

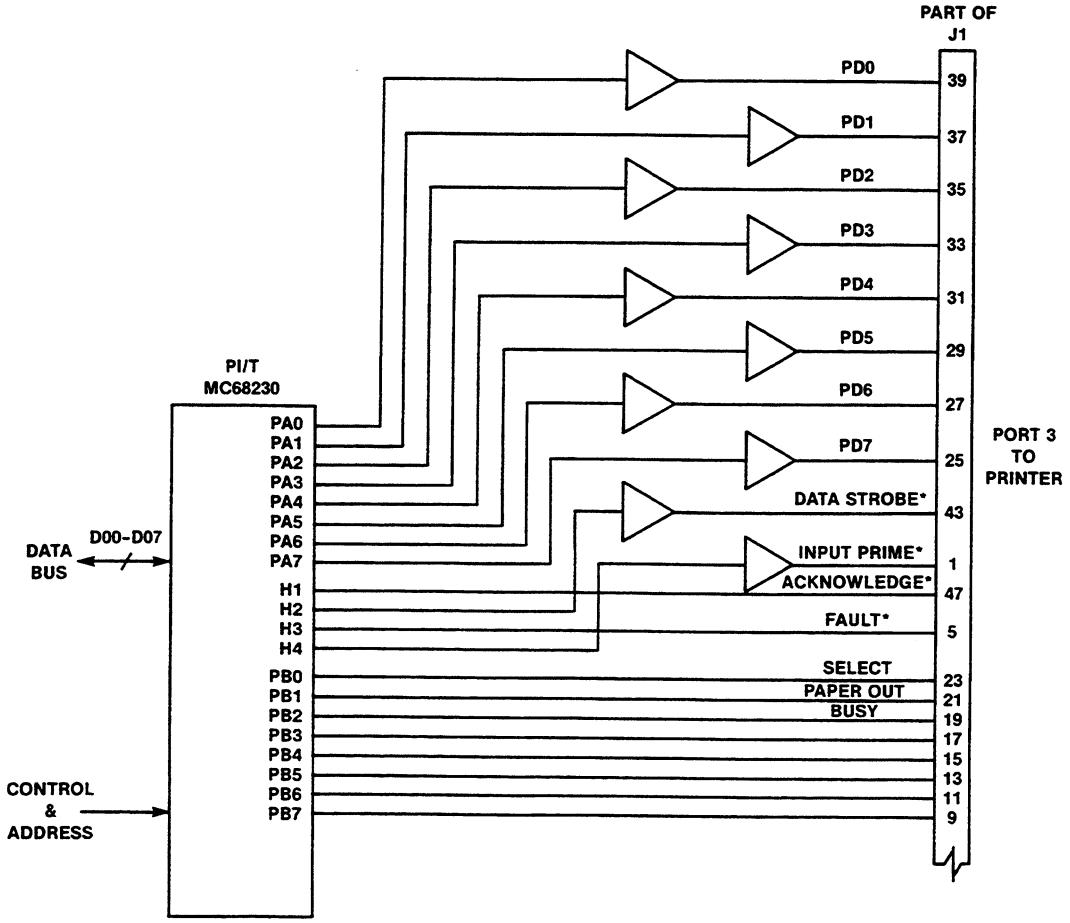


FIGURE 6-5. Printer Interface Port 3 Functional Schematic Diagram

6.3.2 Programming the PI/T

As part of the power-up and reset routines the PI/T registers are initialized, modes are selected, and the direction of data at the various ports is defined. The necessary PI/T registers are shown in Table 6-4. The base address of the MC68230 is $10001. The registers are addressed on the lower half of the MC68000 data bus; i.e., odd address locations. Bits in the Port General Control Register (PGCR) are programmed to select the unidirectional 8-bit mode with all handshake lines active low. The port mode control field of the PGCR should be altered only when H12 enable and H34 enable fields are zero. For this reason, these two fields are initially set to zero when the port mode control field is first programmed. They are later set to one to enable the handshake lines.

The Port Service Request Register is cleared; direct memory access and interrupts are not used. Port A bits are designated as all outputs and Port B as inputs by programming the associated data direction register. A one in any bit indicates an output; zeros indicate inputs. Submodes are determined by the bits of the appropriate port control register. Bit 3 of each of the two port control registers can be a one or a zero depending upon the required state of the H2/H4 handshake lines. Initially, these bits are cleared. The data registers are used to send characters to the printer and to check the status lines. Handshake status bits are displayed in the port status register.

The last step of the initialization sequence is to reset and initialize the printer by pulsing INPUT PRIME* (handshake line H4) low. This is done by setting and then clearing bit 3 of the Port B control register. At this point, the printer is initialized but not selected. It must be selected by the user using the select button; a select code is not sent by the MEX68KECB to automatically select the printer.

Characters can now be sent to the printer. All 7-bit ASCII encoded characters from $20 through $7F, $0D (carriage return), and $0A (line feed) are acceptable. Other codes below $20 are not printable and will be replaced by $2E (period). All 8-bit codes will be masked to 7 bits.

When a printable character has been obtained, it is written to the Port A data register and the printer is strobed to indicate valid data. The status bits PB0-PB2 are then checked. If an error exists the message PRINTER NOT READY will be sent to Port 1. The status lines are monitored until the error condition is removed or the break key is entered at the terminal. After the error is removed, the same byte is sent again. Before the next byte can be sent the PI/T must receive an acknowledge from the printer indicating that it has accepted the data. Bit 0 of the port status register indicates whether either of the Port A output latches can accept new data (ACKNOWLEDGE* received) or whether both latches are full.

TABLE 6-4. PI/T Registers Used in Printer Interface

Register Address	Register Bit 7	6	5	4	3	2	1	0	Register Name	Programmed Value
$10001	Port Mode Control		H34 Enable	H12 Enable	H4 Sense	H3 Sense	H2 Sense	H1 Sense	Port General Control Register	00ØØ 0000
$10003	*	SVCRQ Select		Interrupt PFS		Port Interrupt Priority Control			Port Service Request Register	0000 0000
$10005	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Port A Data Direction Register	1111 1111
$10007	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Port B Data Direction Register	0000 0000
$1000D	Port A Submode		H2 Control			H2 Int Enable	H1 SVCRQ Enable	H1 Stat Ctrl	Port A Control Register	0110 Ø000
$1000F	Port B Submode		H4 Control			H4 Int Enable	H3 SVCRQ Enable	H3 Stat Ctrl	Port B Control Register	1010 Ø000
$10011	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Port A Data Register	----------
$10013	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Port B Data Register	----------
$1001B	H4 Level	H3 Level	H2 Level	H1 Level	H4S	H3S	H2S	H1S	Port Status Register	----------

NOTE: A Ø (zero with a slash) in the programmed value indicates that the bit is programmed with different values depending on operation.

6.4 AUDIO TAPE INTERFACE - PORT 4

The audio tape interface is implemented using the 16-bit timer on the MC68230 and two I/O lines. Information is sent to the tape as a serial bit stream. A digital one is represented by one period of a 50% duty cycle 2000-Hz square wave, whereas, a 1000-Hz 50% duty cycle square wave represents a logic zero. The serial data rate is then somewhere between 1000 and 2000 baud, depending on the bit stream being sent.

Saving and loading programs on tape is discussed in paragraphs 4.5.1 and 4.5.2. In this discussion, the circuitry and operation of the tape interface are discussed in detail.

6.4.1 Data Transfer Baud Rate

As with any data transfer using ASCII encoding, the effective baud rate measured by the time required to transfer a block of data is lower than the data rate on the transmission line. ASCII encoding generates a two-digit byte for every hex digit of data (for example, a 4 becomes ASCII $34). This reduces the transfer rate by one-half. In addition, S-records require overhead bytes such as type of S-record, address of data, number of bytes in the record, and checksums to be sent along with the data. This results in additional baud rate reduction of approximately one-third. The effective baud rate of the tape interface is between 300 and 500 baud, as opposed to the serial transmission rate of 1000 to 2000 baud.

6.4.2 Circuit Operation

The interface circuitry between the MC68230 and the tape player/recorder is shown in Figure 6-6. Port C of the PI/T provides two I/O lines for the tape interface.

Data is sent out via output PC1 of the PI/T. This output drives a voltage divider formed by R1 and R2 and is then AC-coupled to pin 3 of connector J2 designated DATA OUT. The voltage level from the PI/T is reduced by approximately 10 to 1 to avoid overdriving the tape recorder input.

The auxiliary input of the tape recorder is generally used, however, the microphone input will also work with most recorders. In either case, several adjustments should be made prior to recording. If there is no automatic record level control, use the volume control to adjust the record level (about mid-range). If a tone control is present, it should be set to get the best high frequency response. This corresponds to a "tinny" sound for audio recordings. Also, a clean or erased tape provides best results.

Information comes back from the tape player via pin 1 of connector J2 designated DATA IN. This signal passes through the interface circuit shown in Figure 6-6. Comparator U4B is used to square up the slowly changing transitions coming from the tape and produce rapid transitions. Diodes CR1 and CR2 limit the input voltage swing of the comparator. Approximately 450 millivolts of hysteresis is used on the comparator.

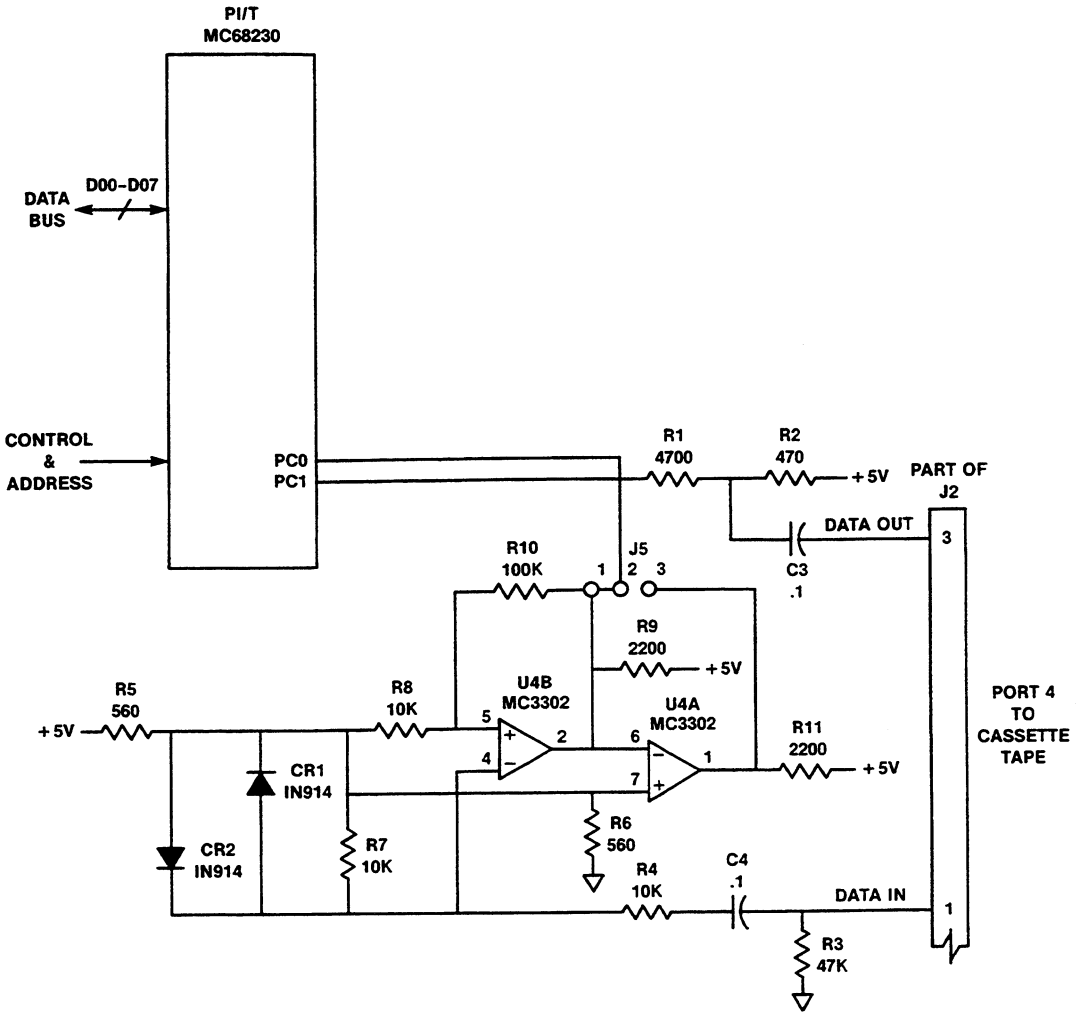


FIGURE 6-6. Audio Tape Interface

The second comparator, U4A, is used to invert the output of U4B. This may or may not be required, depending on the type of tape recorder used. Some tape recorders play back a signal which is inverted from the original input signal; others return a noninverted signal. Comparator U4B inverts the playback signal; if the tape player does not produce a second inversion, one must be produced on the educational computer board using U4A in order to provide the proper signal to the tape driver/receiver firmware. The firmware expects a noninverted signal. Jumper J5 can be used to select the second inversion. See the following paragraph 6.4.3.

6.4.3 Selecting Noninverted Data

Figure 6-7 shows the location of header J5 which is used to provide noninverted data to the tape receiver firmware (refer to Figure 6-6). If it is determined that the tape player does not invert the data, the user must change the polarity of the signal provided to the MC68230. Perform the following steps to invert the data:

a. Cut the signal trace between pin 1 and pin 2 of header J5 on the back side of the printed circuit board. BE CAREFUL — be sure to cut the correct trace; it is approximately 1/8 inch long.

b. Place a plastic cap jumper on header J5 between Pin 2 and Pin 3.

If it is ever desired to restore the signal to the original configuration, the plastic cap jumper can be placed between pin 1 and pin 2.

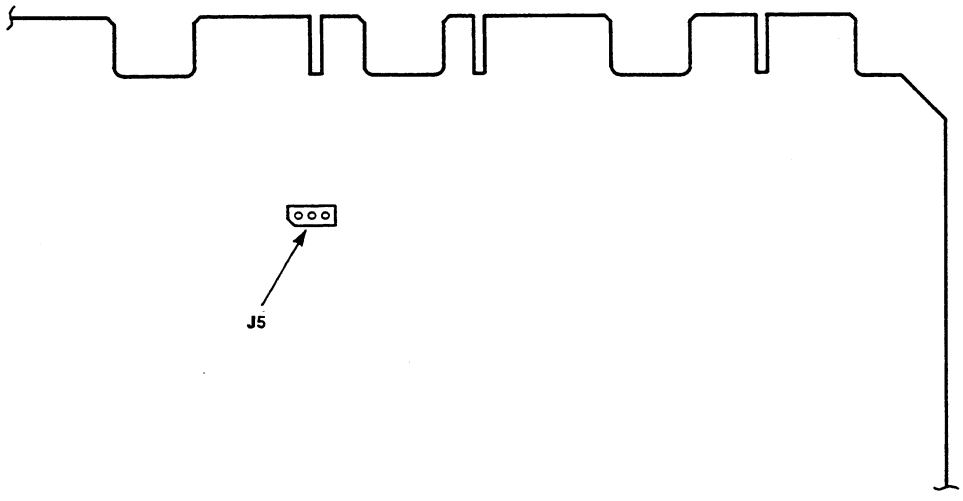


FIGURE 6-7. Header J5 Location

6.4.4 Programming the PI/T

Port C of the PI/T and the on-board timer are used by the tape driver/receiver firmware to send and receive tape data. Bit 1 of Port C (PC1) is specified as an output to transmit data via the data direction register (all other Port C bits are inputs). The 24-bit timer is used to generate and measure the time intervals for the 2000 Hz and 1000 Hz square waves. The timer prescaler is used and is clocked by the falling edge of the 4-MHz system clock.

To transmit data, the following sequence of events occurs. After obtaining the bit that will be sent, the driver firmware loads a count representing one half the required period (i.e., 500 microseconds for 1 kHz and 250 microseconds for 2 kHz) into the counter preload registers. Port C bit 1 is set high and the timer is started. Bit 0 of the timer status register is monitored until the specified time has elapsed, causing this bit to be set. At this point the timer is stopped, PC1 is cleared, and the timer is restarted. The counter is automatically loaded with the contents of the preload registers. The status register is again monitored to determine the end of the specified time period and the completion of the output sequence. Another bit is obtained and the output sequence is repeated.

To receive data, the tape input firmware measures the time between rising edges of the input square wave to determine whether a logic one or zero is being sent. It can now be seen why the polarity of the incoming signal is so important. If the signal is inverted then the elapsed time measured is really the time between the middle of one square wave (falling edge) and the middle of the next square wave of the original wave form. The data would obviously become garbled and lost.

The first step in the input sequence is to initialize the Port C data direction register and then look for a low-to-high transition at PC0. This synchronizes the firmware to the incoming signal and no data is lost because at least one null character (eight zeros) is always recorded before any data.

When the high level is received at PC0, the timer is started using the same mode as the output routine. However, the preload value is different. PC0 is monitored until a low level followed by a high level is received, at which point the timer is stopped and the value in the count register is read and saved. The timer is restarted. The period of the square wave is determined from the difference between the original timer preload value and the count left in the timer when it is halted. Additional bits are received in the same way.

6.5 PI/T TIMER

The MC68230 PI/T contains a 24-bit synchronous down counter that can generate periodic interrupts, a square wave, or a single interrupt after a programmed time period. Also, it can be used for elapsed time measurement.

The PI/T timer is loaded from three 8-bit Counter Preload Registers. The 24-bit counter can be clocked from the output of a 5-bit (divide-by-32) prescaler, or directly from a clock source. The clock source can be the 4 MHz system clock (tied to the CLK input) or an external clock tied to the TIN pin. Several different modes can be programmed by the user.

The counter signals occurrence of an event primarily through zero detection (when the counter value is zero). The zero detect status (ZDS) bit is set in the Timer Status Register (TSR). This is the only bit in the TSR. Also, an interrupt can be generated with the zero detect.

The timer is fully configured and controlled by programming the 8-bit Timer Control Register. It controls (1) the choice between the Port C operation and the timer operation of three timer pins, (2) whether the counter is loaded from the Counter Preload Register or rolls over when zero detect is reached, (3) the clock input, (4) whether the prescaler is used, and (5) whether the timer is enabled.

The contents of the counter can be read via the three count registers. The counter must be stopped to get an accurate reading when accessing these registers.

To summarize, the PI/T timer is fully controllable and available to the user. For detailed information on using the timer, the MC68230 Data Sheet should be referenced. On the Educational Computer Board, the following applies:

a. The timer can be clocked from the 4 MHz system clock or an external clock.

b. The external clock can be connected to pin 5 of connector J2. The maximum allowable clock frequency with an external signal is 4 MHz when using the prescaler and 125 KHz when not using the prescaler.

c. The timer registers are all available from the bus (addresses $010021 through $010035). See Table 6-1.

d. The timer can generate a level 2 interrupt on the MC68000. See paragraph 6.6 for details.

6.6 SYSTEM INTERRUPTS

The I/O devices discussed in this chapter have the ability to generate interrupts when they require attention. Also, the ABORT function is activated via the interrupt structure. Interrupts can be generated on the Educational Computer Board from the following sources:

a. The ABORT switch generates the equivalent of an unmaskable interrupt when activated.

b, Serial Ports - For each MC6850 ACIA, an interrupt can be generated when either the Transmit Data Register is empty requiring another byte of data or the Receive Data Register is full containing a new byte of information. In either case, the interrupt condition will not cause an interrupt unless enabled in the ACIA status register. An enabled interrupt condition pulls the $\overline{\text{IRQ}}$ pin of the ACIA low.

c. Parallel Port 3 - The parallel port of the MC68230 has an independent interrupt capability. The dual function pin $\overline{\text{PIRQ}}$ provides an active low parallel port interrupt request when enabled.

The $\overline{\text{PIRQ}}$ output is activated when any of the status bits associated with handshake lines H1, H2, H3, and H4 goes to a 1. Bits 3, 4, 5, and 6 of the PI/T Port Service Request Register (PSRR) enable and disable the $\overline{\text{PIRQ}}$ and $\overline{\text{PIACK}}$ functions and define whether interrupt or DMA requests are generated from activity on the H1 and H3 handshake lines. Each of the four individual interrupt conditions can be selectively enabled in the Port A and Port B control registers.

d. PI/T Timer - The MC68230 timer can also independently generate an interrupt. TOUT provides an active low timer interrupt request when enabled. The timer generates an interrupt when the 24-bit counter decrements from $000001 to $000000. Bits 7, 6, and 5 of the timer control register (TCR) are used to enable the timer interrupt function and control timer operation.

e. M6800 IRQ - A special auto-vectored interrupt request level is provided for the wirewrap area.

6.6.1 MC68000 Interrupt Structure

The MC68000 recognizes seven interrupt priority levels. The interrupt priority levels are numbered from one to seven with level seven having the highest priority (the equivalent of a non-maskable interrupt). The MC68000 status register contains a three-bit mask which indicates the current processor priority level. Interrupts are inhibited for priority levels less than or equal to the current processor priority. An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request lines with a zero indicating no interrupt requests.

The interrupt priority levels assigned on the MC68000 Educational Computer are shown in Table 6-5. The ABORT button is assigned the level 7, non-maskable interrupt. Level 4 is reserved for an M6800 type bus interface interrupt that can be implemented in the wirewrap area of the board. Level 1 is not assigned.

All interrupts except those generated by the MC68230 are serviced through autovectoring. Autovectoring is used when the interrupting device cannot provide the processor with an exception vector from which the processor can fetch the address of the interrupt service routine. During the processor interrupt acknowledge cycle, VPA* must be asserted to indicate that an internally generated vector is to be used. The processor then generates a vector number which is determined by the interrupt level number. The seven autovector numbers are vector numbers 25 through 31 (decimal). During an interrupt acknowledge cycle for interrupt levels 7, 6, 5, and 4, VPA* is asserted by the ECB hardware and the autovector is used. During an acknowledge cycle for level 3 or 2 interrupts, the interrupting device (MC68230) must provide an 8-bit vector number to the processor.

TABLE 6-5. Interrupt Priority Levels

INTERRUPT LEVEL	INTERRUPTING DEVICE	AUTOVECTOR NUMBER (DECIMAL)
7	ABORT Button*	31
6	ACIA2 (Host)*	30
5	ACIA1 (Terminal)*	29
4	M6800 IRQ*	28
3	PI/T Parallel Ports ($\overline{\text{PIRQ}}$)	Not Used
2	PI/T Timer (TOUT)	Not Used
1	Not Used	Not Used

*Autovectored Interrupts

The 8-bit interrupt vectors associated with $\overline{\text{PIRQ}}$ and TOUT are written into the Port Interrupt Vector Register (PIVR) and the Timer Interrupt Vector Register (TIVR), respectively. Only the upper six bits of the port interrupt vector number are programmed by the user. Each of the four interrupt sources has its own vector which together appear as a contiguous block of four vector numbers whose common upper six bits are programmed in the PIVR. The lower two bits are determined by the interrupt source:

H1 Source - 00
H2 Source - 01
H3 Source - 10
H4 Source - 11

If a vector number is not programmed in the appropriate interrupt vector register before an interrupt occurs, the MC68230 will supply 15 (decimal) for the vector number, where 15 is defined as the uninitialized interrupt vector in the MC68000 exception vector table.

When acknowledging MC68000 compatible vectored interrupts, MC68230 input pins $\overline{\text{PIACK}}$ and $\overline{\text{TIACK}}$ are used. When $\overline{\text{PIACK}}$ or $\overline{\text{TIACK}}$ is asserted and a port or timer interrupt request is being asserted, the PI/T places the corresponding vector on the data bus. The appropriate pin is asserted by the system logic during an interrupt acknowledge cycle. $\overline{\text{PIACK}}$ and $\overline{\text{TIACK}}$ are dual function pins; the appropriate function is selected in the Port Service Request Register or in the Timer Control Register.

6.6.2 Interrupt Software Routines

The TUTOR firmware does not support any interrupts except level 7 which is the ABORT function. Although the ECB hardware can generate interrupts for levels 2 through 6, it is the user's responsibility to provide the interrupt service routines and to initialize the MC68000 exception vector table with the starting addresses of the service routines.

The MC68000 translates an 8-bit vector number into an address as shown in Figure 6-8. The vector number is multiplied by four (shifted left two places) to generate an address within the exception vector table. At this location in RAM, the starting address (32 bits) of the interrupt handler routine is stored. All 8-bit vector numbers — both autovectored and user supplied — are treated in the same way.

A23	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
All Zeros		v7	v6	v5	v4	v3	v2	v1	v0	0	0

FIGURE 6-8. Address Translated from 8-Bit Vector Number

To use the interrupts, the user must enable the interrupts through the appropriate peripheral control registers, program the vector numbers where required on the MC68230, initialize the exception vector table, and, finally, supply the interrupt handler routines. All of these tasks must be part of the user program and its system initialization procedures.

CHAPTER 7

HARDWARE DESCRIPTION

Chapter 7 provides a functional description of the MC68000 Educational Computer Board and detailed information on using the wire-wrap facilities.

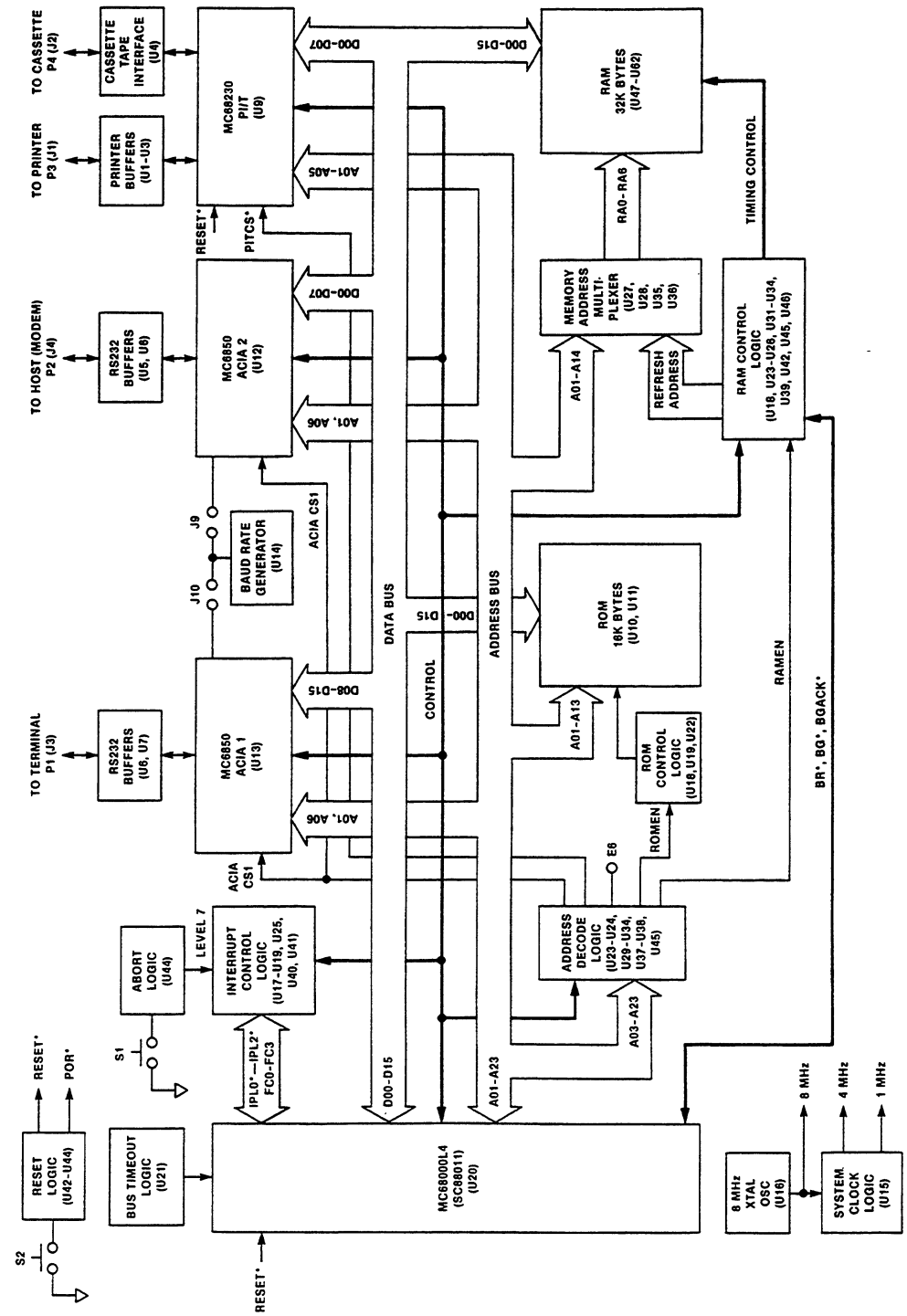


FIGURE 7-1. Block Diagram - Educational Computer Board

CHAPTER 7

HARDWARE DESCRIPTION

7.1 INTRODUCTION

Chapter 7 provides a functional description of the MC68000 Educational Computer Board hardware, including a block diagram. With the description contained here and the schematic drawings of Chapter 8, the user can gain a good understanding of the board's design. Also discussed in this chapter is use of the wire-wrap area of the board. Throughout Chapters 7 and 8, the asterisk (*) is used to denote active low signals.

7.2 FUNCTIONAL DESCRIPTION

The MC68000 Educational Computer Board is a complete microcomputer system built around a 4 MHz processor (MC68000L4). All memory and I/O devices communicate with the processor via a common unbuffered bus structure. The block diagram of the board is shown in Figure 7-1, which illustrates data paths, the addressing scheme, and control logic flow. The functional areas are described in the following paragraphs.

7.2.1 MC68000L4 Microprocessor

The L4 version of the MC68000 is a 4 MHz clock device. The 4 MHz clock is the time base from which all processor timing is derived (see MC68000 data sheet for details). The 4 MHz rate does not relate to bus transaction times directly because these can be either asynchronous in nature or generated by an E clock which is supplied by the MC68000L4 (4 MHz divided by 10 = 400 kHz E clock).

7.2.2 Address Decode

The memory map for the educational computer is shown in Table 7-1. The Address Decode Logic (U23-U24, U29-U34, U37-U38, U45) shown in Figure 7-1 generates enable signals RAMEN, ROMEN, ACIA CS1, and PITCS* in accordance with this memory map. Address lines A03-A23 are decoded and used with the proper control signals to generate these signals.

The RAM is addressed at the bottom of the map ($000007-$007FFF) excluding the first eight locations which contain the initial stack pointer and program counter contents and are stored in ROM. The RAM is divided into two areas; that is, $000008-$0008FF the system area reserved for use by the system firmware, and $000900-$007FFF the user area.

Within the system area, addresses $000000-$0003FF are used for the MC68000 exception vector table. The remaining 1280 bytes (addresses $000400-$0008FF) are used as scratchpad memory for the TUTOR firmware including data buffers, pointers, temporary storage, etc.

The firmware ROM (EPROM) is located just above the RAM in the map at $008000-$00BFFF.

All I/O devices are mapped into the same 64K byte page at $010000-$01FFFF. Redundant mapping occurs within the page (that is, the same device appears at several addresses) because the address is not fully decoded. Chapter 6 contains Tables 6-1 and 6-2 which give the I/O address maps in detail.

Also, a special signal (E6) is provided as a 64K-byte page decode located at $030000-$03FFFF. This signal is intended to allow an M6800 type bus interface via the wire-wrap capability.

TABLE 7-1. Memory Map

FUNCTION			ADDRESS
System Memory	Exception Vector Table	ROM/EPROM RAM	$000000-$000007 (1) $000008-$0003FF
	Tutor Scratchpad	RAM	$000400-$0008FF
User Memory		RAM	$000900-$007FFF
Tutor Firmware		ROM/EPROM	$008000-$00BFFF (1)
Not Used			$00C000-$00FFFF
I/O Devices		PI/T (Lower byte only)	$010000-$01003F
		ACIA2 (Lower byte) & ACIA1 (Upper byte)	$010040-$010043
		Redundant Mapping	↓ $01FFFF
Not Used			$020000-$02FFFF
M6800 Page (E6)			$030000-$03FFFF
Not Used			$040000-$FFFFFF

NOTE: (1) Denotes read only

7.2.3 32K Byte RAM

Sixteen three-supply MCM4116B (16K x 1) devices (U47-U62) make up the dynamic RAM array. These can be accessed either on a byte or word basis, and data transfers to and from the MC68000 use asynchronous bus transfers. The memory access time is approximately 450 nanoseconds and the RAM DTACK* is generated about 500-625 nanoseconds after the start of a read or write cycle.

Operation of the memories is determined by the RAM CONTROL LOGIC (U18, U23-U26, U31-U34, U39, U42, U45, U46). The control logic generates timing control for the RAM devices as well as control signals for the MEMORY ADDRESS MULTIPLEXER (U27, U28, U35, U36). The multiplexer generates row and column addresses from lines A01-A14 during read and write cycles. Refresh addresses are also routed to the memories by the multiplexer during memory refresh.

The DRAM's are completely refreshed once every 1.5 milliseconds on the average (1.9 milliseconds worst case) using a technique called RAS only refresh. When the refresh timer indicates the MCM4116B's need to be refreshed, the RAM CONTROL LOGIC requests control of the MC68000 bus via a Bus Request (BR*) signal. This is a convenient way to prevent the processor from accessing RAM during refresh.

The MC68000 releases the bus and asserts Bus Grant (BG*) in response to the BR*. The RAM CONTROL LOGIC then asserts Bus Grant Acknowledge (BGACK*), releases the BR*, and proceeds with the memory refresh. After the BR* is released, the MC68000 releases BG* and waits for BGACK* to release.

During a refresh cycle, eight rows are refreshed at the rate of one row per microsecond. Sixteen such cycles are required to completely refresh the memory every 1.5-1.9 milliseconds. At the end of each cycle the BGACK* is released, the MC68000 regains control of the bus, and processing proceeds.

7.2.4 16K Byte ROM

The system firmware (TUTOR) is stored in two 64K bit ROM's (U10, U11). MCM68764 EPROM's or MCM68A364 ROM's can be used. Access time for the ROM's can vary from 350 nanoseconds to 450 nanoseconds, depending on the device used.

The system ROM can be read on a byte or word basis. Attempting a write to ROM will result in a bus timeout error. The ROM also uses an asynchronous bus interface with the ROM DTACK* returned 500-625 nanoseconds after ROMEN is received.

7.2.5 Serial Communications Ports

Paragraph 6.2 discusses operation of the serial communications ports in detail. Two MC6850 ACIA's provide the bus interface for the serial ports. ACIA1 (U13) is used for the terminal Port 1 and is connected to bits D08-D15 of the data bus. ACIA2 (U12) is used for the host Port 2 and is connected to bits D00-D07.

The baud rate generator (U14) provides transmit and receive clocks for both ACIA's. Headers J9 and J10 are used to jumper select baud rates varying from 110 to 9600 baud.

Both serial ports are RS-232C compatible. Buffers U5, U6, and U7 translate the ACIA voltage levels to RS-232C interface levels.

The ACIA's are the only devices on the Educational Computer Board that take advantage of the MC68000's M6800 compatible synchronous interface. The ACIA's can only be accessed using a synchronous type of bus transfer. These devices are clocked by a signal called the E clock which is supplied by the MC68000 (E = 4 MHz divided by 10 = 400 kHz). Whenever the address decode signals that either ACIA is to be accessed (VPA* is asserted), the MC68000 synchronizes itself with the E clock and uses a synchronous bus cycle. More detail concerning this mode of operation is given in Section 6 of the MC68000 User's Manual, MC68000UM.

7.2.6 MC68230 PI/T (Printer Interface, Cassette Tape Interface, and Timer)

The MC68230 provides several features on the educational board. The PI/T contains an on-board programmable 24-bit timer. Parallel Ports A and B of the PI/T are buffered to drive a Centronics-compatible printer. Also, Port C of the PI/T is buffered as a cassette tape interface. These are discussed in Chapter 6 of this manual.

The MC68230 is tied to data bus lines D00-D07 and uses the MC68000 asynchronous interface.

7.2.7 Interrupt Control Logic

Devices U17-U19, U25, U40, and U41 compose the INTERRUPT CONTROL LOGIC. The interrupt priority levels and vectoring techniques are discussed in paragraph 6.6. The logic priority encodes the interrupt request and inputs the highest request level to the MC68000 (IPL0*-IPL2*). The logic also monitors the function codes (FC0-FC2) and generates interrupt acknowledge signals for the proper device. ACIA1, ACIA2, ABORT, and the special MC6800 IRQ all require an autovectored interrupt, and the VPA* signal is asserted. Also, when the timer interrupt is acknowledged, TIACK* is asserted to the PI/T. Finally, when the PI/T parallel port interrupt is acknowledged, PIACK* is asserted.

7.2.8 System Clocks

An 8-MHz crystal oscillator (U16) is the time base from which all clock frequencies are derived. Counter U15 is used to generate 4-MHz and 1-MHz clocks. The MC68000 runs from the 4-MHz clock. Control logic and other devices in the system use all three frequencies as time bases.

7.2.9 Bus Timeout Logic

With an asynchronous bus interface, bus timeout logic (U21) must be provided. The timeout logic ends the bus cycle if a device fails to respond within the allotted time, and a bus error is signaled. A device may fail to respond due to circuit failure, addressing a non-used location, or attempting a write cycle to ROM. The bus timeout on the Educational Computer Board is about 10 microseconds long.

7.2.10 System Initialization

The RESET LOGIC (U42-U44) provides system initialization under two modes. Under system power-up, a timer activates both the RESET* and Power On Reset (POR*) signals. RESET* initializes the MC68000 and MC68230. All other timing devices are initialized by POR*.

The second mode is when the reset switch S2 is activated. In this case, only RESET* is activated and thus only the MPU and PI/T are initialized.

It should be noted that the HALT* line to the processor is also activated during power-up. An LED indicator is driven from the HALT* line and lights whenever the processor is in a halt condition.

The initialization sequence for the MC68000 includes loading the supervisory stack pointer and program counter values stored in ROM ($000000-$000007), setting the status register to interrupt level 7, and beginning processing at the PC address. This process starts up the TUTOR firmware package and initializes all system registers and devices.

7.2.11 ABORT Function

Switch S1 activates the ABORT LOGIC (U44). Under this condition, a level 7 non-maskable interrupt is generated, which returns control of the system to TUTOR. The ABORT routine does not reinitialize the system. The ABORT function is useful to regain control of processing without destroying system conditions such as existing register and memory contents.

The interrupt vector to the ABORT firmware is located in RAM. If this vector is altered, the ABORT firmware may not be executed; the results are indeterminate. A user-provided vector can be used. In this case, the user software will be executed when the ABORT button is pressed. RESET will reprogram the vector to point to the TUTOR firmware.

7.3 INTERFACE USING THE WIRE-WRAP AREA

The MC68000 Educational Computer Board provides a small wire-wrap area for those users desiring to do custom interface. The location of the wire-wrap area and the signal connection points are shown in Figure 7-2. The following paragraphs provide a detailed discussion of various aspects of doing custom interface to the ECB.

7.3.1 Wire-wrap Device Mounting Area

An area of approximately 3.5 square inches is provided to mount devices (see detail Figure 7-3). Six rows of holes (24 holes/row) are available to mount sockets or devices. The holes are on one-tenth inch centers and the rows are three-tenths inch apart. The component side of the board has seven metal areas which are tied to the board's ground plane. On the opposite side of the board, metal strips are provided which are tied to +5.0 Vdc power.

With the hole pattern provided, most standard dual-in-line devices can be mounted, up to and including a 48-pin package. This is large enough to accommodate another MC68230 PI/T as an example.

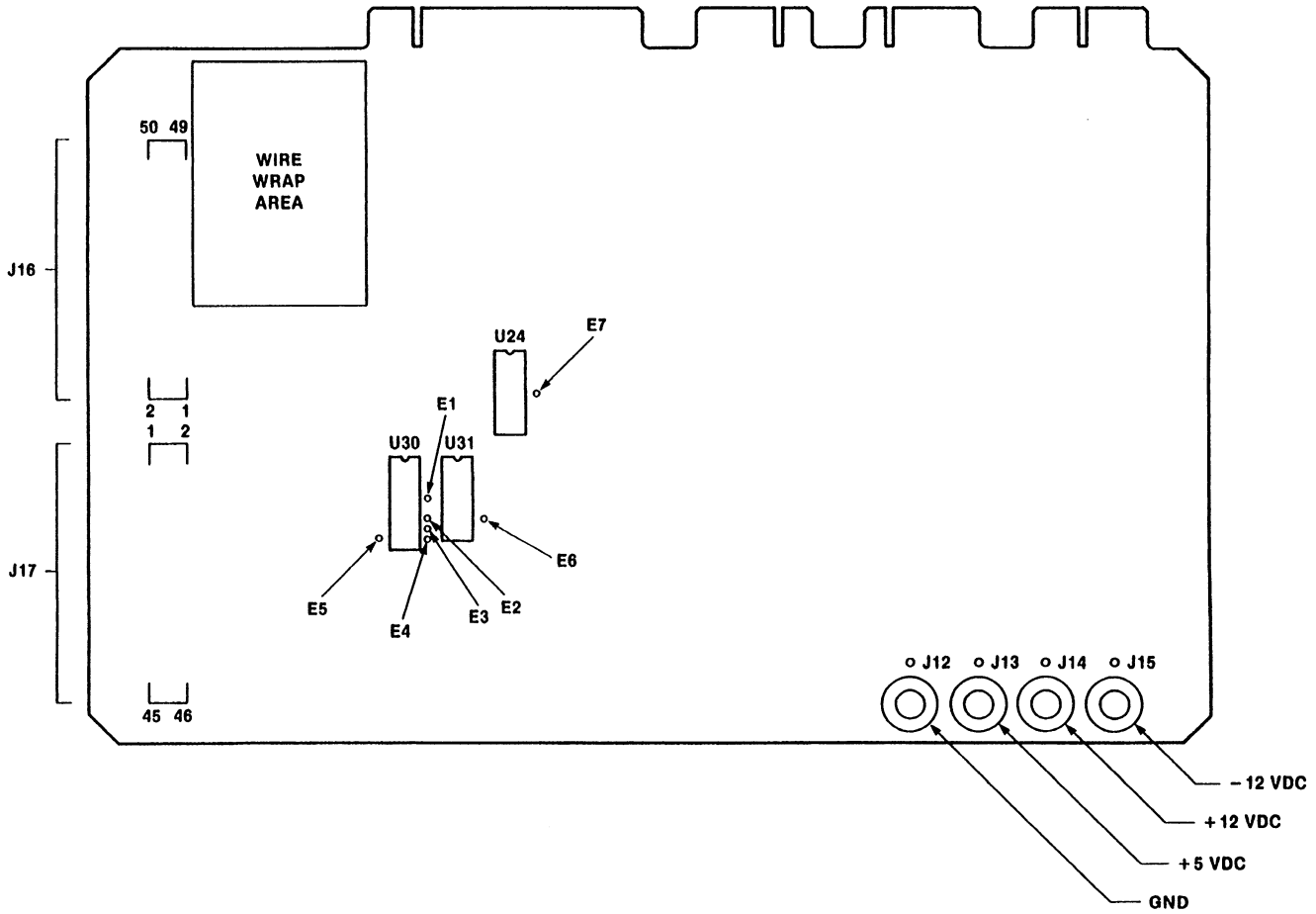


FIGURE 7-2. MEX68KECB Signal Connection Points For Wire-Wrap

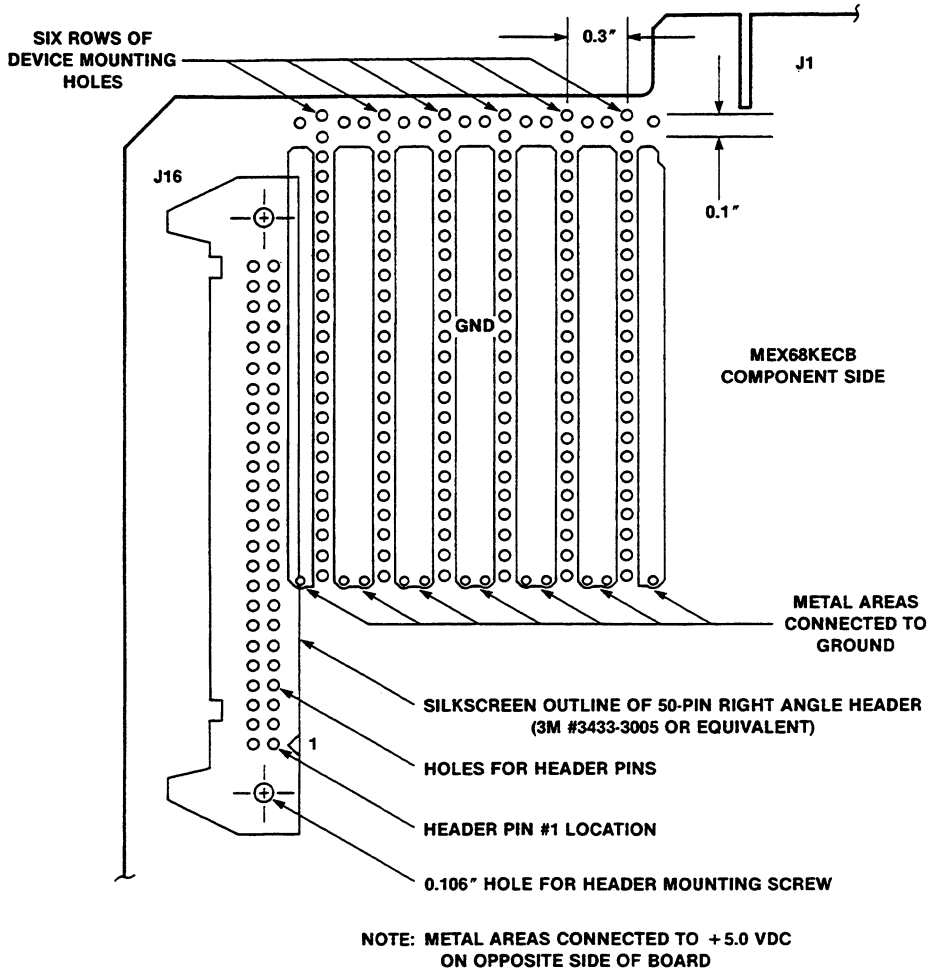


FIGURE 7-3. Detail of Wire-wrap Area

7.3.2 Auxiliary I/O Header J16

In addition to the wire-wrap area, the ECB has provision for an auxiliary I/O header (designated J16). The Figure 7-3 detail shows the location of the header, and Figure 7-4 illustrates the header mounting detail. The hole pattern is designed to accept a 50-pin wrap tail right angle header (standard profile) which is 3M #3433-3005 or equivalent.

When mounting the header as shown in Figure 7-4, two #2 x 3/8 inch screws and two #2 hex nuts are used. The board silkscreen shows an outline of the header. A 50-pin ribbon cable can be connected to this header, and signals are wire-wrapped to the header wrap tails.

7.3.3 MC68000 Bus Signal Connections

A connection area designated J17 gives access to the MC68000 bus signals and system timing. Figure 7-2 shows the location of J17 and gives pin locations. Table 7-2 lists J17 pin number vs. signal designation and also shows attributes of these signal lines.

To use the signal lines, individual wire-wrap pins should be mounted in the holes. These can be soldered in or press fit. Connections are then wrapped to these pins.

7.3.4 Extending System Address Decode

The memory map for the educational computer is given in Table 7-1. When using the wire-wrap area, the designer must not put devices at any of the occupied address locations. To facilitate user address decode, connection points E1 through E6 have been provided on the board, which give enable signals for unused areas of the MC68000 memory map. These connection points include two types:

a. Connection points E1 through E5 (see Figure 7-2 for location) give decode signals for various segments of the MC68000 upper memory map. Figure 7-5 shows the address decode logic and Table 7-3 lists the decoded segments.

   Each signal is low when enabled, and the user can utilize one of these enables as upper address decode. The signal is valid whenever the selected address segment is decoded and AS* is asserted.

b. Connection point E6 is used to select a memory segment for M6800 synchronous interface located at addresses $030000-$03FFFF. Paragraph 7.3.6.1 discusses this in detail.

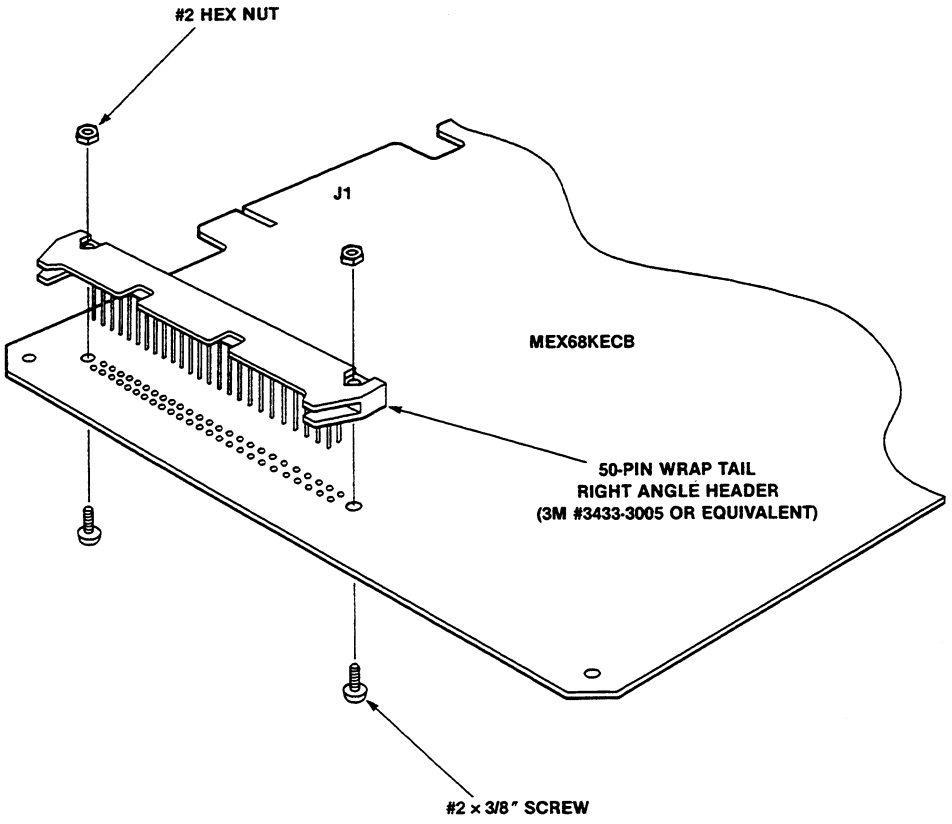


FIGURE 7-4. Auxiliary I/O Header Mounting Detail

TABLE 7-2. J17 Signal Designations

PIN NO.	SIGNAL NAME	DESCRIPTION	4700 ohm PULLUP	THREE-STATE
1	D04	Data Bus Bit 4	Yes	Yes
2	D03	Data Bus Bit 3	Yes	Yes
3	D05	Data Bus Bit 5	Yes	Yes
4	D02	Data Bus Bit 2	Yes	Yes
5	D06	Data Bus Bit 6	Yes	Yes
6	4 MHz CLK	4 MHz System Clock	No	No
7	D07	Data Bus Bit 7	Yes	Yes
8	D14	Data Bus Bit 14	Yes	Yes
9	D08	Data Bus Bit 8	Yes	Yes
10	D15	Data Bus Bit 15	Yes	Yes
11	D09	Data Bus Bit 9	Yes	Yes
12	RESET*	System Reset	Yes	No (O.C.)
13	D10	Data Bus Bit 10	Yes	Yes
14	D01	Data Bus Bit 1	Yes	Yes
15	D11	Data Bus Bit 11	Yes	Yes
16	E	E Clock (400 kHz)	No	No
17	D12	Data Bus Bit 12	Yes	Yes
18	AS*	Address Strobe	Yes	Yes
19	D13	Data Bus Bit 13	Yes	Yes
20	UDS*	Upper Data Strobe	Yes	Yes
21	D00	Data Bus Bit 0	Yes	Yes
22	LDS*	Lower Data Strobe	Yes	Yes
23	A15	Address Bus Bit 15	Yes	Yes
24	R/W*	Read/Write	Yes	Yes
25	A14	Address Bus Bit 14	Yes	Yes
26	A13	Address Bus Bit 13	Yes	Yes
27	A12	Address Bus Bit 12	Yes	Yes
28	FC2	Function Code Bit 2	No	Yes
29	A11	Address Bus Bit 11	Yes	Yes
30	FC1	Function Code Bit 1	No	Yes
31	A10	Address Bus Bit 10	Yes	Yes
32	FC0	Function Code Bit 0	No	Yes
33	A09	Address Bus Bit 9	Yes	Yes
34	A01	Address Bus Bit 1	Yes	Yes
35	A08	Address Bus Bit 8	Yes	Yes
36	A02	Address Bus Bit 2	Yes	Yes
37	A06	Address Bus Bit 6	Yes	Yes
38	A03	Address Bus Bit 3	Yes	Yes
39	A07	Address Bus Bit 7	Yes	Yes
40	A04	Address Bus Bit 4	Yes	Yes
41	A05	Address Bus Bit 5	Yes	Yes
42	DTACK*	Data Transfer Ack. (1)	No	No
43	8 MHz CLK	8 MHz System Clock	No	No
44	6800 IRQ*	M6800 Interrupt Request (2)	Yes	
45	1 MHz CLK	1 MHz System Clock	No	No
46	VMA*	Valid Memory Address	No	Yes

NOTES:

(1) DTACK* cannot have device outputs connected to it. See Paragraph 7.3.5.

(2) 6800 IRQ* is an input only line.

TABLE 7-3. Address Segment Enable Signals for Wire-wrap Users

ENABLE SIGNAL	ADDRESS SEGMENT
E1	$020000-$02FFFF
E2	$040000-$04FFFF
E3	$050000-$05FFFF
E4	$060000-$06FFFF
E5	$070000-$07FFFF
NOTE: Signals are a low TTL level when enabled.	

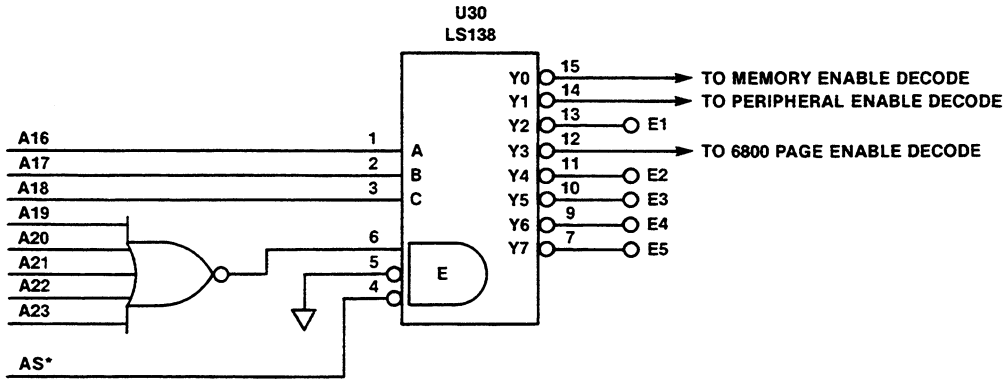


FIGURE 7-5. Address Decode Logic For Memory Map Primary Segments

7.3.5 Asynchronous Bus Interface

Although the MC68000 is capable of doing synchronous bus operations, it is primarily an asynchronous bus machine. The user can interface additional devices to the on-board asynchronous bus via the wire-wrap area; however, care must be taken. The following guidelines apply:

a. The on-board MC68000 bus is unbuffered. The user must not exceed loading requirements of the MC68000.

b. The user must meet timing requirements specified on the MC68000 Data Sheet.

c. Access to the MC68000 signal lines is provided via J17. Special care must be taken with DTACK* (Data Transfer Acknowledge). The processor DTACK* is generated by ANDing DTACK PIT*, DTACK RAM*, and DTACK ROM*, as shown in Figure 7-6. The processor DTACK* goes low whenever any of these go low. The user cannot add another signal to the processor DTACK* because this signal is <u>not</u> an open-collector output.

   A USER DTACK* is connected to the system via connection point E7 as shown in Figure 7-6. The PIT DTACK* is turned off when not required, and the USER DTACK* can be bussed to this point. A 4700 ohm resistor holds PIT DTACK* high when the driver is turned off. The USER DTACK* must be an open-collector or three-state driver. (E7's location is between packages U24 and U25 as shown in Figure 7-2.)

d. If an interrupt capability is required, the user is restricted to an M6800 type autovectored priority level 4 interrupt. See paragraph 7.3.6.2 for usage.

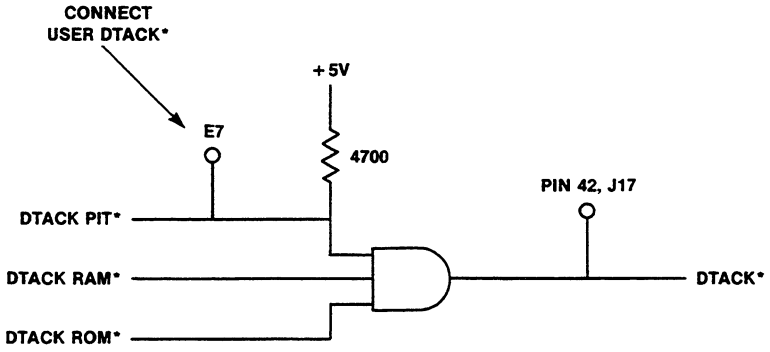


FIGURE 7-6. DTACK* Signal Generation

7.3.6 M6800 Type Synchronous 8-Bit Bus Interface

The MC68000 supports M6800 type synchronous bus transfers through the use of signal lines VMA*, VPA*, and E. Section 6 of the MC68000 User's Manual (MC68000UM) discusses these signal line functions in detail. The user can utilize the synchronous interface on the educational computer. The existing MC6850's also use this interface.

The same guidelines apply concerning bus loading, processor timing, and J17, as mentioned in paragraph 7.3.5. The educational computer also has special provision for user interface into the synchronous bus.

7.3.6.1 M6800 Page Address Decode. A 64K-byte segment of the system memory map is reserved for an M6800 type interface. Connection point E6 is enabled high whenever this page ($030000-$03FFFF) is selected. Connection point E6 can be located on Figure 7-2, and Figure 7-7 shows the logic generating signal E6.

The M6800 page enable E6 is activated when memory page $030000-$03FFFF is selected and both VMA* and LDS* are asserted. The memory page enable is first activated, which, in turn, activates VPA*. After the MC68000 receives VPA*, the processor synchronizes itself to the E clock and continues the bus cycle by asserting VMA*. Signal E6 recognizes that the M6800 page has been selected, VMA* has been asserted for a synchronous cycle, and the LDS* is asserted indicating a bus transfer on the lower eight data bus bits. Thus, the user must interface into the lower eight bits of the data bus when using signal E6.

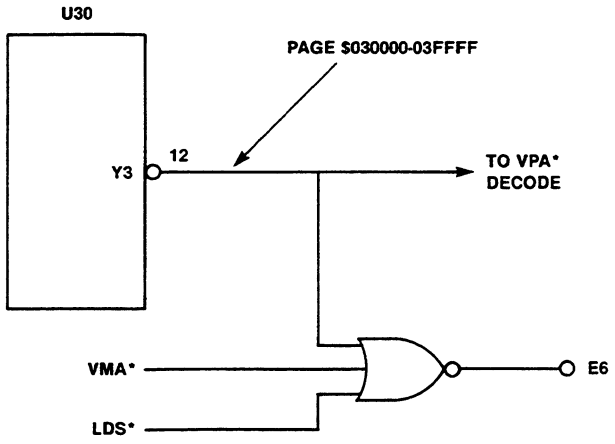


FIGURE 7-7. M6800 Page Address Signal Generation

7.3.6.2 Autovectored Interrupt Level 4. To facilitate an M6800 type interface, the educational computer also provides an autovectored interrupt request via Pin 44 of J17. When the interrupt request line is asserted (taken low), the MC68000 receives a level 4 priority interrupt. A level 4 interrupt acknowledge cycle from the MC68000 causes an autovectored response with the vector number equal to 28 (decimal) or $1C (hex).

The interrupt request must be held asserted until the interrupt service routine clears the interrupt request. The user must supply the interrupt service routine in his software, and he must also initiate the exception vector table at address $000070.

The M6800 interrupt request can also be used for devices on the asynchronous bus interface. The user must only follow the same rules for use and be aware the response is an autovectored interrupt.

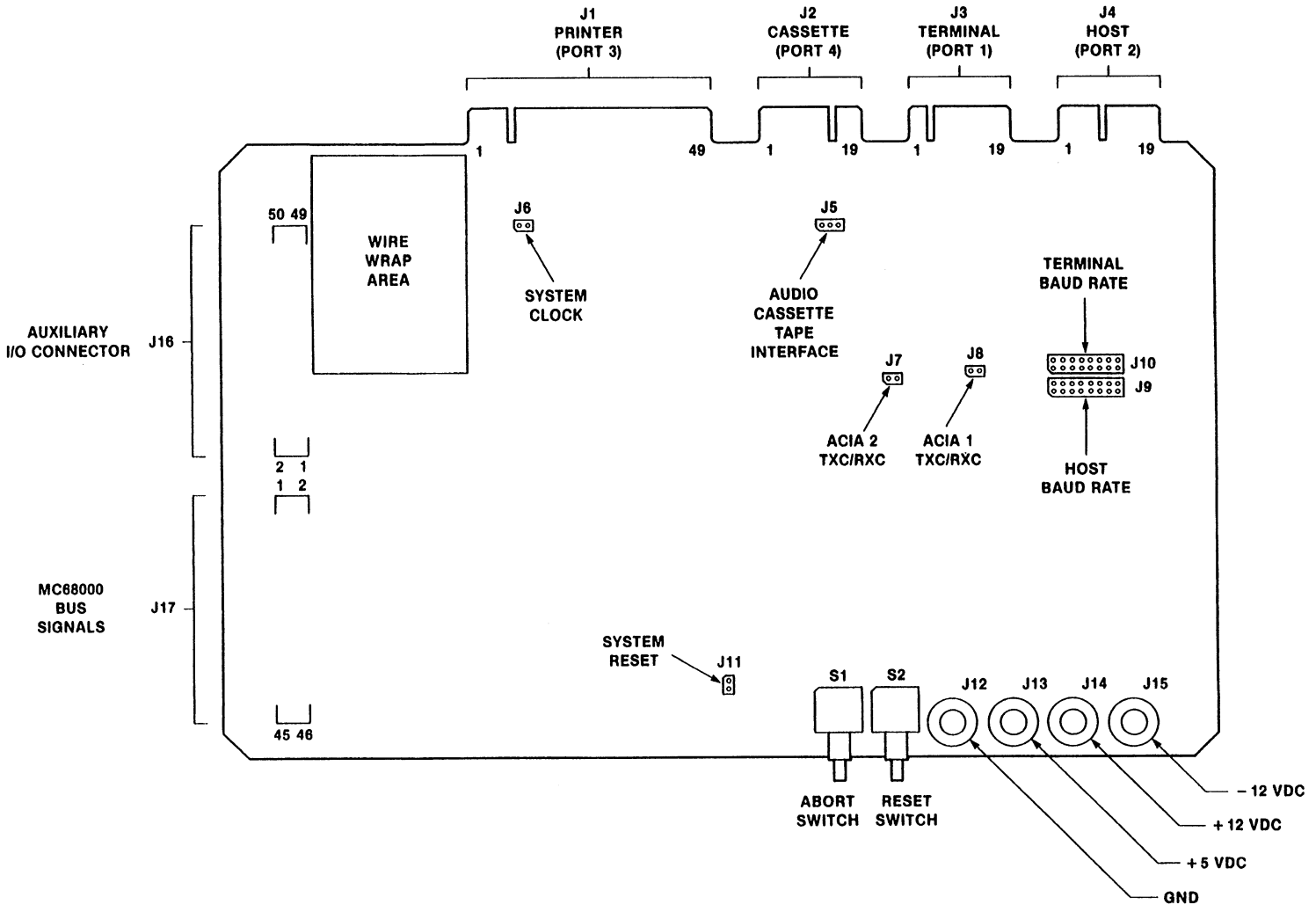


FIGURE 8-1. MEX68KECB Header, Connector, and Switch Locations

CHAPTER 8

SUPPORT INFORMATION

8.1 INTRODUCTION

This chapter provides the interconnection signals, parts list, and schematic diagrams for the MC68000 Educational Computer Board.

8.2 CONNECTOR SIGNAL DESCRIPTIONS

Tables 8-1 through 8-4 give pin numbers, signal mnemonics, and signal names and descriptions for connectors J1 through J4.

8.3 JUMPER HEADER, CONNECTOR, AND SWITCH LOCATIONS

Figure 8-1 shows the MEX68KECB jumper header, connector, and switch locations. Table 8-5 lists the connection "J" numbers and gives the appropriate manual paragraph where each is described. The only jumper not discussed elsewhere in the manual is SYSTEM RESET header J11. If pins 1 and 2 of the header are jumpered together, a total system reset occurs (HALT*, POR*, and RESET* are activated). This is normally used only for test purposes.

8.4 PARTS LIST

Table 8-6 lists the components of the MEX68KECB. Figure 8-2 illustrates part locations. The parts list reflects the latest issue of hardware at the time of printing.

8.5 DIAGRAMS

Figure 8-3 shows the schematic diagram for the MEX68KECB MC68000 Educational Computer Board.

TABLE 8-1. Connector J1 Printer Port 3 Pin Assignments

PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
9	PB7	Port B, bit 7 - Unbuffered data line for PI/T Port B
11	PB6	Port B, bit 6 - Unbuffered data line for PI/T Port B
13	PB5	Port B, bit 5 - Unbuffered data line for PI/T Port B
15	PB4	Port B, bit 4 - Unbuffered data line for PI/T Port B
17	PB3	Port B, bit 3 - Unbuffered data line for PI/T Port B
19	BUSY (PB2)	BUSY - Signal from printer when high indicates that printer is busy (connected to PB2).
21	PAPER OUT (PB1)	PAPER OUT - Signal from printer when high indicates that printer is out of paper (connected to PB1).
23	SELECT (PB0)	SELECT - Signal from printer when low indicates printer is deselected (connected to PB0).
1	INPUT PRIME*	INPUT PRIME - Buffered output to printer when low causes printer input buffer to be cleared and printer logic to be initialized.
5	FAULT*	FAULT - Signal from printer when low indicates fault condition.
43	DATA STROBE*	DATA STROBE - Buffered output to printer when low indicates valid data on PD0-PD7.
47	ACKNOWLEDGE*	ACKNOWLEDGE - Signal from printer when low indicates printer has accepted data on PD0-PD7.
25	PD7	Printer data, bit 7 - Buffered data output to printer (connected to PI/T Port A)
27	PD6	Printer data, bit 6 - Buffered data output to printer (connected to PI/T Port A)
29	PD5	Printer data, bit 5 - Buffered data output to printer (connected to PI/T Port A)
31	PD4	Printer data, bit 4 - Buffered data output to printer (connected to PI/T Port A)
33	PD3	Printer data, bit 3 - Buffered data output to printer (connected to PI/T Port A)

TABLE 8-1. Connector J1 Printer Port 3 Pin Assignments (cont'd)

PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
35	PD2	Printer data, bit 2 - Buffered data output to printer (connected to PI/T Port A)
37	PD1	Printer data, bit 1 - Buffered data output to printer (connected to PI/T Port A)
39	PD0	Printer data, bit 0 - Buffered data output to printer (connected to PI/T Port A)
All even pins plus 3,7,41, 45,49	GND	GROUND

TABLE 8-2. Connector J2 Audio Cassette Tape Interface Port 4 Pin Assignments

PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
1	DATA IN	DATA IN - Data input to tape interface logic. Connected to tape recorder output for data playback.
3	DATA OUT	DATA OUT - Data output from tape interface logic. Connected to tape recorder microphone or auxiliary input to record data.
5	TIN	TIMER IN - Input to PI/T timer that can be used as external clock source or clock enable.
7	PC4	PI/T Port C, bit 4
9,11, 13,15, 17,19	NC	Not connected
2,4,6, 8,10,12, 14,16, 18,20	GND	GROUND

TABLE 8-3. Connector J3 Serial Communications Port 1 (To Terminal) Pin Assignments

PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
3	TX DATA	TRANSMITTED DATA - Serial data signal from terminal to educational board.
5	RX DATA	RECEIVED DATA - Serial data signal to terminal from educational board.
9	CTS	CLEAR TO SEND - Control signal to terminal. Activated by DTR on educational computer.
11	DSR	DATA SET READY - Control signal to terminal. Activated by DTR on educational computer.
14	DTR	DATA TERMINAL READY - Control signal from terminal indicating terminal is on-line.
15	DCD	SIGNAL DETECT - Control signal to terminal. Activated by DTR on educational computer.
13	GND	Signal ground.
1,2,4,6,7,8, 10,12,16,17, 18,19,20	NC	Not connected.

TABLE 8-4. Connector J4 Serial Communications Port 2 (To Host/Modem) Pin Assignments

PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
3	TX DATA	TRANSMITTED DATA - Serial data signal to host/modem from educational board.
5	RX DATA	RECEIVED DATA - Serial data signal from host/modem to educational board.
7	RTS	REQUEST TO SEND - Control signal to host/modem. Always high level.
9	CTS	CLEAR TO SEND - Control signal from host/modem. Indicates host/modem can accept transmitted data.
14	DTR	DATA TERMINAL READY - Control signal to host/modem. Indicates educational board is on-line and ready.
13	GND	Signal ground.
1,2,4,6,8, 10,11,12, 15,16,17, 18,19,20	NC	Not connected.

TABLE 8-5. MEX68KECB Connector and Header Manual References

DESIGNATION	NAME	REFERENCE PARAGRAPH
J1	Printer connector - Port 3	2.5.1
J2	Cassette tape connector - Port 4	2.5.3
J3	Terminal connector - Port 1	2.3.1
J4	Host connector - Port 2	2.5.2.2
J5	Audio cassette tape interface	6.4.3
J6	System clock	2.2.3
J7	ACIA2 TXC/RXC	2.5.2.1
J8	ACIA1 TXC/RXC	2.2.4.2
J9	Host baud rate	2.5.2.1
J10	Terminal baud rate	2.2.4.1
J11	System reset	8.3
J12	Ground	2.2.2.2
J13	+5 Vdc	
J14	+12 Vdc	
J15	-12 Vdc	
J16	Auxiliary I/O connector	7.3.2
J17	MC68000 bus signals	7.3.3

TABLE 8-6. MEX68KECB Parts List

REFERENCE DESIGNATION	MOTOROLA PART NUMBER	DESCRIPTION
	84-W8111B01	Printed wiring board, MEX68KECB
C1-C20, C22-C24,C26	21SW992C025	Capacitor, ceramic, .100 uF @ 50 Vdc
C21	21SW992C014	Capacitor, ceramic, .010 uF @ 50 Vdc
C25	21NW9604A60	Capacitor, ceramic, 1000 pF @ 50 Vdc
C27	21NW9604A11	Capacitor, ceramic, .47 uF @ 50 Vdc
C28-C58	21NW9702A09	Capacitor, ceramic, .1 uF @ 50 Vdc
C59	23NW9704A23	Capacitor, tantalum, .33 uF @ 35 Vdc
C60,C62	23NW9618A33	Capacitor, electrolytic, 22 uF @ 25 Vdc
C61	23NW9618A09	Capacitor, electrolytic, 100 uF @ 16 Vdc
CR1,CR2	48NW9616A03	Diode, silicon, 1N4148/1N914
CR3	48NW9612A24	Diode, light emitting, red
J5	28NW9802D86	Header, single row, 3-pin
J6-J8,J11	28NW9802D01	Header, double row, 2-pin
J9,J10	28NW9802B34	Header, double row, 16-pin
R1,R17	06SW-124A65	Resistor, film, 4.7k ohm, 5%, 1/4 W
R2	06SW-124A41	Resistor, film, 470 ohm, 5%, 1/4 W
R3	06SW-124A89	Resistor, film, 47k ohm, 5%, 1/4 W
R4,R7,R8	06SW-124A73	Resistor, film, 10k ohm, 5%, 1/4 W
R5,R6	06SW-124A43	Resistor, film, 560 ohm, 5%, 1/4 W
R9,R11	06SW-124A57	Resistor, film, 2.2k ohm, 5%, 1/4 W
R10	06SW-124A97	Resistor, film, 100k ohm, 5%, 1/4 W
R12	51NW9626A51	Resistor SIP, five 27k ohm
R13,R16,R30,R36	51NW9626A47	Resistor SIP, seven 4.7k ohm
R14,R15,R29,R35	51NW9626A41	Resistor SIP, nine 4.7k ohm

TABLE 8-6. MEX68KECB Parts List (cont'd)

REFERENCE DESIGNATION	MOTOROLA PART NUMBER	DESCRIPTION
R18-R28	06SW-124A17	Resistor, film, 47 ohm, 5%, 1/4 W
R31	06SW-124A92	Resistor, film, 62k ohm, 5%, 1/4 W
R32	06SW-124A74	Resistor, film, 11k ohm, 5%, 1/4 W
R33,R34	06SW-124B22	Resistor, film, 1.0M ohm, 5%, 1/4 W
R37	06SW-124A29	Resistor, film, 150 ohm, 5%, 1/4 W
R38	06SW-124B50	Resistor, film, 15M ohm, 5%, 1/4 W
S1,S2	40NW9801A54	Switch, push, SPDT, momentary contact
S1	38NW9404B96	Switch cap, red, medium
S2	38NW9404A56	Switch cap, black, medium
U1	51NW9615D27	I.C. SN74S32N
U2,U3,U34	51NW9615C24	I.C. SN74LS32N
U4	51NW9615B75	I.C. MC3302
U5,U7	51NW9615B29	I.C. MC1488L
U6	51NW9615B30	I.C. MC1489AL
U8,U25,U44	51NW9615E91	I.C. SN74LS00N
U9	51NW9615H45	I.C. MC68230L8
U10	51AW4129B09	Programmable I.C., U10, TUTOR
U11	51AW4129B10	Programmable I.C., U11, TUTOR
U12,U13	51NW9615B94	I.C. MC6850P
U14	51NW9615B54	I.C. MC14411P
U15	51NW9615H47	I.C. SN74LS93N
U16	48AW1068B04	Crystal oscillator, 8.0 MHz, 1%
U17	51NW9615F05	I.C. SN74LS20N
U18,U32	51NW9615C21	I.C. SN74LS04N
U19	51NW9615F35	I.C. SN74LS21N

TABLE 8-6. MEX68KECB Parts List (cont'd)

REFERENCE DESIGNATION	MOTOROLA PART NUMBER	DESCRIPTION
U20	51NW9615H81	I.C. SC88011L (MC68000L4)
U21,U22,U39	51NW9615F16	I.C. SN74LS175N
U23	51NW9615C22	I.C. SN74LS08N
U24,U45	51NW9615F76	I.C. SN74LS11N
U26	51NW9615F38	I.C. SN74LS393N
U27,U28,U35,U36	51NW9615E84	I.C. SN74LS153N
U29,U37	51NW9615E89	I.C. SN74LS260N
U30	51NW9615C69	I.C. SN74LS138N
U31	51NW9615E77	I.C. SN74LS27N
U33	51NW9615C20	I.C. SN74LS02N
U38	51NW9615E88	I.C. SN74LS10N
U40	51NW9615G10	I.C. SN74LS148N
U41	51NW9615F52	I.C. SN74LS273N
U42	51NW9615C60	I.C. MC3456P
U43	51NW9615A90	I.C. MC7405P
U46	51NW9615C25	I.C. SN74LS74N
U47-U61	51NW9615H86	I.C. MCM4116BP-30
VR1	51NW9615H08	I.C. MC79L05ACP
Y1	48BW1357X01	Crystal 1.8432 MHz
	09NW9811A04	Socket, I.C., D.I.L., 16-pin
	09NW9811A02	Socket, I.C., D.I.L., 14-pin
	09NW9811A15	Socket, I.C., D.I.L., 24-pin
	09NW9811A30	Socket, I.C., D.I.L., 64-pin
	29NW9805B17	Jumper, shorting insulated
	28NW9802E35	Banana jack, .250" mounting hole
	04SW995A014	Washer, interlocking, .250"

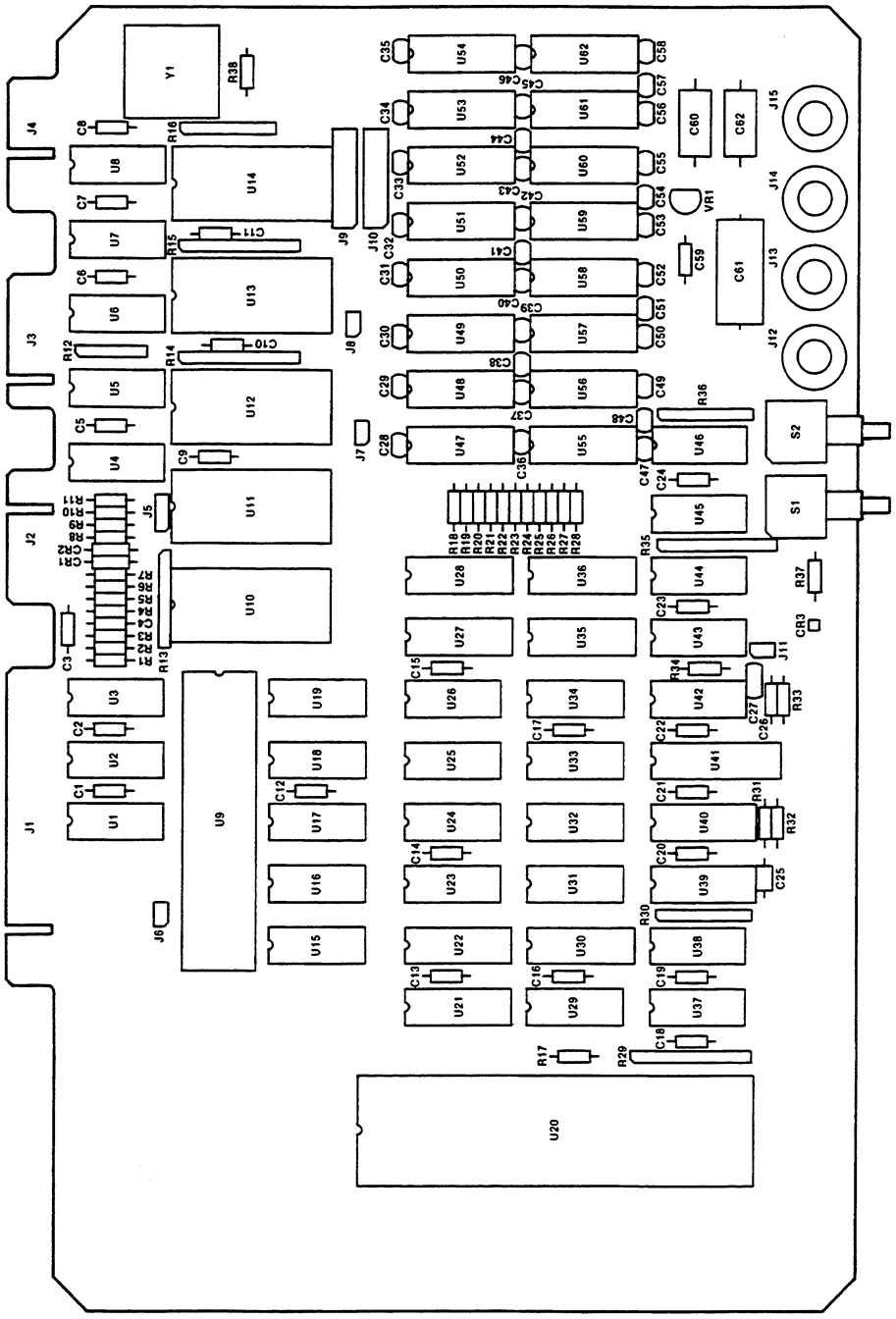


FIGURE 8-2. MEX68KECB Parts Location Diagram

APPENDIX A

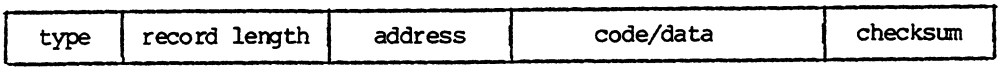
S-RECORD OUTPUT FORMAT

The S-record format for output modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can thus be visually monitored and the S-records can be more easily edited.

S-RECORD CONTENT

When viewed by the user, S-records are essentially character strings made of several fields which identify the record type, record length, memory address, code/data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The 5 fields which comprise an S-record are shown below:



where the fields are composed as follows:

FIELD	PRINTABLE CHARACTERS	CONTENTS
type	2	S-record type — S0, S1, etc.
record length	2	The count of the character pairs in the record, excluding the type and record length.
address	4, 6, or 8	The 2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
code/data	0-2n	From 0 to n bytes of executable code, memory-loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record).
checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

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's manual for that program must be consulted. TUTOR, the firmware supplied with the educational computer, supports S0, S1, S2, S8, and S9 records. The S2 and S8 records are not often used however because all of the on-board RAM and ROM can be addressed with a 2-byte address.

An S-record-format module may contain S-records of the following types:

S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. Under VERSAdos, the resident linker's IDENT command can be used to designate module name, version number, revision number, and description information which will make up the header record. The address field is normally zeroes.

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

S2 A record containing code/data and the 3-byte address at which the code/data is to reside.

S3 A record containing code/data and the 4-byte address at which the code/data is to reside.

S5 A record containing the number of S1, S2, and S3 records transmitted in a particular block. This count appears in the address field. There is no code/data field.

S7 A termination record for a block of S3 records. The address field may optionally contain the 4-byte address of the instruction to which control is to be passed. There is no code/data field.

S8 A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.

S9 A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. Under VERSAdos, the resident linker's ENTRY command can be used to specify this address. If not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. S7 and S8 records are usually used only when control is to be passed to a 3- or 4-byte address. Normally, only one header record is used, although it is possible for multiple header records to occur.

CREATION OF S-RECORDS

S-record-format programs may be produced by several dump utilities, debuggers, VERSAdos' resident linkage editor, or several cross assemblers or cross linkers. On EXORmacs, the Build Load Module (MBLM) utility allows an executable load module to be built from S-records, and has a counterpart utility in BUILDS, which allows an S-record file to be created from a load module.

Several programs are available for downloading a file in S-record format from a host system to an 8-bit microprocessor-based or a 16-bit microprocessor-based system. Programs are also available for uploading an S-record file to or from an EXORmacs system.

EXAMPLE

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

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

The module consists of one S0 record, four S1 records, and an S9 record.

The S0 record is comprised of the following character pairs:

S0	S-record type S0, indicating that it is a header record.
06	Hexadecimal 06 (decimal 6), indicating that six character pairs (or ASCII bytes) follow.
00 00	Four-character 2-byte address field, zeroes in this example.
48 44 52	ASCII H, D, and R - "HDR".
1B	The checksum.

The first S1 record is explained as follows:

S1	S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.
13	Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.
00 00	Four-character 2-byte address field; hexadecimal address 0000, where the data which follows is to be loaded.

The next 16 character pairs of the first S1 record are the ASCII bytes of the actual program code/data. In this assembly language example, the hexadecimal opcodes of the program are written in sequence in the code/data fields of the S1 records:

OPCODE	INSTRUCTION	
285F	MOVE.L	(A7)+,A4
245F	MOVE.L	(A7)+,A2
2212	MOVE.L	(A2),D1
226A0004	MOVE.L	4(A2),A1
24290008	MOVE.L	FUNCTION(A1),D2
237C	MOVE.L	#FORCEFUNC,FUNCTION(A1)

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

2A The checksum of the first S1 record.

The second and third S1 records each also contain $13 (19) character pairs and are ended with checksums 13 and 52, respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The S9 record is explained as follows:

S9 S-record type S9, indicating that it is a termination record.

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

00
00 The address field, zeroes.

FC The checksum of the S9 record.

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted. For example, the first S1 record above is sent as:

type				length				address								code/data								...	checksum			
S		1		1		3		0		0		0		0		2		8		5		F		...	2		A	
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

APPENDIX B

OPERATION WITH MECHANICAL AND LOW SPEED TERMINALS

Difficulties may be encountered when the MEX68KECB is tied to a mechanical terminal at Port 1. Mechanical terminals are inherently slower than CRT terminals when performing certain functions such as a carriage return because of the physical movement required. The paper printout used with mechanical terminals also presents problems which are not encountered with a CRT terminal. These problems are discussed in this appendix.

INITIALIZATION SEQUENCE FOR MECHANICAL TERMINALS

When a mechanical terminal is interfaced to the educational computer, an added initialization sequence is required. Using the Port Format command for Port 1 (PF1), the user must change the number of null characters sent after each character and/or after each line. Without the correct number of nulls, the TUTOR prompt may or may not be displayed; in some cases, only the last part of the prompt will be displayed. Other transmissions from the educational computer may also be garbled. Mechanical terminals need to receive a number of null characters after each carriage return/line feed and, in some cases, after each character to allow their mechanism to catch up; that is, a carriage return/line feed (CR/LF) sequence requires more time than two printable characters, and the additional nulls fill in the extra time. Without the nulls, part of the message (or prompt) is lost during the CR/LF sequence.

At lower baud rates, mechanical terminals usually require nulls only after a CR/LF and not after each character. For example, a TI 700 Series terminal requires only CR/LF nulls at 110, 150, and 300 baud (refer to Table B-1). When only CR/LF nulls are required, characters are missed at the beginning of each line but the rest of the line is received correctly. At higher baud rates where nulls are required after each character, all characters are unrecognizable until the nulls have been added. The entire line will be garbled without the nulls.

Although received lines can be garbled, the lines transmitted by the terminal do not require nulls and are not garbled. The Port Format command should be used to specify the number of null characters required. All user entries under the PF command (paragraph 3.5.21), including carriage returns, should be entered, regardless of how much of the educational computer's response is received; in some cases, the response may be unintelligible until all parameters have been entered. Table B-1 lists the number of nulls required by a TI 700 series terminal at various baud rates. The number of nulls required by other terminals must be determined by the user.

After all parameters have been entered, several carriage returns or a character followed by a carriage return will be required before the prompt is displayed on terminals which require nulls after each character. Terminals which require only CR/LF nulls should display the prompt as soon as the PF command is complete.

NOTE

A reset changes the Port Format parameters back to their initial values. The user must go through the above initialization sequence any time the RESET button is pushed or after a power-on reset.

TABLE B-1. TI 700 Series Null Requirements

BAUD RATE	CHARACTER NULLS	CR/LF NULLS
110	0	1
150	0	1
300	0	4
1200	3	17
2400	7	2F

PAPER PRINTOUT FOR MECHANICAL TERMINALS

Mechanical terminals suffer from a second difficulty, which is caused by the paper printout. CRT terminals allow erasures and overwrites, whereas paper-listing terminals do not.

The educational computer assembler/disassembler utilizes the overwrite capability of CRT terminals when inputting source lines. Under the Memory Modify command with the disassemble option, bytes in memory are read and the disassembled source line is displayed. The user may enter a new source line, which will result in the original address, object code, and source code being erased and replaced by the new address, object code, and source code. On a paper-type terminal, no erasure is possible; the two lines are written on top of each other and neither is legible.

The same problem occurs at the printer when it is attached. When using a CRT terminal, erasure and overwrite produce a current assembly listing where old source lines are not interspersed with new lines. To get a readable listing using a paper-listing terminal, make all changes and then use the Memory Display command with the disassemble option to produce the listing.

If an error is made when entering a source line using a paper-type terminal, the error indicator, which appears under the field suspected of causing the error, is of little use because the fields of the source line are illegible. This problem can be overcome by forcing the ECB to generate an auto line feed each time a carriage return is entered so that it will not overwrite. The third byte of the 6-byte OPTIONS variable is the auto line feed/no auto line feed flag (refer to paragraph 3.5.21). This byte is initialized to $00, indicating no auto line feed. To force the auto line feed mode, this byte should be set to a non-zero value. The old source line, new source line, and error indicator will now all be legible.

Any utilities supported by a host which use erasures or other screen control commands will cause the same type of problem when the educational computer is operated in the transparent mode.

TERMINAL BAUD RATES

The educational computer will operate at baud rates which range from 110 to 9600. A major source of problems is selecting a baud rate for Port 1 or Port 2 which is not the same as the baud rate of the terminal or host which is connected to the port. The baud rate selected on the educational computer must match the baud rate of the terminal or host connected to the educational computer. However, the baud rate of the terminal and of the host do not need to be the same except when the board is operated in the transparent mode.

When the educational computer is operated in the transparent mode, the terminal port (Port 1) and the host port (Port 2) are tied together. The educational computer is effectivly bypassed. Its only function in the transparent mode is to monitor the information sent by the terminal for the exit character. In the transparent mode, the terminal and host baud rates must be identical.

When using the educational computer in the trace mode or with breakpoints, at relatively low baud rates, it may be desirable to suppress the register display. The fourth byte of the OPTIONS variable (paragraph 3.5.21) determines whether or not the registers will be displayed. This byte is initialized to $00, indicating that the registers will be displayed. If this byte is set to a non-zero value, the registers will not be displayed after each instruction is traced or when a breakpoint is encountered. The registers can be examined, however, with the DF command. The register display flag is set to zero by RESET.

Example:

```
TUTOR  1.X > T
PHYSICAL ADDRESS=00001000
PC=00001004 SR=2700=.S7..... US=FFFFFFFF SS=00000786
D0=00306D4D D1=00000000 D2=00000000 D3=00000000
D4=00306D4D D5=0000002C D6=00000002 D7=00000000
A0=00010040 A1=00000618 A2=000004B8 A3=00000540
A4=00001006 A5=00000540 A6=00000541 A7=00000786
--------------------001004    6D1C                BLT.S    $001022

TUTOR  1.X :> MM 4E9                               Set non-zero register display
0004E9    00 ?F0.                                  flag.

TUTOR  1.X > T
PHYSICAL ADDRESS=00001004
001006    0C000039            CMP.B   #57,D0

TUTOR  1.X :>
```

APPENDIX C

RS-232C SERIAL COMMUNICATIONS

E.I.A. RS-232C STANDARD

Written in 1969 by the Electronic Industries Association (EIA), RS-232C is a serial communications standard established to define electrical and mechanical requirements for interconnecting data communications equipment (DCE) and data terminal equipment (DTE). The standard describes both synchronous and asynchronous serial binary communications with data rates ranging from zero to 20,000 bits/second. Twenty-five signal lines are described by the standard, although most are not used in typical applications. Table C-1 summarizes key features of the E.I.A. RS-232C standard.

TABLE C-1. E.I.A. RS-232C Standard

PARAMETER	RS-232C
Line length (recommended maximum - may be exceeded with proper design.)	50 ft.
Input Z	3k to 7k ohm 2500 pF
Maximum frequency (baud)	20k baud
Transition time (time in undefined area between "1" and "0") tr = 10 to 90%	4% of bit period or 1 ms
dV/dt (wave shaping)	30 V/us
Mark (Data "1") Space (Data "0")	-3 V +3 V
Common mode voltage (for balanced receiver)	-
Output Z	-
Open-circuit output voltage (V_o)	3 V < $\|V_o\|$ <25V
V_t = loaded V_o	5 < $\|V_o\|$ <15V 3k to 7k ohm load
Short circuit current	500 mA
Power-off leakage (V_o applied to unpowered device)	> 300 ohm 2 V < $\|V_o\|$ <25V V_o applied
Minimum receiver input for proper V_o	> ± 3 V

The standard connector used for RS-232C compatible equipment is a 25-signal subminiature "D" type. Table C-2 lists the pin number, signal name, and signal description.

TABLE C-2. RS-232C Signal Description

RS-232C PIN NUMBER	RS-232C SIGNAL NAME	DESCRIPTION AND SIGNAL DIRECTION
1	AA	Frame ground
2	BA	Transmitted data (to DCE)
3	BB	Received data (from DCE)
4	CA	Request to send (to DCE)
5	CB	Clear to send (from DCE)
6	CC	Data set ready (from DCE)
7	AB	Signal ground
8	CF	Received line signal detector (from DCE)
9	—	Positive DC test voltage
10	—	Negative DC test voltage
11	—	Unassigned
12	SCF	Secondary received line signal detector (from DCE)
13	SCB	Secondary clear to send (from DCE)
14	SBA	Secondary transmitted data (to DCE)
15	DB	Transmitter signal element timing (from DCE)
16	SBB	Secondary received data (from DCE)
17	DD	Receiver signal element timing (from DCE)
18	—	Unassigned
19	SCA	Secondary request to send (to DCE)
20	CD	Data terminal ready (to DCE)
21	CG	Signal quality detector (from DCE)
22	CE	Ring indicator (from DCE)
23	CH/CI	Data rate selector (to/from DCE)
24	DA	Transmitter signal element timing (to DCE)
25	—	Unassigned

MEX68KECB RS-232C INTERFACE

Ports 1 and 2 of the MC68000 Educational Computer Board support asynchronous serial communications as described by the RS-232C standard. Because transmit and receive clocks are not sent out on the interface, synchronous communications are not supported. Port 1 constitutes a DCE or modem interface type; that is, data terminal equipment is connected to Port 1. Port 2 is a DTE interface and connects to data communication equipment. Baud rates at each port range from 110 to 9600 baud.

Of the 25 signal lines described in Table C-2, the educational computer supports a set of seven. These are:

BA - Transmitted data - TxDATA
BB - Received data - RxDATA
CA - Request to send - RTS
CB - Clear to send - CTS
CC - Data set ready - DSR
CF - Received line signal detector - DCD
CD - Data terminal ready - DTR

In addition, there are two ground signals:

AB - Signal ground - GND
AA - Frame ground

The frame ground is not connected to the educational computer's signal ground, but can be connected externally if necessary. Tables 8-3 and 8-4 list the pin number, signal mnemonic, and signal description for Port 1 connector J3 and Port 2 connector J4, respectively.

The following paragraphs are a description of the signal lines supported by Ports 1 and 2. The format used is:

Signal name
Signal direction
Signal function description

1. TxDATA Transmitted data

   Serial data output from terminal (DTE) to modem (DCE)

   The line/signal through which the terminal (DTE) sends data to the modem (DCE).

2. RxDATA Received data

   Serial data input to terminal (DTE) from modem (DCE)

   The line/signal through which the modem (DCE) sends data to the terminal (DTE).

CHAPTER 8

SUPPORT INFORMATION

3. RTS Request to send

Control output from terminal (DTE) to modem (DCE).

The line/signal through which the terminal (DTE) requests permission to transmit data to the modem (DCE).

Assertion of RTS instructs the DCE to prepare to receive data from the DTE and to signify that it is ready to receive by asserting CTS. However, RTS is not monitored at Port 1; it is assumed that Port 1 is always ready to receive data. CTS is activated any time the terminal (DTE) asserts DTR indicating that it is ready to transmit or receive data.

At Port 2 RTS is asserted upon power up to prepare DCE connected to Port 2 for data reception.

4. CTS Clear to send

Control input to terminal (DTE) from modem (DCE).

The line/signal through which the modem (DCE) acknowledges the acceptance of a terminal (DTE) request to send data.

As stated above, Port 1 asserts CTS any time an active level is received from the DTE on DTR.

Port 2 receives CTS from the DCE connected to Port 2 and will interrupt data transmission when inactive.

5. DSR Data set ready

Control input to terminal (DTE) from modem (DCE).

The line/signal through which the modem (DCE) indicates its on-line, in-service, or active status.

Port 1 activates DSR whenever an active level is received on DTR. Port 1 is always on-line.

Port 2 uses only the CTS input from the DCE to indicate whether data may be sent. DSR is not used in making the decision.

6. DTR Data terminal ready

Control output from terminal (DTE) to modem (DCE).

The line/signal through which the terminal (DTE) indicates its on-line, in-service or active status.

$\overline{\text{DTR}}$ is used by Port 1 to enable and disable the transmission of data via the $\overline{\text{CTS}}$ input of the Port 1 ACIA — MC6850. When $\overline{\text{CTS}}$ is driven high, transmission will stop following the completion of any in-process transmission. Port 2 activates DTR as part of the power-up/reset firmware. A write to the ACIA control register which causes the $\overline{\text{RTS}}$ output to go low will activate DTR at Port 2.

7. DCD Signal Detect

Control input to terminal (DTE) from modem (DCE).

The line/signal through which the modem (DCE) indicates that the communication channel to which the modem (DCE) interfaces (the other/non-terminal side of the modem) is in an acceptable active state. This signal has meaning only in a communication channel (i.e., telephone line) context. DCD is off when no signal is being received or when the received signal is unsuitable for demodulation. While Port 1 implements a DCE or modem interface, the communications is exclusively digital. There is no need to test the suitability of the signal. DCD, at Port 1, indicates only that DTR has been received from the DTE.

Port 2 does not monitor DCD. Again, all signals are exclusively digital.

MEX68KECB NON-COMPLIANCE WITH RS-232C

In addition to being a functional subset of the full RS-232C standard, the educational computer does not comply strictly to the signal specifications. In addition to signal definition and timing, the RS-232C standard specifies driver, receiver, and interface voltage and impedance levels (refer to Table C-1). The MC6850 ACIA's used in the serial interface are NMOS devices operating with a +5V supply and cannot meet the interface voltage and impedance requirements. Two linear integrated circuits, the MC1488 RS-232C line driver and the MC1489A RS-232C line receiver, provide the required buffering and drive to meet the specifications.

The maximum rate of voltage change is specified as 30 V/us in the RS-232C standard. The MC1488 line drivers have an inherent slew rate which is much too fast. The current limited output of the device can be used to control this slew rate by connecting a capacitor to each driver output. The required capacitance value is given by the formula:

$$C = I_{os} \times \frac{\Delta T}{\Delta V}$$

where C is the capacitance in picofarads, I_{os} is the output short-circuit current in microamps, and $\Delta T/ \Delta V$ is 1/slew rate in microseconds per volt. A 330 pF capacitor on each output of the MC1488 will guarantee a worst case slew rate of 30 V/us. Bear in mind, however, that this capacitance includes cabling capacitance.

These capacitors are not present on the educational computer board.

1ATX35944-1 PRINTED IN USA 8/98 IMPERIAL LITHO 37490 500 LITTRAIN

NOTES:

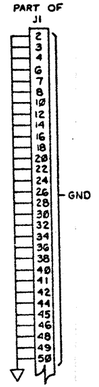
1. FOR REFERENCE DRAWINGS REFER TO BILL OF MATERIAL 01-W3111B01
2. UNLESS OTHERWISE SPECIFIED:
   ALL RESISTORS ARE IN OHMS, ±5 PCT, 1/4 WATT.
   ALL CAPACITORS ARE IN UF.
   ALL VOLTAGES ARE DC.
3. INTERRUPTED LINES CODED WITH THE SAME LETTER OR LETTER COMBINATIONS ARE ELECTRICALLY CONNECTED.
4. ⚠ DEVICE TYPE NUMBER IS FOR REFERENCE ONLY. THE NUMBER VARIES WITH THE MANUFACTURER.
5. J16 CUSTOMER USE OPTION (50 PINS). DEVICE TYPE NUMBERS AND CONNECTIONS NOT SHOWN ON SYMBOL ARE LISTED BELOW. UNDERLINED PORTION OF TYPE NUMBER IS USED AS A CODE TO IDENTIFY DEVICES ON DIAGRAM.

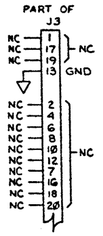
REF DES	TYPE ⚠		GND	+5V	-5V	+12V	-12V
U1	74 S32		7	14			
U2	74LS32		7	14			
U3	74LS32		7	14			
U4	MC3302		12	3			
U5	MC1488		7			14	1
U6	MC1489A		7	14			
U7	MC1488		7			14	1
U8	74LS08		7	14			
U9	MC68230		38	12			
U10	MCM6876A		12	24			
U11	MCM6876A		12	24			
U12	MC6850		1	12			
U13	MC6850		1	12			
U14	MC14411		12	24			
U15	74LS93		10	5			
U16			7	14			
U17	74LS20		7	14			
U18	74LS04		7	14			
U19	74LS21		7	14			
U20	MC68000L8		56,53	54,49			
U21	74LS175		8	16			
U22	74LS175		8	16			
U23	74LS08		7	14			
U24	74LS11		7	14			
U25	74LS00		7	14			
U26	74LS393		7	14			
U27	74LS153		8	16			
U28	74LS153		8	16			
U29	74LS260		7	14			
U30	74LS138		8	16			
U31	74LS27		7	14			
U32	74LS04		7	14			
U33	74LS02		7	14			
U34	74LS32		7	14			
U35	74LS153		8	16			
U36	74LS153		8	16			
U37	74LS260		7	14			
U38	74LS10		7	14			
U39	74LS175		8	16			
U40	74LS148		8	16			
U41	74LS273		10	20			
U42	MC3456		7	14			
U43	MC7405		7	14			
U44	74LS00		7	14			
U45	74LS11		7	14			
U46	74LS74		7	14			

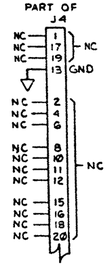
POWER/GROUND TABLE CONT'D

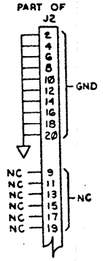
REF DES	TYPE ⚠		GND	+5V	-5V	+12V	-12V
U47	MCM4116		16	9	1	8	
U48	MCM4116		16	9	1	8	
U49	MCM4116		16	9	1	8	
U50	MCM4116		16	9	1	8	
U51	MCM4116		16	9	1	8	
U52	MCM4116		16	9	1	8	
U53	MCM4116		16	9	1	8	
U54	MCM4116		16	9	1	8	
U55	MCM4116		16	9	1	8	
U56	MCM4116		16	9	1	8	
U57	MCM4116		16	9	1	8	
U58	MCM4116		16	9	1	8	
U59	MCM4116		16	9	1	8	
U60	MCM4116		16	9	1	8	
U61	MCM4116		16	9	1	8	
U62	MCM4116		16	9	1	8	

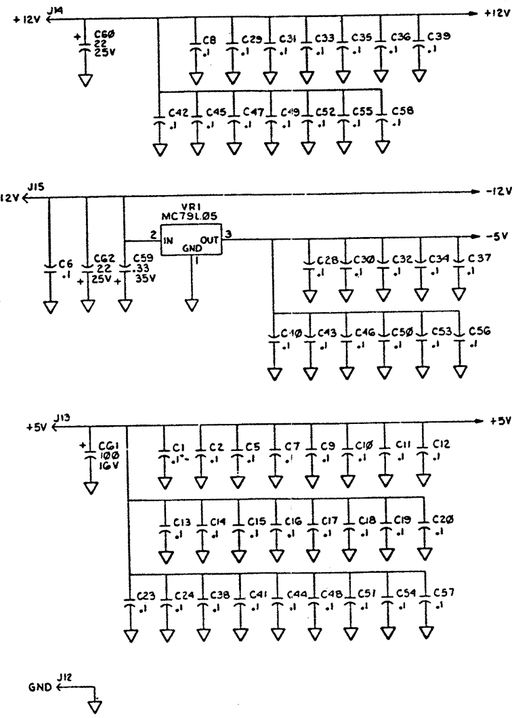
Y1	
VR1	
U62	
S2	
R38	
J17	
E7	
CR3	
CG2	
HIGHEST NUMBER USED	NOT USED
REFERENCE	DESIGNATIONS











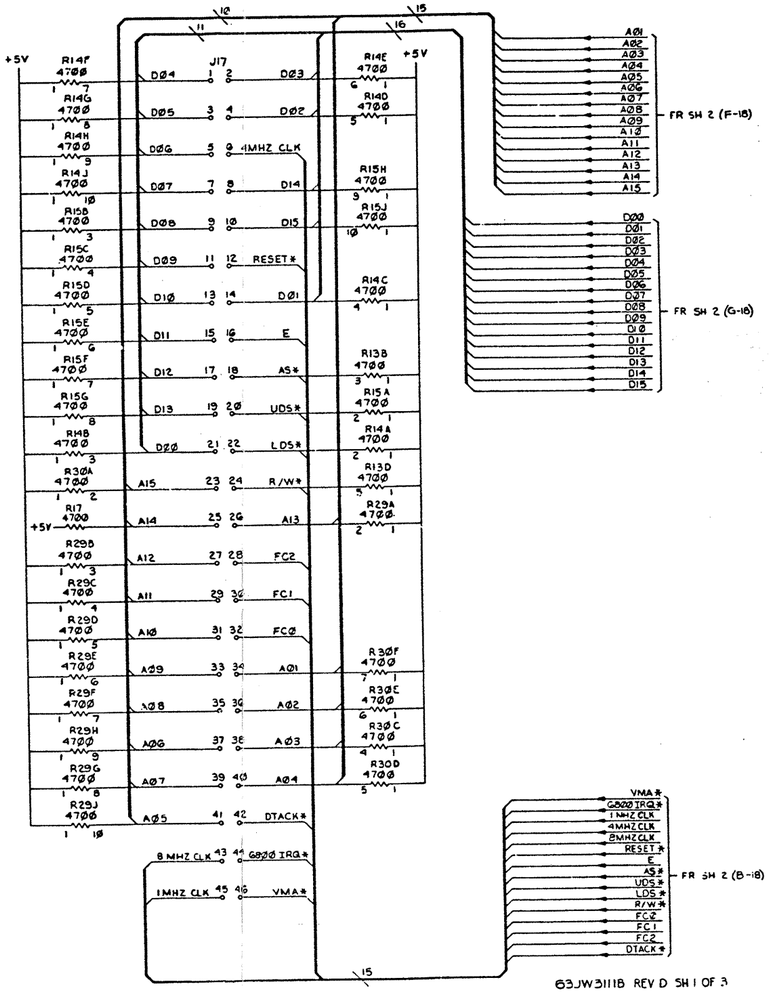


FIGURE 8-3. MEX68KECB MC68000 Educational Computer Board Schematic Diagram (Sheet 1 of 3)

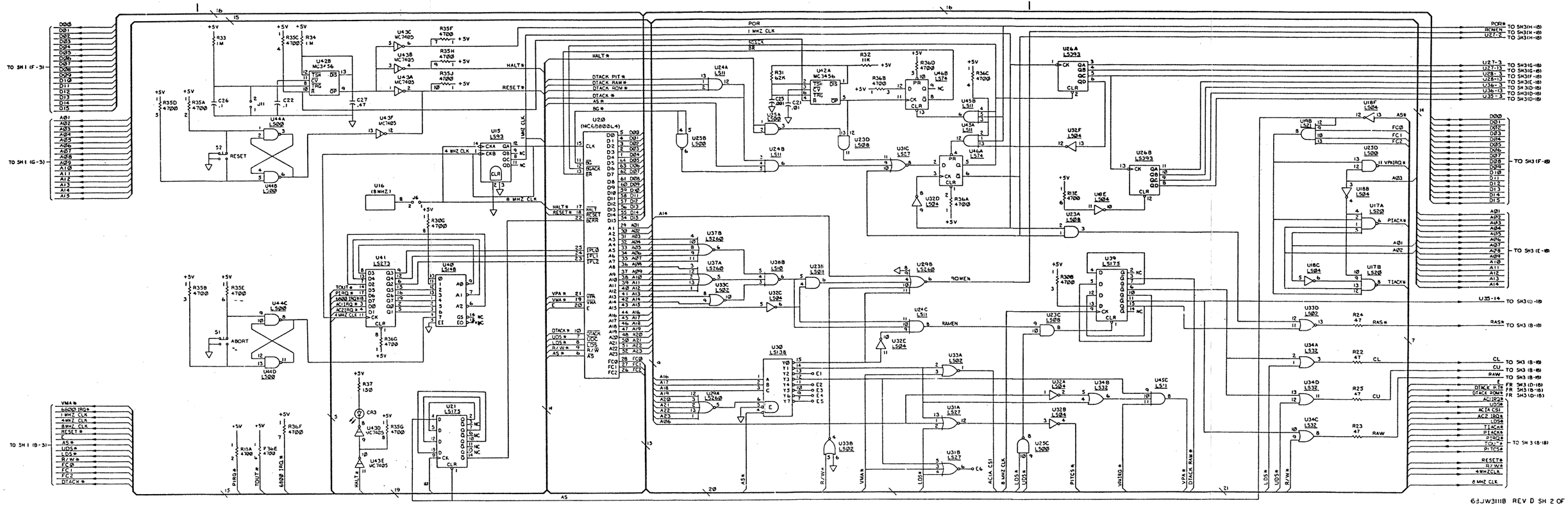


FIGURE 8-3. MEX68KECB MC68000 Educational Computer Board Schematic Diagram
(Sheet 2 of 3)

8-15/8-16

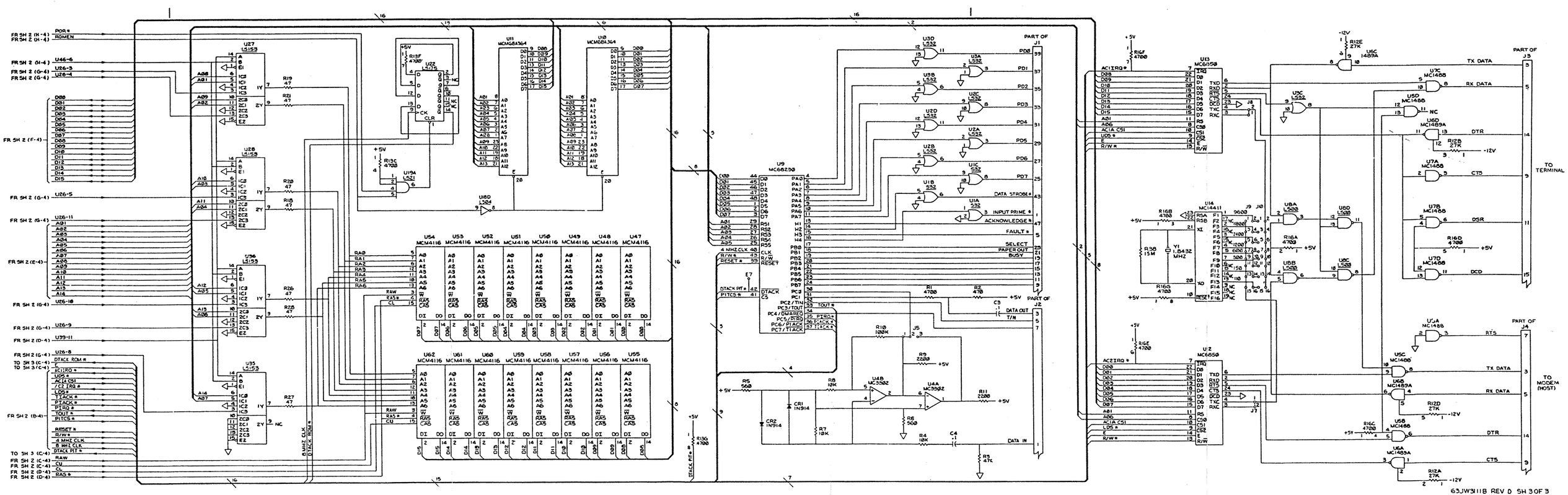


FIGURE 8-3. MEX68KECB MC68000 Educational Computer Board Schematic Diagram
(Sheet 3 of 3)